Stefano Spaccapietra (Ed.)

Journal on Data Semantics IV

Springer

Volume Editor

Stefano Spaccapietra
EPFL-IC-IIF-LBD, INJ 236 (Bâtiment INJ), Station 14
1015 Lausanne, Switzerland
E-mail: stefano.spaccapietra@epfl.ch

Library of Congress Control Number: 2005937594

CR Subject Classification (1998): H.2, H.3, I.2, H.4, C.2

ISSN	0302-9743
ISBN-10	3-540-31001-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-31001-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11603412 06/3142 5 4 3 2 1 0

The LNCS Journal on Data Semantics

Computerized information handling has changed its focus from centralized data management systems to decentralized data exchange facilities. Modern distribution channels, such as high-speed Internet networks and wireless communication infrastructure, provide reliable technical support for data distribution and data access, materializing the new, popular idea that data may be available to anybody, anywhere, anytime. However, providing huge amounts of data on request often turns into a counterproductive service, making the data useless because of poor relevance or inappropriate level of detail. Semantic knowledge is the essential missing piece that allows the delivery of information that matches user requirements. Semantic agreement, in particular, is essential to meaningful data exchange.

Semantic issues have long been open issues in data and knowledge management. However, the boom in semantically poor technologies, such as the Web and XML, has boosted renewed interest in semantics. Conferences on the Semantic Web, for instance, attract crowds of participants, while ontologies on their own have become a hot and popular topic in the database and artificial intelligence communities.

Springer's *LNCS Journal on Data Semantics* aims at providing a highly visible dissemination channel for the most remarkable work that in one way or another addresses research and development on issues related to the semantics of data. The target domain ranges from theories supporting the formal definition of semantic content to innovative domain-specific application of semantic knowledge. This publication channel should be of the highest interest to researchers and advanced practitioners working on the Semantic Web, interoperability, mobile information services, data warehousing, knowledge representation and reasoning, conceptual database modeling, ontologies, and artificial intelligence.

Topics of relevance to this journal include:

- Semantic interoperability, semantic mediators
- Ontologies
- Ontology, schema and data integration, reconciliation and alignment
- Multiple representations, alternative representations
- Knowledge representation and reasoning
- Conceptualization and representation
- Multimodel and multiparadigm approaches
- Mappings, transformations, reverse engineering
- Metadata
- Conceptual data modeling
- Integrity description and handling
- Evolution and change
- Web semantics and semi-structured data
- Semantic caching
- Data warehousing and semantic data mining
- Spatial, temporal, multimedia and multimodal semantics

- Semantics in data visualization
- Semantic services for mobile users
- Supporting tools
- Applications of semantic-driven approaches

These topics are to be understood as specifically related to semantic issues. Contributions submitted to the journal and dealing with semantics of data will be considered even if they are not within the topics in the list.

While the physical appearance of the journal issues is like the books from the well-known Springer LNCS series, the mode of operation is that of a journal. Contributions can be freely submitted by authors and are reviewed by the Editorial Board. Contributions may also be invited, and nevertheless carefully reviewed, as in the case for issues that contain extended versions of the best papers from major conferences addressing data semantics issues. Special issues, focusing on a specific topic, are coordinated by guest editors once the proposal for a special issue is accepted by the Editorial Board. Finally, it is also possible that a journal issue be devoted to a single text.

The journal published its first volume in 2003 (LNCS 2800), its second volume at the beginning of 2005 (LNCS 3360), and its third volume in Summer 2005 (LNCS 3534). The first two volumes are special issues composed of selected extended versions of the best conference papers. The third volume is a special issue on "Semantic-Based Geographical Information Systems", coordinated by guest editor Prof. Esteban Zimányi. This fourth volume is the first "normal" volume, consisting of spontaneous submissions on any of the topics of interest to the journal. Currently planned volumes include a special issue on emergent semantics.

The Editorial Board comprises one Editor-in-Chief (with overall responsibility) and several members. The Editor-in-Chief has a four-year mandate to run the journal. Members of the board have a three-year mandate. Mandates are renewable. More members may be added to the board as appropriate.

We are happy to welcome both readers and authors, and hope we will share this privileged contact for a long time.

Stefano Spaccapietra
Editor-in-Chief
http://lbdwww.epfl.ch/e/Springer/

JoDS Volume IV - Preface

This fourth JoDS volume is the outcome of the selection of papers spontaneously submitted to the journal, in particular in response to a Call for Papers issued on July 17, 2004. The call invited submissions on any topic that falls within the scope of the journal.

Altogether, 38 submissions were reviewed. After the first round of reviews, 24 submissions were asked to perform a major revision and resubmit. Most of these were actually resubmitted and went through a second round of reviews, with the same reviewers as allocated for the first round. Eventually, 10 papers were accepted for publication, after some last modifications suggested by the reviewers.

Accepted papers cover a wide range of topics, from traditional data semantics (information modeling, data model transformation, knowledge representation, data and schema integration) to the newest trends (multimedia, Semantic Web annotation, information extraction, and knowledge discovery).

A new Call for Papers is open at the moment for a volume to appear in 2006. We are looking forward to your contributions.

Stefano Spaccapietra
Editor-in-Chief

In Memoriam

Prof. Maurizio Panti

On July 3, 2005, at 9:45AM, Prof. Maurizio Panti, Head of the Department of Computer, Management and Automation Engineering at the Polytechnic University of Marche, passed away.

Approximately seven months earlier, he had been diagnosed with a late-stage aggressive cancer. In this brief period of illness, he never stopped his activities, showing devotion to his work and students and a strong will power that will be an example for all of us. He spent his final hours surrounded by family and friends.

Prof. Maurizio Panti was full professor of Information Systems and Data Bases. He promoted the growth of informatics both in academia, contributing to the foundation of the Informatics Institute, which then developed into the present department, and in the regional economic arena, also serving as a member in the Scientific and Technical Board for the Regional Information System.

His loss will be greatly felt by all those who knew and worked with him.

Prof. Hongjun Lu

Prof. Hongjun Lu was a world-renowned researcher who served the database community with dedication and distinction in various capacities over the years. He was a trustee of the VLDB Endowment, a member of the ACM SIGMOD Advisory Board (1998–2002), an associate editor of IEEE Transactions on Knowledge and Data Engineering (TKDE), Chair of the Steering Committee of the International Conference on Web-Age Information Management (WAIM), and Co-chair (1998-2001) and Chair (2001-2003) of the Steering Committee of Pacific-Asia Conference of Knowledge Discovery and Data Mining (PAKDD). In December 2004, the China Computer Federation Database Society honored him with a Contribution Award and, just recently, he was honored with this year's inaugural PAKDD Distinguished Contribution Award.

His passing is a great loss not only to the China Database Society but also to the International Database Society. He has made sustained and outstanding contributions to the international database research community as well as to database research.

Organization

External Reviewers

All Editorial Board members contributed reviews for the selection of the submissions. In addition, a number of colleagues helped us in this reviewing task. We would like to express here our gratitude for their cooperation and our warmest thanks for the job they did.

Bettina Berendt
Sonia Bergamaschi
Claudio Bettini
Mokrane Bouzeghoub
Anita Coleman
Jos De Bruin
Flavio De Paoli
Giovanni Denaro
Ying Ding
Gillian Dobbie
Anastasios D. Doulamis
Oleksandr Drutskyy
Birte Glimm
Sven Hartmann
Georges Hébrail
Martin Hepp
Jane Hunter
Mustafa Jarrar
Roland Kaschek
Evgeny Zolin

Yehuda Koren
Mong Li Lee
Lei Li
Xiaonan Lu
R. Manmatha
Andrea Maurino
Rym Mili
Moira Norrie
Axel Polleres
Helen Purchase
Brian Roark
Ingo Schmitt
Richard Smiraglia
Peter Spyns
Paolo Terenziani
Christelle Vangenot
Michalis Vazirgiannis
James Z. Wang
X. Sean Wang

JoDS Editorial Board

Table of Contents

Generic Relationships in Information Modeling

Mohamed Dahchour[1], Alain Pirotte[2], and Esteban Zimányi[3]

[1] Institut National des Postes et Télécommunications,
Av. Allal Al Fassi, Rabat, Morocco
`dahchour@inpt.ac.ma`

[2] Université catholique de Louvain, IAG School of Management,
1 Place des Doyens, B-1348 Louvain-la-Neuve, Belgium
`pirotte@info.ucl.ac.be`

[3] Université Libre de Bruxelles, Department of Computer and Network Engineering,
CP 165/15, 50 Av. F. Roosevelt, B-1050 Brussels, Belgium
`ezimanyi@ulb.ac.be`

Abstract. Generic relationships are abstraction patterns used for structuring information across application domains. They play a central role in information modeling. However, the state of the art of handling generic relationships leaves open a number of problems, like differences in the definition of some generic relationships in various data models and differences in the importance given to some generic relationships, considered as first-class constructs in some models and as special cases of other relationships in other models. To address those problems, we define a list of dimensions to characterize the semantics of generic relationships in a clear and systematic way. The list aims to offer a uniform and comprehensive analysis grid for generic relationships, drawn from a careful analysis of commonalities and differences among the generic relationships discussed in the literature. The usefulness of those dimensions is illustrated by reviewing significant generic relationships, namely, materialization, role, aggregation, grouping, and ownership. Based on those dimensions, a new metamodel for relationships is proposed.

1 Introduction

Information modeling is the activity of creating abstract representations of some aspects of physical and social systems and their environment. Information models are typically built in the early stages of system development, preceding design and implementation. But information models can also be useful even if no system is contemplated: they then serve to clarify ideas about structure and functions in a perception of the world.

Advances in information modeling involve narrowing the gap between concepts in the real world and their representation in information models by identifying powerful abstractions allowing to better represent application semantics (see, e.g., [1,29,31,34,38,46]).

Generic relationships are such powerful abstraction mechanisms. They are high-level templates for relating real-world entities. Well-known generic relationships include the following.

– *Classification* relates a class with a set of objects sharing the same properties. An object must be an instance of at least one class. It is also known as *is-of.* For example John is an instance of class Person.
– *Association* represents a structural connection among classes. Associations can be *binary* or *n-ary* ($n \geq 3$). An example of a binary association is teaches(Professor,Course). An example of a ternary association is prescription(Doctor,Medicine, Patient).
– *Generalization* relates *superclasses* to their specializations called *subclasses.* Subclasses inherit all properties (attributes, methods, roles, integrity constraints) from their superclasses. Subclasses may define new specific properties. For example, Vehicle is a generalization of Car.
– *Aggregation* associates an *aggregate* (or whole or composite) to its *components* (or parts). It is also known as *part-whole* or *part-of.* For example, Car is an aggregation of Body, Engine, and Wheel.

Generic relationships model patterns abstracting collections of related *specific* relationships. Specific relationships are instances of generic relationships in a particular application. For example, Vehicle◁—Car is a specific generalization with pattern SuperClass◁—SubClass (see Figure 1).

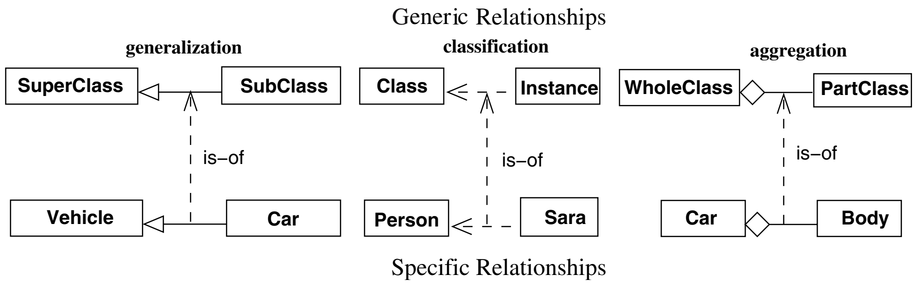


Fig. 1. Generic and specific relationships

Research on information modeling has studied other generic relationships like *materialization* [9], *ownership* [48], *role* [45,10], *grouping* [33], *viewpoint* [32], *generation* [14], *versioning* [3,21], *realization* [25], *transition* [15], and *refinement* [41]. These generic relationships naturally model phenomena typical of complex application domains whose semantics escapes direct representation with classical generic relationships (i.e., association, generalization, classification, and aggregation).

Generic relationships play a central role in information modeling. However, the state of the art of handling them leaves open a number of problems. The semantics of some generic relationships differs among models. Other generic relationships have often been badly understood, underestimated, or merely ignored in some models. In addition, some generic relationships, considered as first-class constructs in some models, are considered as special cases of other relationships in other models. For instance, in UML [41], the aggregation relationship is considered a special kind of the ordinary association, whereas in most models (e.g., [46,36,19,30,23]) aggregation enjoys a status of its own and comes with more specific features than those defined in UML. Another example concerns the *grouping* relationship relating a collection (e.g., TennisClub) and its members (e.g., TennisPlayer). Grouping was defined in [33] as an independent generic relationship with specific characteristics, while it is just considered as a special case of aggregation, e.g., in [46,19].

Another problem concerns the adequacy of choosing some generic relationships rather than others when modeling an application domain. For example, it can be argued that the relationship between students and employees on the one hand and persons on the other hand is more adequately modeled as a role relationship than as a generalization. Generalization seems more adequate to represent the relationship between males and females on the one hand and persons on the other hand. The idea is that the role relationship captures the temporal and evolutionary aspects of real-world objects (e.g., persons may be students and later become employees), while the usual generalization relationship deals with their more static aspects (e.g., most persons are permanently males or females).

Such difficult questions of adequacy or validity of generic relationships for modeling real-world situations are not directly discussed in this paper, although they are illustrated through a number of examples. Instead, the paper precisely characterizes the structural semantics of generic relationships, to help conceptual database designers precisely evaluate the adequacy of choosing one model rather than another.

We argue that some problems with generic relationships mainly concern the absence of formalizable dimensions or criteria along which the relationships can be characterized in a systematic way. The paper defines such dimensions and illustrates their effectiveness by reviewing some generic relationships. Implementation issues are not presented in this paper. They are discussed in detail in [8].

Notations and Conventions. Table 1 gathers the main notations used in the paper. We use UML [41] notations to specify classes, instances, generalization, instantiation, and aggregation. We add notations to represent concepts that have no equivalent in UML. We prefer to draw associations as boxes with rounded corners rather than using the UML notation. As in UML, instances of relationships are called links.

The rest of the paper is organized as follows. Section 2 presents a preliminary classification of generic relationships. Section 3 presents an overview of our relationship model. Section 4 presents the characteristics of a basic binary relationship (denoted $\mathcal{BBR}$). Section 5 defines a set of dimensions that characterize binary generic relationships. Section 6 reviews several generic relationships in the light of those dimensions. Section 7 gives some guidelines to identify and define new generic relationships. Section 8 presents a new metamodel for generic relationships based on their semantics as presented in Sections 4 and 5. Section 9 summarizes and concludes the paper.

Table 1. Notations and their meanings

Notation	Meaning
$I \dashrightarrow C$ or $I \in C$	I is an instance of class C
$C_1 \text{—} \boxed{\text{R}} \text{—} C_2$	R is a binary association between classes C_1 and C_2
$S \text{—}\!\triangleright G$	S is a subclass of superclass G
$W \diamond\!\text{—} P$	whole class W is composed of part class P
$A \text{—}\!\ast C$	abstract class A materializes as concrete class C
$R \rightarrow\!\circ O$	R is a role class of object class O
$O \prec\!\cdots P$	owner class O owns property class P
$M \twoheadrightarrow S$	M is a member class of set S
$\mathcal{BBR}$	the basic binary relationship
$C_1 \overset{R}{\text{———}} C_2$	R is a binary relationship between classes C_1 and C_2
$\rho_C(R)$	the role played by class C in relationship R
$C_1(v_1, v_2) \overset{R}{\text{———}} C_2$	cardinality of role $\rho_{C_1}(R)$ is (v_1, v_2)
$\pi_C(R)$	the set of instances of C participating in R with role $\rho_C(R)$
$R(C_1, C_2)$	R relates C_1 and C_2
$R(c_1, c_2)$	there is an instance of R (a link) relating c_1 and c_2
$R_1 \otimes R_2$	relationships R_1 and R_2 are exclusive
$\rho_C(R_1) \otimes \rho_C(R_2)$	roles $\rho_C(R_1)$ and $\rho_C(R_2)$ are exclusive
$R_1 \subseteq R_2$	relationship R_1 is inclusive in relationship R_2
$\rho_C(R_1) \subseteq \rho_C(R_2)$	role $\rho_C(R_1)$ is inclusive in role $\rho_C(R_2)$
$C_0 \overset{R\,\mid\,d}{\text{———}} \{C_1, \ldots, C_n\}$	partition of R in classes C_i according to discriminator d

2 Classification of Generic Relationships

Generic relationships can be classified along the following three orthogonal dimensions, as depicted in Figure 2: (1) degree, (2) structurality and dynamicity, and (3) dependency on application domains.

Degree. It is the number of participating classes in a relationship. A relationship of degree two is said to be *binary*, and one of degree n ($n \geq 3$) is *n-ary*. Examples of binary generic relationships include:

– *classification* (of pattern Class$\dashleftarrow$Instance) relates an instance (e.g., Sarah) to its class (e.g., person);
– *generalization* (of pattern SuperClass$\triangleleft\!\text{—}$SubClass) relates a superclass (e.g., persons) to its subclasses (e.g., males and females).
– *materialization* (of pattern Abstract$\text{—}\!\ast$Concrete) [39,9] relates a class of categories (e.g., models of cars) with a class of more concrete objects (e.g., individual cars);
– *ownership* (of pattern Property$\cdots\!\succ$Owner) [48] relates an owner class (e.g., persons) and a property (e.g., cars) possessed by their objects;
– *aggregation* (of pattern WholeClass$\diamond\!\text{—}$PartClass) [30,46] relates composites (e.g., cars) to their components (e.g., body and engine);

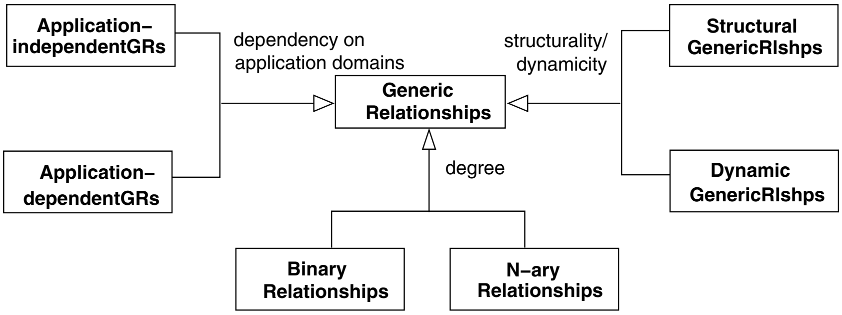


Fig. 2. Classification of generic relationships

- *role* (of pattern ObjectClass∘←RoleClass) [10] relates an object class (e.g., persons) and a role class (e.g., employees), describing dynamic states for the object class;
- *grouping* (of pattern MemberClass→»SetClass) [33] relates a member class (e.g., players) and a grouping class (e.g., teams);
- *viewpoint* (of pattern Class——⊙View) [32] represents partial information about a class viewed from a particular standpoint.

Although most generic relationships are binary, there are situations that are best modeled with n-ary generic relationships. Examples of n-ary generic relationships include the following:

- *view*(Actor,Concept,Facet) [26] relating an actor who sees a concept in a particular facet. Unlike the *viewpoint* relationship above which is closer to the view concept in databases, relationship *view* was defined in the context of requirements engineering;
- *dependency*(Depender,Dependee,Dependum) [49] relating an agent (depender) depending on the other (dependee) for something (dependum) in order that the former may attain some goals.

Structural and dynamic generic relationships. Structural generic relationships are relationships whose semantics can be expressed in terms of static constraints and rules. They are also referred to as *organizational* relationships. Examples of structural generic relationships include the usual classification (where an instance cannot change its class), generalization, aggregation, materialization, grouping, and ownership.

By contrast, dynamic relationships involve some dynamic behavior. Examples include: *dynamic classification* [11], where an instance can dynamically change its class, *generation* [14] where an instance (or a set of instances) of the input class produces an instance (or a set of instances) of the output class; *versioning* [3], which relates an object class and its time-varying versions, modeling various states of the object class; *realization* [25], a variant of classification that allows an object to add structure to that defined by its class; *transition* [15], which keeps track of migrations of instances from source classes to target classes; refinement [41], which specifies that a class is at a finer degree of abstraction than another class; and *role* [45,10], relating an object class with a role class describing its dynamic states.

Dependency on application domains. Generic relationships can be *application independent* or *application specific*. Examples of generic relationships with application-independent semantics include generalization, aggregation, classification, materialization, grouping, role, versioning, and view. An example of application-specific generic relationships is *BinaryDirectedLink* that relates two nodes (or a node and a link) of an hypermedia graph of a document [44]. Another example deals with architectural design [38] where, for example, a relationship *hasGeometry* relates a symbolic object and a geometric object; *adjacentTo* relates two objects unrelated by aggregation and whose distance is less than a specific threshold; *connectedTo* is similar to *adjacentTo*, but the related objects have overlapping volumes. In business and organizational applications, a network of dependency relationships (e.g., *goal dependency*, *task dependency*, *resource dependency*, and *softgoal dependency*) among actors was proposed in [49]. In the domain of resource management, relationship *production*(Resource,Producer,Consumer) can be defined among resources, their producers, and their consumers.

This paper focuses on structural generic relationships with application-independent semantics. The term *generic relationships* will henceforth denote those structural generic relationships.

3 Our Relationship Metamodel: Overview

Our relationship metamodel, depicted in a simplified way in Figure 3, encompasses three kinds of generic relationships: (i) the basic binary relationship $\mathcal{BBR}$, (ii) binary generic relationships like generalization, aggregation, materialization, and association, and (iii) n-ary generic relationships. This section gives an overview of each of those relationships. Their characteristics are presented in detail in Sections 4 and 5. A more elaborate version of the metamodel in Figure 3 is given in Section 8.

The basic binary relationship $\mathcal{BBR}$. It is a generic relationship (of pattern MetaClass——MetaClass) between two metaclasses denoted by MetaClass. MetaClass is the generic metaclass for all application classes. $\mathcal{BBR}$ defines a basic semantics for relationships comprising cardinalities, existence dependency, symmetry, instance transitivity, exclusiveness, inclusion, and attribute propagation. $\mathcal{BBR}$ can be directly instantiated as specific relationships between application classes, without going through generic relationships like generalization, aggregation, or materialization. These specific relationships, like Employee—works—Company, are called *associations* in UML. We will henceforth use that terminology for those relationships.

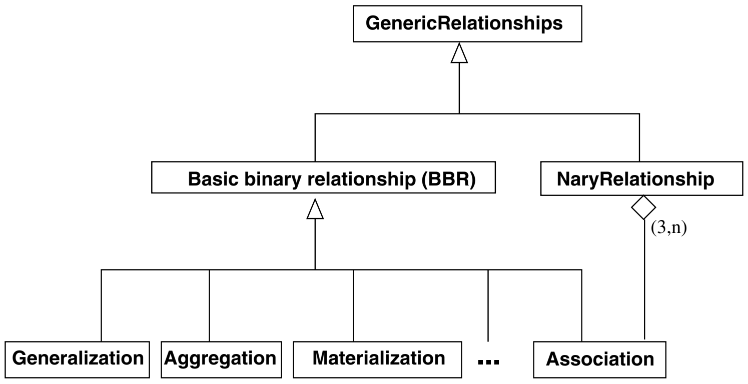


Fig. 3. A simplified view of our metamodel for relationships

Binary generic relationships. Generic relationships like generalization, aggregation, and materialization need the basic semantics of $\mathcal{BBR}$ in addition to their specific semantics. It is thus suitable that generic relationships inherit the semantics of $\mathcal{BBR}$ by subclassing. Hence, the role of $\mathcal{BBR}$ is twofold: directly represent associations and factor out the common semantics of all binary generic relationships.

Generic relationships may refine or redefine the inherited semantics to fit their specific semantics. For instance, cardinality can be inherited from $\mathcal{BBR}$, and refined by generalization as follows: SubClass (1,1)—▷(0,1) SuperClass. Thus, in all specific generalizations (e.g., Car—▷Vehicle), an instance of the subclass corresponds to exactly one instance of the superclass and an instance of the superclass corresponds to at most one instance of the subclass.

N-ary generic relationships. An n-ary relationship relates n classes, for $n \geq 3$. N-ary relationships are more complex than binary ones, they have been much less studied, and have often been poorly understood in practice. Several data models (e.g., OML [12], ORM [18]) do not directly support n-ary relationships. Other models advise against the usage of n-ary relationships, like in the reference manual of UML 2.0 [41], page 472: "In general it is best to avoid n-ary associations, because binary associations are simpler to implement and they permit navigation". It is often argued that n-ary relationships are not frequent in real-world applications, but this point of view is rather a testimony of the typical sophistication of the models produced by current practice in conceptual database design. In practice, n-ary relationships are often represented, and often approximately, by some of their binary projections to be supplemented by consistency constraints. The only correct general way to do without an n-ary relationship is to reify it as a new class with n binary relationships to the given n classes. Our relationship metamodel follows this approach.

We define the semantics of n-ary relationships along a subset of the basic semantics of $\mathcal{BBR}$ (see Section 4). Namely, this semantics includes cardinality, exclusiveness, inclusion, existence dependency, and attribute propagation. The semantics of an n-ary relationship can be defined in terms of the n binary relationships representing it coupled with some dependency constraints. In Figure 3, n-ary relationships are represented by the metatype NaryRelationship that is defined as an aggregate of three or more binary associations represented by metatype Association. Details concerning the decomposition of n-ary relationships into binary equivalent structures can be found in, e.g., [28,43,20].

4 The Basic Binary Relationship

We define the semantics of the basic binary relationship $\mathcal{BBR}$ along several dimensions, including cardinality, existence dependency, symmetry, instance transitivity, exclusiveness, inclusion, and attribute propagation mechanisms. These characteristics are defined in detail in Sections 4.1 to 4.7. They are independent of one another except for some consistency constraints to be defined for combining exclusiveness and inclusion. The completeness of our set of dimensions, although intuitively very important, cannot be proved. It can only be addressed in a pragmatic and empirical manner. The list of characteristics presented in this paper aims to offer a uniform and comprehensive analysis grid for generic relationships; it was drawn after carefully analyzing commonalities and differences among the generic relationships discussed in the literature.

4.1 Cardinality

The cardinality dimension constrains the number of relationship links in which an object can participate. Let R be a relationship associating classes C_1 and C_2. A cardinality (min, max) at the side of C_1 means that each instance of C_1 must participate in at least min and at most max links of R at all times. The most frequent cardinalities are: (0,1) (i.e., at most one), (1,1) (i.e., exactly one), (0,n) (i.e., any number, the unconstrained case), and (1,n) (i.e., at least one). We find this traditional definition more intuitive and expressive than that of UML.

4.2 Existence Dependency

Existence dependency characterizes whether or not an object can exist independently of related objects. There are two typical cases:

– *dependency*, meaning that the existence of an object of C_1 depends on the existence of related objects of C_2. This is known as *mandatory* participation in ER modeling. It is expressed by a minimum cardinality of 1 at the side of C_1;
– *independency*, meaning that the existence of an object of C_1 is independent of the existence of related objects of C_2. This is known as *optional* participation in ER modeling. It is expressed by a minimum cardinality of 0 at the side of C_1.

Existence dependency also specifies how insertion or deletion of one object may influence the existence of connected objects. Let o_1 and o_2 be two objects related by link r. With respect to the deletion operations, there are three main options to maintain the existence dependency:

- *default deletion:* the deletion of an object implies the deletion of all its links (e.g., the deletion of o_1 implies the deletion of r);
- *cascade deletion:* the deletion of an object implies the deletion of all its links as well as all other objects involved in those links (e.g., the deletion of o_1 implies the deletion of r and o_2);
- *restrict deletion:* the deletion of an object is prohibited if the object is involved in at least one link (e.g., the deletion of o_1 is disallowed while link r exists).

Those deletion options can be associated to each role of a given relationship. For example, in Figure 4, *cascade* deletion associated with role employs means that the deletion of a department implies the deletion of all its employees. The *default* deletion associated with role worksOn states that the deletion of an employee only implies the deletion of its link to the related department.

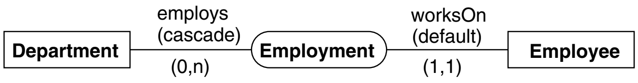


Fig. 4. The cascade and default deletion options

Deletion options must be carefully chosen to avoid inconsistencies. For the example in Figure 4, assume now that the deletion option on role worksOn is *restrict* instead of *default*. When a department is deleted, option *cascade* on role employs requires the deletion of all related employees, while option *restrict* on role worksOn states that those employees cannot be deleted. Thus a contradiction arises.

Existence dependency is sometimes referred to as *referential integrity*. In object models, this means that for any object o_1 containing a reference to an object o_2, the referred object o_2 must indeed exist. In systems where referential integrity is not automatically ensured, the problem of *dangling pointers* may arise if a referred object is deleted.

In the relational model, referential integrity is an inclusion constraint between a set of attributes (called *foreign key*) of a *child* relation and the attributes forming the primary key of a *parent* relation. For the child relation, this concerns insert and update operations. Various repair actions can be specified to avoid violations resulting from deletions and updates in the parent relation:

- *cascade:* in case of update, the new values in the key are propagated to the referencing children, whereas in case of deletion the referencing children are also deleted;

– *set null:* the foreign key attributes in the referencing tuples of the child relation are set to null;
– *set default:* the foreign key attributes in the referencing tuples of the child relation are set to a given default value;
– *no action:* referential integrity remains violated and, if no other operation is executed to correct the mismatch of the corresponding tuples, the complete transaction is rolled back.

Some relational database systems introduce another referential action called *restrict*. Its semantics forbids any change (update or delete) to the primary key of a parent tuple as long as there are referencing child tuples.

4.3 Symmetry

A binary relationship R associating classes C_1 and C_2 is *symmetric* iff $\forall c_1 \in C_1 \, \forall c_2 \in C_2 \; (R(c_1, c_2) \Leftrightarrow R(c_2, c_1))$. Relationships that are not symmetric are called *asymmetric*. Most binary relationships are asymmetric.

The properties of symmetry and asymmetry are particularly relevant for recursive relationships (i.e., where C_1 and C_2 are the same class). Examples of symmetric recursive relationships include siblingOf(Person,Person) and jointlyTaxedWith(Person,Person). Symmetric recursive relationships are also said to be *reflexive*. Examples of asymmetric recursive relationships include supervises(Employee,Employee), assembly(Part,Part), and ancestorOf(Person,Person). Asymmetric recursive relationships are also said to be *irreflexive*.

4.4 Instance Transitivity

Let R be a binary recursive relationship whose instances relate two instances of class C. R is *instance transitive* iff $\forall c_1, c_2, c_3 \in C \; (R(c_1, c_2) \wedge R(c_2, c_3) \Rightarrow R(c_1, c_3))$. For instance, relationship ancestorOf(Person,Person) is instance transitive. Instance transitivity is different from class transitivity, which is presented in Section 5.3.

4.5 Exclusiveness

The *exclusiveness* dimension for binary relationships can be defined for both relationships and roles [18]. As for notation, exclusiveness is represented by a dashed line labeled with the symbol $\otimes$.

Relationship exclusiveness. Exclusiveness between two relationships R_1 and R_2 is defined for relationships relating the same classes C_1 and C_2. It means that the set of links of both relationships are disjoint.

For example, Figure 5 shows two exclusive relationships borrows and reserves: a student cannot simultaneously borrow and reserve the same book (i.e., borrows $\cap$ reserves $= \emptyset$).

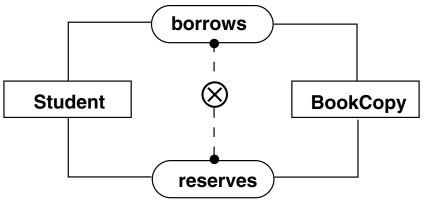


Fig. 5. Exclusiveness between relationships borrows and reserves

Role exclusiveness. Let R_1 and R_2 be two relationships sharing the same class C_0 at one end. Let $\rho_{C_0}(R_i)$ denote the role played by C_0 in R_i and $\pi_{C_0}(R_i)$ denote the set of instances of C_0 participating in R_i. Exclusiveness between roles $\rho_{C_0}(R_1)$ and $\rho_{C_0}(R_2)$ means that $\pi_{C_0}(R_1)$ and $\pi_{C_0}(R_2)$ are disjoint. Formally, $\rho_{C_0}(R_1) \otimes \rho_{C_0}(R_2) \Leftrightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset$.

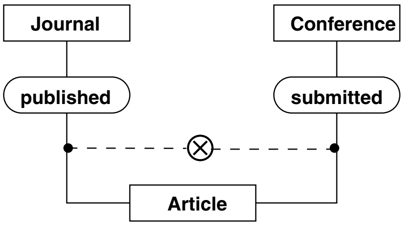


Fig. 6. Exclusiveness between roles $\rho_{\mathsf{Article}}(\mathsf{published})$ and $\rho_{\mathsf{Article}}(\mathsf{submitted})$

For example, the exclusiveness between roles in Figure 6 means that an article cannot simultaneously be submitted to a conference and be published in a journal. In this example, the end classes Journal and Conference of relationships published and submitted, respectively, are distinct. An example of exclusiveness between roles involved in relationships sharing the same end classes is shown in Figure 7: the exclusiveness between roles $\rho_{\mathsf{Apartment}}(\mathsf{rent})$ and $\rho_{\mathsf{Apartment}}(\mathsf{sell})$ means that an apartment cannot simultaneously be rented and sold to clients.

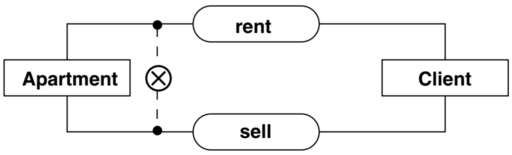


Fig. 7. Exclusiveness between roles $\rho_{\mathsf{Apartment}}(\mathsf{rent})$ and $\rho_{\mathsf{Apartment}}(\mathsf{sell})$

4.6 Inclusion

Like exclusiveness, the *inclusion* dimension for binary relationships can be defined for both relationships and roles. Graphically, inclusion of relationship R_1 in R_2 is represented by a dashed arrow labeled $\subseteq$ with its origin in R_1 and its destination in R_2.

Relationship inclusion. Inclusion of relationship R_1 in relationship R_2 is defined for relationships sharing the same classes C_1 and C_2 at their ends. It means that the set of links of R_1 is a subset of the set of links of R_2.

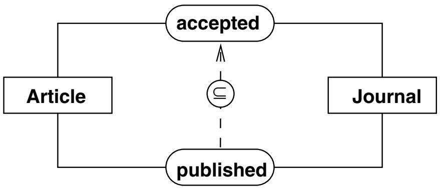


Fig. 8. Inclusion of relationship published in relationship accepted

For example, Figure 8 shows inclusion of relationship published in relationship accepted. This means that an article published in a journal has necessarily been accepted for publication in that journal (i.e., published $\subseteq$ accepted).

Role inclusion. Inclusion of role $\rho_{C_0}(R_1)$ in role $\rho_{C_0}(R_2)$ is defined for relationships sharing the same class C_0 at one end. It means that the set of instances of C_0 participating in role R_1 is a subset of the set of instances of C_0 participating in role R_2. Formally, $\rho_{C_0}(R_1) \subseteq \rho_{C_0}(R_2) \Rightarrow \pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2)$.

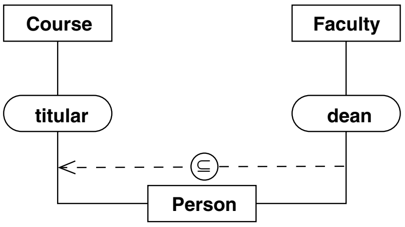


Fig. 9. Inclusion of role ρ_{Person}(dean) in role ρ_{Person}(titular)

For example, Figure 9 shows inclusion of role ρ_{Person}(dean) in role ρ_{Person}(titular). This means that a faculty dean must be titular of a course. In this example, the end classes Faculty and Course of relationships dean and titular, respectively, are distinct. An example of inclusion between roles participating in relationships sharing the same end classes is shown in Figure 10.

Role $\rho_{\mathsf{Student}}(\mathsf{practices})$ is included in role $\rho_{\mathsf{Student}}(\mathsf{registers})$, meaning that a person practicing a sport has necessarily registered in a sport. That allows John to practice Tennis while being registered in Football. If it is required that each person who practices a sport should necessarily register for that sport, then the constraint should be an inclusion of relationship practices in relationship registers.

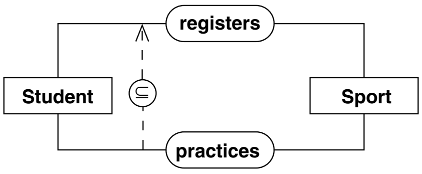


Fig. 10. Inclusion of role $\rho_{\mathsf{Student}}(\mathsf{practices})$ in role $\rho_{\mathsf{Student}}(\mathsf{registers})$

Consistency rules. The following additional rules hold for exclusiveness and inclusion constraints on relationships R_1 and R_2:

- relationship inclusion and relationship exclusiveness cannot coexist, because $R_1 \subseteq R_2 \Rightarrow R_1 \cap R_2 \neq \emptyset$ and conversely $R_1 \cap R_2 = \emptyset \Rightarrow (R_1 \not\subseteq R_2) \wedge (R_2 \not\subseteq R_1)$;
- relationship inclusion and role exclusiveness cannot coexist, because $R_1 \subseteq R_2 \Rightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) \neq \emptyset$ and conversely $\pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset \Rightarrow (R_1 \not\subseteq R_2) \wedge (R_2 \not\subseteq R_1)$;
- role inclusion and role exclusiveness cannot coexist, because $\pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2) \Rightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) \neq \emptyset$ and conversely $\pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset \Rightarrow (\pi_{C_0}(R_1) \not\subseteq \pi_{C_0}(R_2)) \wedge (\pi_{C_0}(R_2) \not\subseteq \pi_{C_0}(R_1))$.
- however, role inclusion and relationship exclusiveness may coexist, because $\pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2) \not\Rightarrow R_1 \cap R_2 \neq \emptyset$ and conversely $R_1 \cap R_2 = \emptyset \not\Rightarrow \pi_{C_0}(R_1) \not\subseteq \pi_{C_0}(R_2)$.

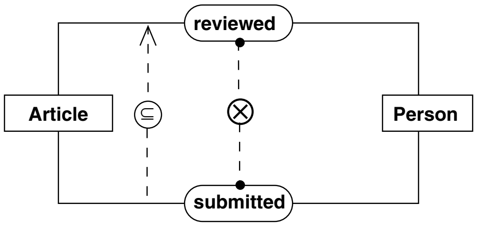


Fig. 11. Coexistence between role inclusion and relationship exclusiveness

For example, in Figure 11 role $\rho_{\mathsf{Article}}(\mathsf{submitted})$ is included in role $\rho_{\mathsf{Article}}(\mathsf{reviewed})$ (a reviewed paper has necessarily been submitted), while relationships submitted and reviewed are mutually exclusive (an article cannot be submitted and reviewed by the same person).

4.7 Propagation Mechanisms

Given a binary relationship R associating classes C_1 and C_2, attributes can propagate from C_1 to C_2 and vice versa. We define two kinds of propagated attributes: *derived* attributes and *computed* attributes.

An attribute of class C_1 is said to be derived from C_2 through R if it is inherited as is from C_2 to C_1. An attribute a_1 of class C_1 is said to be computed from attributes a_{2_1}, ..., a_{2_p} ($p \geq 1$) of C_2 through R if the value of a_1 for a given object c_1 of C_1 is computed from values of attributes a_{2_1}, ..., a_{2_p} of objects belonging to C_2 that are related to c_1 via R. This means that value(a_1)= f(value(a_{2_1}), ..., value(a_{2_p})). Function f is an aggregate operator such as +, -, *, min, max, avg.

For example, attribute publicationDate of class Article can be derived from attribute issueDate of class Journal via relationship published, since its value for each article is the same as its value for the issue of the journal in which the article is published. Similarly, the number of pages of a journal can be computed as the sum of the number of pages of its articles. Also, the average grade of a class can be computed as the average of individual grades of all students of that class.

Propagation mechanisms are expressed in ODMG [5] by means of *path expressions*. For example, the publication date of an article a1 can be expressed by the path expression a1.j1.issueDate that returns the publication date of journal j1 where a1 appeared. Such propagations are not specific to relationships. Derived and computed attributes can also be defined within a single class. An usual example is attribute age of class Person that can be derived from attribute birthDate of the same class.

5 Binary Generic Relationships

This section describes the characteristics of binary generic relationships in a systematic way. Part of these characteristics are inherited from the basic relationship $\mathcal{BBR}$ by subclassing. As mentioned earlier, some of these properties, like cardinality, can be redefined by some generic relationships to conform with their particular semantics. Another part of the characteristics of generic relationships is described along several dimensions like class- and instance-level semantics, composition, class transitivity, class nonrecursivity, multiplicity, and partitioning. These dimensions are described in detail in Sections 5.1 to 5.9.

Although these dimensions have been identified by carefully studying a substantial collection of generic relationships, we cannot claim that they are exhaustive. The list remains open to other dimensions that could be identified by exploring other generic relationships.

The following notations and assumptions are needed to formally define some dimensions below:

– Let $R(MC_1, MC_2)$ be a predicate stating that R is a binary generic relationship between two metaclasses MC_1 and MC_2. Specific relationships, which

are instances of R, have the same name and are defined between classes denoted by C_1 and C_2. Thus, $R(C_1, C_2)$ means that there is a specific relationship R between classes C_1 and C_2 where C_1 is an instance of MC_1 and C_2 is an instance of MC_2.
- The role played by MC_1 in R, noted $\rho_{MC_1}(R)$, is the same as the role played by C_1 in R, noted $\rho_{C_1}(R)$. For example, the whole role played by WholeClass in WholeClass◇—PartClass is the same as the role played by Car in the specific aggregation Car◇—Body.
- $C \in MC$ means that class C is an instance of metaclass MC.

5.1 Class- and Instance-Level Semantics

The semantics of generic relationships concerns both classes and instances of these classes. Consequently, comprehensive semantics must deal with both the *class level* and the *instance level* in a coordinated manner.

For example, the class-level semantics of generalization states that:

- a class can have several superclasses and several subclasses;
- each class inherits all properties from its superclasses;
- conflicts induced by multiple inheritance are avoided with a specified strategy;
- each class has a (1,1) cardinality regarding each of its superclasses and a (0,1) cardinality regarding each of its subclasses.

At the instance level, the generalization relationship expresses the following semantics:

- an instance of a class C cannot be an instance of another class that is not direct or indirect superclass of C[1];
- an instance cannot have additional properties than those of its class[2].

5.2 Composition

Generic relationships can be involved in compositions, where a class plays several roles of the same generic relationship R in several specific relationships based on R, as schematized in Figure 12(a). Formally, a generic relationship R can be composed iff $\exists C_1, C_2, C_3$ such that $R(C_1, C_2) \wedge R(C_2, C_3)$ where $C_1 \in MC_1, C_2 \in MC_2, C_2 \in MC_1$, and $C_3 \in MC_2$. C_2 plays at the same time role $\rho_{MC_1}(R)$ and role $\rho_{MC_2}(R)$.

An example of composition of generalizations is Person◁—Student◁—GraduateStudent, where Student is at the same time a superclass of GraduateStudent and a subclass of Person (see Figure 12(b)). Similarly, in the composition of aggregations Car◇—Body◇—Door, Body is at the same time a composite of Door and a component of Car.

[1] This is possible in models allowing multiple classification, like Telos [35] or MADS [37], where an object can be an instance of several classes not related, directly or indirectly, by the generalization link.

[2] This restriction is overcome with the realization relationship [25].

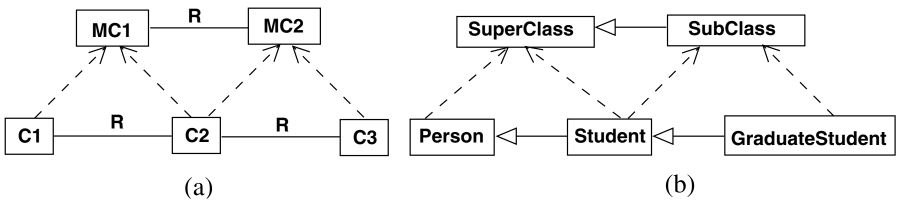


Fig. 12. Composition of relationships

5.3 Class Transitivity

Some generic relationships that can be composed may be class transitive. A generic relationship R is class transitive iff $\forall C_1, C_2, C_3 \; R(C_1, C_2) \wedge R(C_2, C_3) \Rightarrow R(C_1, C_3)$. For example, generalization is class transitive: the generalizations Person◁—Student◁—GraduateStudent imply Person◁— GraduateStudent. By contrast, aggregation is not class transitive in general.

5.4 Class Nonrecursivity

A generic relationship R is said to be *class nonrecursive* if R does not hold at the class level between two occurrences of the same class. For example, generalization is class nonrecursive, since a class cannot be a subclass of itself, while aggregation may be class recursive (e.g., Program◇—Program).

5.5 Multiplicity

A role in a generic relationship is said to be *multiple*[3] if the same class can participate with that role in several instances of the generic relationship (see Figure 13(a)). Formally, role $\rho_{MC_1}(R)$ is multiple iff $\exists C_1, C_2, C_3$ such that $R(C_1, C_2) \wedge R(C_1, C_3)$ where $C_1 \in MC_1, C_2 \in MC_2$, and $C_3 \in MC_2$. Multiplicity of role $\rho_{MC_2}(R)$ can be defined in a similar way. Figure 13(a) shows an example of multiplicity for role $\rho_{MC_1}(R)$.

Most generic relationships allow multiplicity in each role. For example, with generalization, a class can have several superclasses and several subclasses as shown in Figure 13(b). Also, with aggregation, a composite can have several components and a component can be part of several composites.

5.6 Partitioning

Generalization and aggregation were defined so far as binary relationships. However, it is often natural to group several binary relationships that involve the same superclass for generalizations, or the same component or the same composite for aggregations. Such groupings add semantics to the semantics of the

[3] This definition of multiplicity should not be confused with its use, e.g., in UML, as a synonym of *cardinality*.

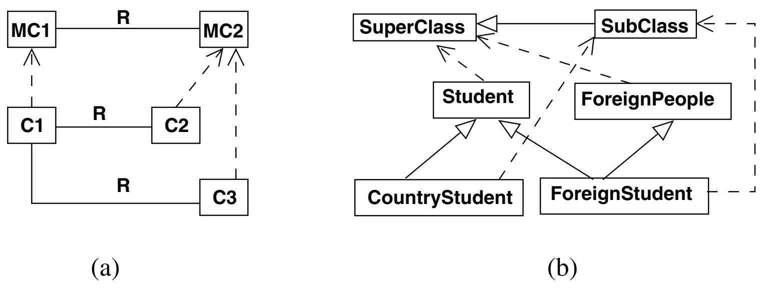


Fig. 13. Multiplicity of relationships

participating binary relationships. As in UML, such groupings are referred to as *partitions*[4]. Each partition, together with the binary generalizations and aggregations taking part in it, carry a unique discriminator (or label). A partition for a generic relationship R is noted $C_0 \overset{R\,|\,d}{\text{———}} \{C_1, \ldots, C_n\}$ where the $C_0 \overset{R}{\text{———}} C_i$ are specific binary relationships that are instances of R, and d is the discriminator of the partition.

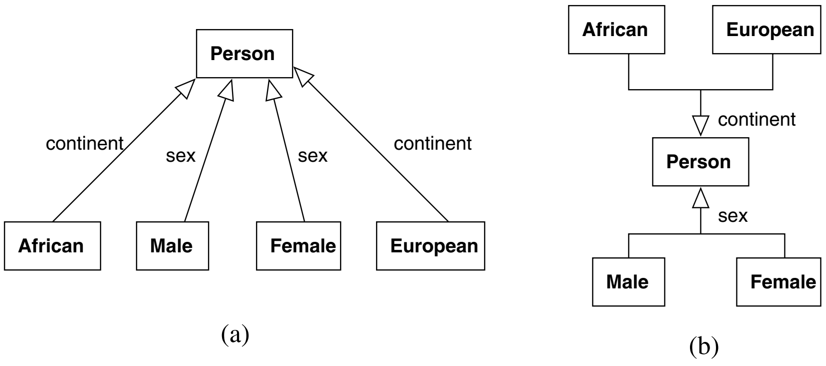


Fig. 14. Partitions for generalizations

Figure 14(b) shows two partitions for generalizations, Person$\overset{\text{sex}}{\triangleleft\!\!—}$$\{$Male, Female$\}$ and Person$\overset{\text{continent}}{\triangleleft\!\!———}$$\{$African, European$\}$, obtained by grouping binary generalizations in Figure 14(a), with discriminators sex and continent, respectively. Similarly, Figure 15(b) shows two partitions obtained by grouping binary aggregations with discriminators space and time.

Partitions of generalizations can be characterized along the usual dimensions of *totality* and *exclusiveness*, resulting in four combinations: (total, exclusive),

[4] The term *partition* has a specific meaning in set theory. In our context, the term *grouping* could be more appropriate, but we avoid it because it denotes another generic relationship.

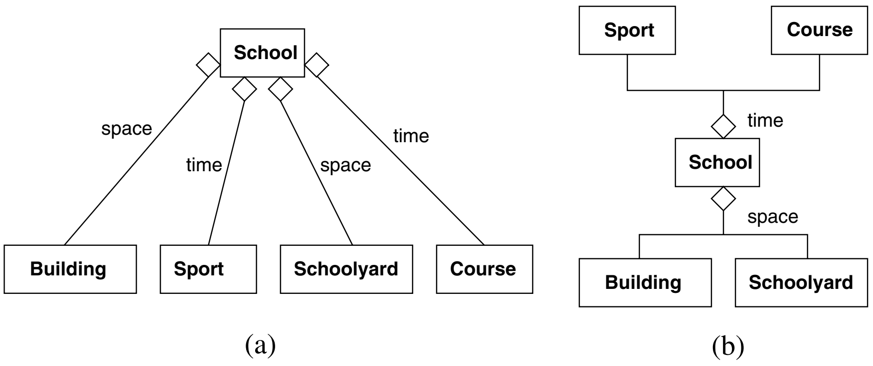


Fig. 15. Partitions for aggregations

(partial, exclusive), (total, overlapping), and (partial, overlapping). More formally, let $\mathcal{P} = C_0 \overset{d}{\lhd\!\!-} \{C_1, \ldots, C_n\}$ be a partition of a superclass C_0 into a set of subclasses $C_1, \ldots, C_n$ according to discriminator d. The dimensions above are defined as follows.

– *Total:* every instance of the superclass is an instance of at least one subclass in $\mathcal{P}$. Formally, $\mathcal{P}$ is total iff $\forall c \in C_0\ \exists i \in [1, n]\ (c \in C_i)$.
– *Partial:* an instance of the superclass needs not be an instance of a subclass within $\mathcal{P}$. Formally, $\mathcal{P}$ is partial if it may be the case that $\exists c \in C_0\ \forall i \in [1, n]\ (c \notin C_i)$.
– *Exclusive:* an instance of the superclass may be an instance of no more than one subclass within $\mathcal{P}$. Formally, $\mathcal{P}$ is exclusive iff $\forall i \in [1, n]\ \forall c \in C_i\ \nexists j \in [1, n]\ (i \neq j \wedge c \in C_j)$.
– *Overlapping:* an instance of one subclass may simultaneously be an instance of another subclass in $\mathcal{P}$. Formally, $\mathcal{P}$ is overlapping if it may be the case that $\exists c \in C_0\ \exists i, j \in [1, n]\ (i \neq j \wedge c \in C_i \wedge c \in C_j)$.

Some authors (e.g., [45]) distinguish *static* from *dynamic* partitions. Roughly, static partitions correspond to total and exclusive partitions, and dynamic partitions correspond to total and overlapping partitions.

5.7 Exclusiveness

This characteristic is inherited from relationship $\mathcal{BBR}$ which defines two categories of exclusiveness: relationship exclusiveness and role exclusiveness. However, generic relationships with the multiplicity property for a role may only involve exclusiveness between roles.

For example, in the aggregations Proceedings◇—Article—◇Journal, Proceedings◇—Article and Journal◇—Article can be exclusive, with the meaning that the same article cannot appear both in conference proceedings and in a journal. Similarly, in the materializations Video∗—Movie—∗ DVD, Movie—∗DVD and Movie—∗Video can be considered as exclusive if a movie can materialize either in a DVD or in a Video but not in both at the same time.

5.8 Inclusion

Like exclusiveness, this characteristic is inherited from relationship $\mathcal{BBR}$. Generic relationships with the multiplicity property for a role may only involve inclusion between roles.

For example, in the generalizations PhDStudent—▷Person◁—Teaching-Assistant, generalization Person◁—TeachingAssistant is inclusive in Person ◁—PhDStudent if a teaching assistant is necessarily a PhD student. Similarly, in the materializations Video∗—Movie—∗DVD, materialization Movie —∗DVD is inclusive in Movie—∗Video if a movie can materialize as a DVD only if it is already materialized as a Video. In the groupings PoliticalParty↞—Deputy—↠Parliament, Deputy—↠Parliament is inclusive in Deputy—↠PoliticalParty if a deputy in a parliament necessarily belongs to a political party.

5.9 Propagation

Most generic relationships allow propagating structure and behavior from one participant to another. This is carried out by *inheritance* or by *delegation*. In some cases, propagation is unidirectional. For example, in generalization, subclasses (totally or partially) inherit attributes and methods from superclasses. In aggregation, composites can access some properties of their parts (and vice versa) by delegation. For example, Car inherits the color attribute of its component Body.

Inheritance by delegation [27] can be characterized as follows. Assume a message m(arguments) is sent to an object o_1 and method m is not defined in o_1's class. According to the usual message-passing semantics, the message handler would signal an error. But, if the object o_1 was related to an object o_2 with respect to some semantic relationship, it could make sense to find out whether o_2 would be able to return a semantically meaningful response by executing method m. Thus, the object o_1 can be seen as delegating the execution of method m to its related object o_2.

6 A Review of Some Generic Relationships

This section reviews some generic relationships and defines their semantics in the light of the dimensions defined in Section 5. Of course, the goal here is not to describe in detail that semantics, but rather to illustrate the effectiveness of our dimensions.

6.1 Materialization

Materialization [9] is a binary relationship with pattern Abstract—∗Concrete relating a class of abstract objects and a class of more concrete objects, where each abstract object can be viewed as a category characterizing a subset of the concrete objects.

In the example of Figure 16, CarModel is the abstract class of materialization CarModel—∗Car and Car is its concrete class. CarModel represents information typically displayed in the catalog of car dealers, while class Car represents information about individual cars. Figure 17 shows an instance FiatRetro of CarModel and an instance Nico's Fiat of Car, of model FiatRetro.

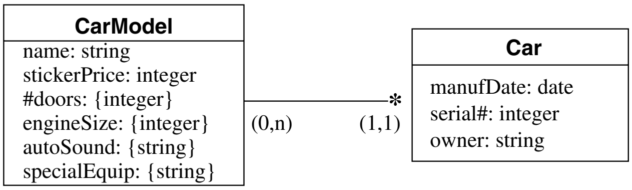


Fig. 16. An example of materialization

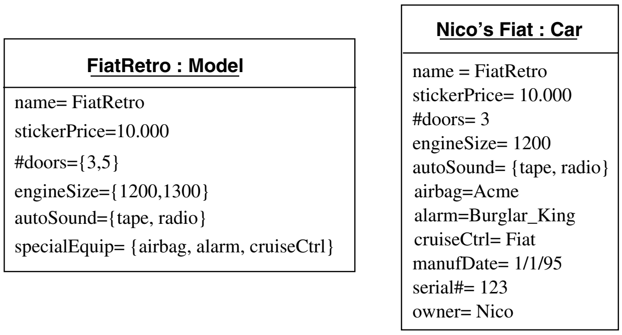


Fig. 17. Instances of CarModel and Car classes from Figure 16

Cardinality. Intuitively, the materialization CarModel—∗Car means that every concrete car (e.g., Nico's Fiat) has exactly one model (e.g., Fiat-Retro), while there can be any number of cars of a given model. Most real-world examples of materialization have cardinality (1,1) at the side of the concrete class and cardinality (0,n) at the side of the abstract class, although the latter cardinality can be further constrained.

Dependency. In a materialization, the deletion of an abstract instance induces the deletion of its associated concrete instances. In the materialization of Figure 16, when model FiatRetro is deleted, all instances of Car associated to that model, e.g., Nico's Fiat, are also deleted. On the other hand, the deletion of a concrete instance induces the deletion of its associated abstract instance only if the minimal cardinality at the abstract side of the materialization is 1. For example, if in CarModel—∗Car the minimal cardinality of CarModel is 1, meaning that a car model has associated at least one car, then the deletion of the last car of a model implies the deletion of that car model.

Attribute propagation. Some information in a concrete instance is naturally viewed as obtained from its associated abstract instance. For example, in Figure 17, Nico's Fiat directly inherits the name and stickerPrice of its model FiatRetro. Further, Nico's Fiat has attributes #doors, engineSize, and autoSound whose values are selections among the options offered by multivalued attributes with the same name in FiatRetro. For example, the value of engineSize for Nico's Fiat is taken from the possible values of the engineSize in FiatRetro. Finally, the value {airbag, alarm, cruiseCtrl} of attribute specialEquip for FiatRetro means that each car of model FiatRetro comes with three pieces of special equipment: an air bag, an alarm system, and a cruise-control system. Thus, Nico's Fiat has three new attributes named airbag, alarm, and cruiseCtrl, whose suppliers are, respectively, Acme, Burglar_King, and Fiat. Other FiatRetro cars may have different suppliers for their special equipment and cars of models other than FiatRetro may have a different set of pieces of special equipment.

These different kinds of attribute propagation from an abstract class to its concrete class are discussed in detail in [9].

Composition. Materializations can be composed in hierarchies, where the concrete class of one materialization is also the abstract class of another materialization. For example, the materializations Play—∗Setting—∗Performance models that theater Plays materialize as Settings that embody the production decisions for a theatrical season. Settings in turn materialize as Performances, at a particular date, with each role of Play assigned to a specific actor for each Performance.

Class transitivity. Materialization is class transitive: for classes A, C, and D, materializations A—∗C—∗D imply A—∗D.

Class nonrecursivity. Materialization is class nonrecursive, in that a class cannot materialize in itself.

Materialization semantics. The semantics of materialization is defined as a combination of generalization, classification, and of a class/metaclass correspondence. This is expressed as a collection of *two-faceted* constructs, each one being a composite structure comprising an object, called the *object facet*, and an associated class, called the *class facet*. The object facet is an instance of the abstract class and the class facet is a subclass of the concrete class. Details about this semantics are given in [9].

6.2 Aggregation

Aggregation (e.g., [16,22,24,30,46]) is a binary relationship with pattern WholeClass◇—PartClass by which a set of (component) objects is considered a higher-level (aggregate) object. For example, Figure 18 shows two aggregations between composite Newspaper and components Editorial and Article.

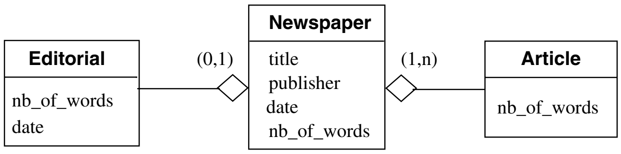


Fig. 18. Examples of aggregation

Cardinality. For the composite role, cardinality determines how many components can be grouped together to form a composite. For example, in Figure 18, a newspaper is composed of (1,n) articles. For the component role it specifies the number of composites that a component can be part of.

The lifetime of parts sometimes depends on that of their wholes and conversely. The *part-to-whole* dependency means that the existence of a part depends on the existence of the corresponding whole, i.e., that the deletion of the whole implies the deletion of the part. As an example, Journal◇—Article is part-to-whole dependent if the deletion of a journal implies the deletion of its articles. The *whole-to-part* dependency means that the existence of a whole depends on the existence of the corresponding part, i.e., the deletion of the part implies the deletion of the whole.

Attribute propagation. Some features of a whole are viewed as features of its parts and vice versa. Thus, there are two kinds of attribute propagation: *upward* propagation from the part class to the whole class and *downward* propagation from the whole class to the part class. For example, in Car◇—Body, the color of a car can be propagated upwards from its corresponding body. Similarly, in Newspaper◇—Article, the date of articles is propagated downwards from their corresponding newspaper. Furthermore, the value of a propagated attribute can be obtained as a combination of values from several source objects. For instance, the color of a car's body can be defined as some combination of the colors of its panels.

Composition. Aggregations can be composed in hierarchies, where the component class of one aggregation is also the composite class of another aggregation as in Building◇—Room◇—Wall.

Aggregation allows multiplicity for both the composite and the component roles. Figure 18 shows an example of multiplicity for the composite role. An example of multiplicity for the component role is Journal◇—Article —◇Compilation where an article can be included in a journal or in a compilation.

Class transitivity. Aggregation is not class transitive in general. For example, the aggregations Hand—◇Musician—◇Orchestra does not imply Hand—◇Orchestra. However, there are some categories of aggregations that are class transitive when taken in combination. For example, the taxonomy proposed in [46] includes seven subcategories of aggregation: (1) component—◇object (Engine—◇Car); (2) feature—◇event (Panel—◇Confe-rence); (3) member—◇collection (Advisor—◇ThesisCommittee); (4) portion —◇mass (Section—◇Chapter); (5) phase—◇activity (Analysis—◇SystemDevelopment); (6) place—◇area (City—◇Country); (7) stuff—◇object (Metal —◇Vehicle). While in [46] only aggregations belonging to the same subcategory are class transitive, class-transitive combinations of aggregations can be defined among categories (1), (4), (5), and (6) [30].

Class nonrecursivity. Aggregations can be recursive: a typical example is that of part-subpart in assemblies, where parts are composed of other parts.

Exclusiveness. Aggregations can be exclusive and shared. A shared aggregation puts no restrictions on the number of composites that a given component can be part of, allowing the component to be shared. An example is Compilation◇—Article if the same article can be included in any number of compilations.

An exclusive aggregation enforces the restriction that a given component can be part of only a single composite. Exclusiveness is natural in physical assemblies. Thus, in Car◇—Engine, two cars cannot share the same engine. This kind of exclusiveness is called *class exclusiveness* as it enforces the exclusive reference constraint within a single class. It is also the case that a car and, say, an airplane cannot share the same engine. This type of exclusiveness is called *global exclusiveness* since it bears on the entire database.

Partitioning. An aggregate class may have several partitions of component classes, each corresponding to a specific discriminator. For example School can be decomposed in two partitions: School$\overset{\text{space}}{\diamond\!\!—}$$\{$Building,Schoolyard$\}$ and School$\overset{\text{time}}{\diamond\!\!—}$ $\{$Course, Sport$\}$.

6.3 Role

Role [2,40,45,13,6,47,10] is a binary relationship with pattern ObjectClass○←RoleClass relating an object class and a role class describing dynamic states for the object class. This definition of role should not be confused with the concept of roles in the entity-relationship model. Differences between those concepts are discussed in [6].

Figure 19 shows two role relationships relating an object class Person, and role classes Student and Employee. In a role relationship, the object class defines permanent properties of objects while each role class defines a set of properties characterizing a particular aspect in which those objects can be viewed during their lifetime. The idea is that the role relationship captures the temporal and

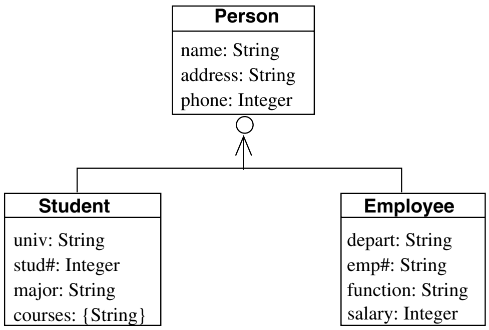


Fig. 19. Examples of roles

evolutionary aspects of real-world objects, while the usual generalization relationship deals with their static aspects. Thus, while classes Male and Female may be linked to Person via generalization links, Student and Employee would rather be linked to Person via role links. Intuitively, the role relationship offers modeling capabilities similar to generalization valid for a limited time.

Cardinality. Each instance of a role class (e.g., Student) is related to exactly one instance of its object class (e.g., Person) but, unlike generalization, each instance of the object class can be related to any number of instances of the role class, depending on the maximal cardinality at the side of the object class. For example, John can be at the same time a student in more than one university and an employee in more than one department.

Dependency. The lifetime of roles depends on that of objects playing those roles. Thus, the deletion of an object induces the deletion of its associated roles. Also, the deletion of a role may induce the deletion of its associated object if the minimal cardinality of the object class is 1. For example, if Person(1,n)○←(1,1)Employee, meaning that a person plays at least once the employee role, then the deletion of the last employee role implies the deletion of that person.

Attribute propagation. Role classes are not introduced for sharing information. This should rather be the responsibility of generalization. If Student is viewed as a subclass of Person, it inherits all properties from Person. Viewed as a role class of Person, Student does not inherit properties of Person. Instead, instances of role classes access properties of their corresponding objects by delegation.

Composition. Role relationships can be composed in hierarchies, where the role class of one role is also the object class of another role. For example, class Person may have role class Employee, and the latter may have two role classes Professor and UnitHead.

Role allows multiplicity for both the object and the role classes. Figure 19 shows an example of multiplicity for the object class. An example of multiplicity for the role class is $\mathsf{Student} \circ\!\!\leftarrow \mathsf{Councilor} \rightarrow\!\!\circ \mathsf{Faculty}$ where both students and faculties can play the role of councilors in the university council.

Class transitivity. Role is class transitive: for role classes R_1, R_2, and object class O, $R_1 \rightarrow\!\!\circ R_2 \rightarrow\!\!\circ O$ implies $R_1 \rightarrow\!\!\circ O$.

Class nonrecursivity. Role is class nonrecursive: for each class C we cannot have $C \circ\!\!\leftarrow C$.

Partitioning. An object class may have several partitions of role classes. For example, in an age perspective, a person may play the role of teenager or an adult, whereas, in an employment perspective, a person may play the role of an employee or an unemployed. Therefore, there are two partitions: $\mathsf{Person} \overset{\mathsf{age}}{\circ\!\!\longleftarrow} \{\mathsf{Teenager}, \mathsf{Adult}\}$ and $\mathsf{Person} \overset{\mathsf{employment}}{\circ\!\!\longleftarrow} \{\mathsf{Employee}, \mathsf{Unemployed}\}$.

6.4 Grouping

Grouping [4,33] is a binary relationship with pattern $\mathsf{MemberClass} \twoheadrightarrow \mathsf{SetClass}$ by which a collection of set members is considered as a higher-level set object. Figure 20 shows an example of grouping between the member class TennisPlayer and the set class TennisClub.

The set class in a grouping has at least one *set-determining* attribute and, optionally, a number of *set-describing* attributes and/or constraints. The set-determining attribute is the attribute whose value is the set of members. In Figure 20, the grouping class TennisClub defines three set-determining attributes grouped under the category ⟨⟨members⟩⟩. A set-describing attribute is an attribute whose value is derived from attributes of the set of members. In Figure 20, under the category ⟨⟨attributes⟩⟩ there are two ordinary attributes name and fee and one set-describing attribute avgAgeOfMembers holding the average age of the club members.

The most important difference between the concepts of grouping and set is that a grouping is concerned with properties and constraints of the grouping viewed as a whole in addition to set membership. Thus, whereas two sets are equal if and only if they have the same members, this is not necessarily so for groupings. Two groupings having the same members, for example, two specific

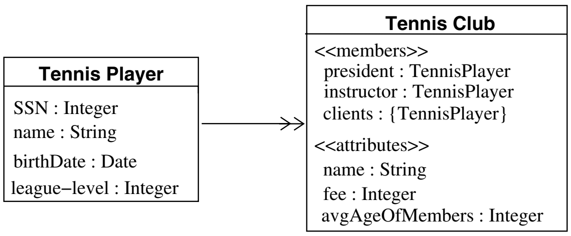


Fig. 20. An example of grouping

clubs, may differ in their internal identifiers or in the values of some property associated with the grouping, such as the minimum age required to be a member of the club.

Cardinality. In general, the grouping relationship constrains a set class to have at least one member whereas the participation of a member in the grouping may be optional or mandatory. For example, the cardinalities in Person(0,1)—↠(1,n)PoliticalParty mean that every person may be a member of at most one political party and that every political party has at least one member.

Dependency. In general, the lifetime of members does not depend on that of their groupings and conversely. However, a dependency may be implied by the cardinality constraints. For example, in Employee(1,1)—↠(1,20)De-partment, due to the cardinality (1,1), the deletion of a department implies the deletion of its employees.

Attribute propagation. Grouping can be seen as a kind of aggregation. Hence, both upwards and downwards propagation are possible, although this is not clearly stated in [4] or [33].

Composition. Grouping can be composed in hierarchies, where the member class of one grouping is also the set class of another grouping. An example is TennisPlayer—↠TennisClub—↠TennisFederation.

Grouping allows multiplicity for both the member and the set classes. Examples are, respectively, TennisClub↞—Employee—↠TradeUnion and Person—↠Sponsors↞—Organization.

Class transitivity. Grouping is normally not class transitive. For example, Book—↠Library —↠LibraryNetwork does not imply Book—↠LibraryNetwork.

Class nonrecursivity. Grouping is class nonrecursive: a class cannot be member of itself.

Exclusiveness. In the grouping relationship, members can be exclusive or shared. A shared member puts no restrictions on the number of groupings that a given element can be member of, allowing the member to be shared. For example TennisClub↞—TennisPlayer is shared if the same player can be a member in any number of clubs.

Member covering. This specifies whether or not all instances of the member class are necessarily related to an instance of the grouping class.

Partial covering means that there is at least one member that does not belong to any grouping. For example, in Employee—↠TennisClub, not all employees must be members of the tennis club. *Complete covering* means that the grouping provides a complete covering of the member class. For example, in TennisPlayer—↠TennisClub the grouping class TennisClub covers all instances of the member class TennisPlayer.

6.5 Ownership

Ownership [48,17] is a binary relationship with pattern Property···≻ Owner relating an owner and a property that is possessed. For example, in Figure 21 Person is the owner class and BankAccount is the property class.

Intuitively, ownership means that the owner of a property has certain rights on the property. Various shades of ownership express the intuitive semantics of the relationship. Thus, the owner can be a person or a legal entity (e.g., a corporation or an organization). Property ownership can be temporary or permanent. A property can be *real* (e.g., a piece of land), *intellectual* (e.g., an idea, a creative work, a patent), or *personal* encompassing everything that is not a real or an intellectual property.

Cardinality. In general, ownership constrains a property to have at least one owner whereas an owner may have 0 or several properties. In the example of Figure 21, a person can have (0,n) bank accounts and a bank account can have (1,n) owners.

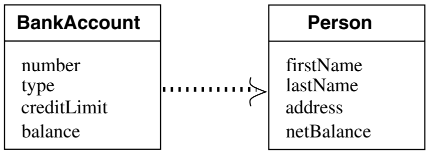


Fig. 21. An example of ownership

Dependency. The deletion of a property can cause deletion of the owner. For example, suppose that an insurance company distinguishes people who own cars from people who do not. This can be modeled by a class Person with a subclass CarOwner and an ownership CarOwner≺···Car. In this case, the car owner is dependent on the car, i.e., if a car owner only owns one car and this car is deleted, then the car owner should be deleted from class CarOwner (but not from class Person).

An example of dependency of the property on the owner is an ownership Employee≺···Car with an additional constraint stating that, when employees stop working for the company, the information about their cars is no longer needed.

Attribute propagation. Some features of a property are naturally viewed as features of its owner or vice versa. For example, the address of persons may be modeled as the address of their house rather than as an attribute of persons. Likewise, the name on a passport can be modeled as the name of the passport owner. In the former case, the value of address is propagated upwards from the property to the owner. In the latter case, the value of name is propagated downwards from the owner to the property. Furthermore, the value of the propagated attribute can be obtained as a combination of values from several objects linked through ownership. For example, in Figure 21, attribute netBalance of a person is computed as the sum of the balance of the person's bank accounts.

Composition. Ownership can be composed where the property of an ownership is also the owner of another ownership, as in Corporation$\prec\!\cdots$Division $\prec\!\cdots$Factory.

Ownership allows multiplicity for both the property and the owner roles. Thus, owner classes can own several properties as in Vehicle$\cdots\!\succ$Person$\prec\!\cdots$House. Also, a property can be owned by several owners as in Person$\prec\!\cdots$Stock$\cdots\!\succ$Company.

Class transitivity. Ownership is generally not class transitive. A counterexample is that Person$\prec\!\cdots$Cat$\prec\!\cdots$Claw does not imply Person$\prec\!\cdots$Claw.

Class nonrecursivity. Ownership can be recursive: for example, a company can own other companies.

Exclusiveness. Ownership can be exclusive or *joint*, i.e., a property may be owned by one owner or shared by several owners. Person$\prec\!\cdots$Retirement-Portfolio is an example of exclusive ownership.

There are two types of joint ownerships. *Free joint* ownership states no explicit partition of the rights of the joint owners in the property. For example, a joint bank account is freely shared by a couple. In *percentage joint* ownership, each owner takes a percentage of the ownership, e.g., when husband and wife each owns 50% of their house. An *equal joint* is when all owners have the same percentage. As noted in [17], percentage joint is unique to ownership, while exclusiveness also concerns other generic relationships.

Partitioning. In the ownership relationship, the owner may own several categories of properties each according to a given perspective. For example, consider the owner class Person. Viewed as a biological being, a person owns a brain and a heart. Viewed as a psychological being, a person possesses a certain responsibility and a personality. We have therefore two partitions: Person $\overset{\text{biology}}{\prec\!\cdots}$ $\{\text{Brain}, \text{Heart}\}$ and Person $\overset{\text{psychology}}{\prec\!\cdots\cdots}$ $\{\text{Responsibility}, \text{Personality}\}$.

7 New Generic Relationships

This section gives some guidelines to identify and define new generic relationships.

Information modeling focuses on capturing and representing certain aspects of the real world relevant to the functions of an information system. The central constructs in the building process of information models are entities (or types, classes) representing important things of the application domain, and relationships among those things.

When building an information model, it is relatively easy to identify adequate entities to capture real-world objects: they directly correspond to the important concepts naturally manipulated by stakeholders in the application domain. The research reviewed in this paper advocates the use of a rich repertoire of generic relationships for modeling relationships between entities. Thus, for the information modeler, the choice of appropriate relationships to associate objects is comparatively more difficult.

Various choices of relationships correspond to sometimes subtle differences in the shades of real-world semantics captured in an information model. For example, the relationship between books and their book copies is better modeled as a materialization than as an association.

In practice, when deciding on which relationship to use for modeling a relationship, the modeler has to choose between: (1) a usual association (like in Employee—$\boxed{\text{works}}$—Employer), (2) a specific relationship derived from a generic relationship within the repertoire of available generic relationships (such as the relationship Employee$\rightarrow\!\circ$Person derived from RoleClass$\rightarrow\!\circ$ObjectClass), and (3) a specific relationship derived from a new generic relationship identified in the application domain.

New relationships can be identified when the same pattern is repeatedly encountered and it does not fit well the available generic relationships, but the decision to define a new generic relationship should be made with care. For example, some candidate generic relationships can be best described as subcategories of already identified relationships (like the subcategories of aggregation discussed in, e.g., [24,42,46]).

When a new generic relationship has been tentatively identified, it must be well defined (i.e., it must be intuitively well understood), it should correspond to a significant number of specific instances validated in application domains, its semantics should be formalized, and it should be associated with an appropriate graphical notation.

The intuitive semantics of a generic relationship is a broad intent about the duties of the relationship in application domains, in the style of the short descriptions of Section 2. The formal semantics fits in two categories: the first one positions the relationship along the various characteristics reviewed in Section 5 while the second characterizes the inherent semantics of the relationship, in particular in terms of the semantics of creation, update, and deletion of objects involved in the relationship.

A graphical notation for a generic relationship includes a notation for participating classes and for the relationship itself. Notations of a varying degree of detail can be defined for participation constraints linked with cardinality (see, e.g., [24]).

The identification and definition of new relationships should carefully explore and characterize their similarities and interactions with existing relationships [10].

8 A Refined Metamodel for Relationships

This section presents our metamodel for generic relationships, based on their semantics as presented in Sections 4 and 5. Its general structure is shown in Figure 22. It is an elaborated version of the hierarchy in Figure 3.

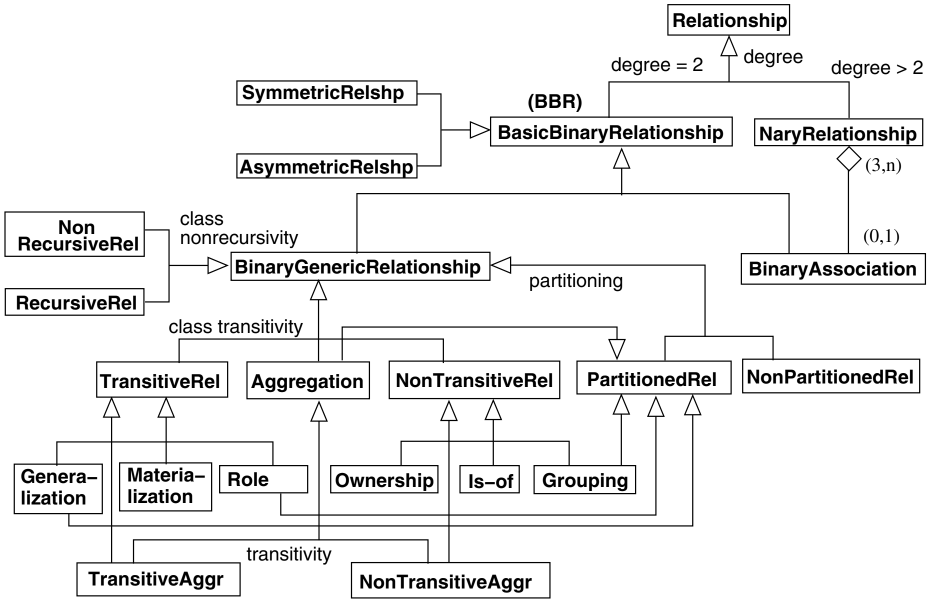


Fig. 22. Our metamodel of generic relationships

Metatype Relationship denotes all kinds of generic relationships. The first partitioning, on the top of the hierarchy, is based on the degree of relationships. We distinguish binary relationships, represented in the figure by metatype BasicBinaryRelationship, and n-ary relationships, represented by metatype NaryRelationship. BasicBinaryRelationship represents the basic binary relationship $\mathcal{BBR}$ described in Section 4. BasicBinaryRelationship is specialized in SymmetricRelshp and AsymmetricRelshp representing, respectively, symmetric and asymmetric relationships.

Metatype BasicBinaryRelationship is in turn classified in two categories: BinaryGenericRelationship representing binary generic relationships described in Section 5 and BinaryAssociation representing binary associations. This decomposition is needed for the following two reasons. First, as seen in Section 5, binary generic relationships (e.g., generalization, materialization, aggregation) can be described along dimensions such as multiplicity, composition, class transitivity, class nonrecursivity, and partitioning. These properties do not concern binary associations represented by metatype BinaryAssociation. Second, an n-ary relationship, represented by metatype NaryRelationship, is defined as an aggregate of three or more binary associations. Note that NaryRelationship cannot be defined as an aggregate of $\mathcal{BBR}$, because the latter also accounts for binary generic relationships such as generalization, aggregations, etc., that cannot compose an n-ary relationship.

Binary generic relationships are classified along class transitivity, class nonrecursivity, and partitioning, which only apply to some generic relationships.

Class-transitive relationships are grouped under category TransitiveRel while nontransitive ones go under category NonTransitiveRel. Aggregation cannot be directly placed under TransitiveRel nor under NonTransitiveRel because some of its categories are class transitive and others are not. The former are gathered under metatype TransitiveAggr and the latter under metatype NonTransitiveAggr. TransitiveAggr and NonTransitiveAggr are defined as subclasses of TransitiveRel and NonTransitiveRel, respectively. Similarly, recursive and nonrecursive generic relationships are represented in the metamodel by metatypes RecursiveRel and NonrecursiveRel.

In parallel to the specialization of binary generic relationships along the dimensions of class transitivity and class nonrecursivity, their specialization along the partitioning dimension gives rise to two metatypes: PartitionedRel representing binary generic relationships (e.g., generalization, aggregation, role, and grouping) which may be organized in several partitions along several discriminators and NonPartitionedRel representing relationships which may not.

This metamodel can be enriched if other dimensions for classifying binary generic relationships are identified.

9 Conclusion

This paper has discussed relationships in information modeling. We first defined a basic binary relationship $\mathcal{BBR}$ that both defines binary associations and represents the common semantics of generic relationships. The semantics of $\mathcal{BBR}$ was defined along several dimensions, including cardinality, existence dependency, exclusiveness and inclusion, symmetry and asymmetry, instance transitivity, and attribute propagations.

We then defined binary generic relationships as specializations of $\mathcal{BBR}$ and characterized them along several important dimensions like class and instance semantics, multiplicity, composition, class transitivity, class nonrecursivity, and partitioning. Then, several generic relationships were reviewed in the light of those dimensions. Therefore, instead of defining the semantics of generic relationships in an ad-hoc manner as is often done, those dimensions provide a useful support for a clear and systematic definition.

We also discussed how new generic relationships can be identified and defined. A new generic relationship should correspond to a significant number of specific instances validated in application domains, its semantics should be formalized according the dimensions above, and it should be associated with an appropriate graphical notation. The identification and definition of new relationships should carefully explore and characterize their similarities with existing relationships.

Finally, we defined a new metamodel for generic relationships based on their semantics.

We are currently comparing our metamodel for relationships with that proposed in UML 2.0. The study would result in the extension of UML with new generic relationships that have no equivalent in UML such as role-of and materialization.

References

1. J. Abrial. Data semantics. In *Proc. of the IFIP Working Conf. on Data Base Management*, pages 1–59. North-Holland, 1974.
2. A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An object data model with roles. In *Proc. of the 19th Int. Conf. on Very Large Data Bases, VLDB'93*, pages 39–51. Morgan Kaufmann, 1993.
3. E. Andonoff, G. Hubert, and A. Le Parc. Modeling inheritance, composition and relationship links between objects, object versions and class versions. In *Proc. of the 7th Int. Conf. on Advanced Information Systems Engineering, CAiSE'95*, LNCS 932, pages 96–111. Springer-Verlag, 1995.
4. M. Brodie. Association: A database abstraction. In P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 583–608. North-Holland, 1981.
5. R. Cattell, D. Barry, M. Berler, and J. Eastman, editors. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann, 2000.
6. W. Chu and G. Zhang. Associations and roles in object-oriented modeling. In *Proc. of the 16th Int. Conf. on Conceptual Modeling, ER'97*, LNCS 1331, pages 257–270. Springer-Verlag, 1997.
7. P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors. *Proc. of the 8th Int. Conf. on Advanced Information Systems Engineering, CAiSE'96*, LNCS 1080. Springer-Verlag, 1996.
8. M. Dahchour. *Integrating Generic Relationships into Object Models Using Metaclasses.* PhD thesis, Département d'ingénierie informatique, Université catholique de Louvain, Belgium, 2001.
9. M. Dahchour, A. Pirotte, and E. Zimányi. Materialization and its metaclass implementation. *IEEE Trans. on Knowledge and Data Engineering*, 14(5):1078–1094, 2002.
10. M. Dahchour, A. Pirotte, and E. Zimányi. A role model and its metaclass implementation. *Information Systems*, 29(3):235–270, 2004.
11. K. Davis, G. Dong, and A. Heuer. Discussion report: Object migration and classification. In *Proc. of the 4th Int. Workshop on Foundations of Models and Languages for Data and Objects*, pages 223–227. Springer-Verlag, 1992.
12. D. Firesmith, B. Henderson-Sellers, and I. Graham. *OPEN Modeling Language OML Reference Manual.* SIGS Books, 1997.
13. G. Gottlob, M. Schrefl, and B. Röck. Extending object-oriented systems with roles. *ACM Trans. on Office Information Systems*, 14(3):268–296, 1996.
14. R. Gupta and G. Hall. An abstraction mechanism for modeling generation. In *Proc. of the 8th Int. Conf. on Data Engineering, ICDE'92*, pages 650–658. IEEE Computer Society, 1992.
15. G. Hall and R. Gupta. Modeling transition. In *Proc. of the 7th Int. Conf. on Data Engineering, ICDE'91*, pages 540–549. IEEE Computer Society, 1991.

16. M. Halper, J. Geller, and Y. Perl. An OODB part-whole model: Semantics, notation, and implementation. *Data & Knowledge Engineering*, 27(1):59–95, 1998.
17. M. Halper, Y. Perl, O. Yang, and J. Geller. Modeling business applications with the OODB ownership relationship. In *Proc. of the 3rd Int. Conf. on AI Applications on Wall Street*, pages 2–10, 1995.
18. T. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design.* Morgan Kaufmann, 2001.
19. B. Henderson-Sellers. OPEN relationships: Compositions and containments. *Journal of Object-Oriented Programming*, 10(7):51–55, 1997.
20. T. Jones and I. Song. Binary equivalents of ternary relationships in entity-relationship modeling: A logical decomposition approach. *Journal of Database Management*, 11(2):12–19, 2000.
21. R. Katz. Towards a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, 1990.
22. W. Kim, E. Bertino, and J. Garza. Composite objects revisited. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, SIGMOD'89*, pages 337–347, 1989. SIGMOD Record 18(2).
23. M. Kolp. *A Metaobject Protocol for Integrating Full-Fledged Relationships into Reflective Systems.* PhD thesis, INFODOC, Université Libre de Bruxelles, Belgium, 1999.
24. M. Kolp and A. Pirotte. An aggregation model and its C++ implementation. In *Proc. of the 4th Int. Conf. on Object-Oriented Information Systems, OOIS'97*, pages 211–224, 1997.
25. Y. Lahlou and N. Mouaddib. Relaxing the instantiation link: Towards a content-based data model for information retrieval. In Constantopoulos et al. [7], pages 540–561.
26. A. Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, 24(11):908–926, 1998.
27. H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications, OOPSLA'86*, pages 214–223, 1986. ACM SIGPLAN Notices 21(11), 1986.
28. T. Ling. A normal form for entity-relationship diagrams. In *Proc. of the 4th Int. Conf. on the Entity-Relationship Approach, ER'85*, pages 24–35, 1985.
29. N. Mattos. Abstraction concepts: The basis for data and knowledge modelling. In *Proc. of the 7th Int. Conf. on the Entity-Relationship Approach, ER'88*, pages 473–492, 1988.
30. R. Motschnig-Pitrik and J. Kaasboll. Part-whole relationship categories and their application in object-oriented analysis. *IEEE Trans. on Knowledge and Data Engineering*, 11(5):779–797, 1999.
31. R. Motschnig-Pitrik and J. Mylopoulos. Classes and instances. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):61–92, 1992.
32. R. Motschnig-Pitrik and J. Mylopoulos. Semantics, features, and applications of the viewpoint abstraction. In Constantopoulos et al. [7], pages 514–539.
33. R. Motschnig-Pitrik and V. Storey. Modelling of set membership: The notion and the issues. *Data & Knowledge Engineering*, 16(2):147–185, 1995.
34. J. Mylopoulos. Information modeling in the time of the revolution. *Information Systems*, 23(3–4):127–155, 1998.

35. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about informations systems. *ACM Trans. on Office Information Systems*, 8(4):325–362, 1990.
36. J. Odell. Six different kinds of composition. *Journal of Object-Oriented Programming*, 6(8):10–15, 1994.
37. C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS approach.* Springer, 2005, to appear.
38. J. Peckham, B. MacKellar, and M. Doherty. Data model for extensible support of explicit relationships in design databases. *Very Large Data Bases Journal*, 4(2):157–191, 1995.
39. A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva. Materialization: a powerful and ubiquitous abstraction pattern. In *Proc. of the 20th Int. Conf. on Very Large Data Bases, VLDB'94*, pages 630–641, 1994. Morgan Kaufmann.
40. D. Renouf and B. Henderson-Sellers. Incorporating roles into MOSES. In *Proc. of the 15th Conf. on Technology of Object-Oriented Languages and Systems, TOOLS 15*, pages 71–82, 1995.
41. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language: Reference Manual.* Addison-Wesley, second edition, 2004.
42. V. Storey. Understanding semantic relationships. *Very Large Data Bases Journal*, 2(4):455–488, 1993.
43. T. Teorey. *Database Modeling and Design.* Morgan Kaufmann, third edition, 1999.
44. J. Wäsch and K. Aberer. Flexible design and efficient implementation of a hypermedia document database system by tailoring semantic relationships. In *Proc. of the IFIP WG2.6 6th Working Conf. on Database Semantics, DS-6*, pages 367–388. Chapman & Hall, 1995.
45. R. Wieringa, W. De Jonge, and P. Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.
46. M. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.
47. R. Wong, H. Chau, and F. Lochovsky. A data model and semantics of objects with dynamic roles. In *Proc. of the 13th Int. Conf. on Data Engineering, ICDE'97*, pages 402–411. IEEE Computer Society, 1997.
48. O. Yang, M. Halper, J. Geller, and Y. Perl. The OODB ownership relationship. In *Proc. of the Int. Conf. on Object-Oriented Information Systems, OOIS'94*, pages 278–291. Springer-Verlag, 1994.
49. E. Yu, L. Liu, and Y. Li. Modelling strategic actor relationships to support intellectual property management. In *Proc. of the 20th Int. Conf. on Conceptual Modeling, ER 2001*, pages 164–178, 2001.

EMMA – A Formal Basis for Querying Enhanced Multimedia Meta Objects

Sonja Zillner and Werner Winiwarter

Faculty of Computer Science,
University of Vienna,
Liebiggasse 4, A-1010 Vienna
sonja.zillner@univie.ac.at

Abstract. Today's multimedia content formats primarily encode the presentation of content but not the information the content conveys. However, this presentation-oriented modeling only permits the inflexible, hard-wired presentation of multimedia content. For the realization of advanced operations like the retrieval and reuse of content, automatic composition, or adaptation to a user's needs, the multimedia content has to be enriched by additional semantic information, e.g. the semantic interrelationships between single multimedia content items. Enhanced Multimedia Meta Objects (EMMOs) are a novel approach to multimedia content modeling, which combines media, semantic relationships between those media, as well as functionality on the media (such as rendering) into tradeable and versionable knowledge-enriched units of multimedia content. For the processing of EMMOs and the knowledge they incorporate, suitable querying facilities are required. Based on the formal definition of the EMMO model, in this paper, we propose and formally define the EMMO Algebra EMMA, a query algebra that is adequate and complete with regard to the EMMO model. EMMA offers a rich set of orthogonal query operators, which are sufficiently expressive to provide access to all aspects of EMMOs and enable efficient query rewriting and optimization. In addition, they allow for the seamless integration of ontological knowledge within queries, such as supertype/subtype relationships, transitive and inverse associations, etc. Thus, EMMA represents a sound and adequate foundation for the realization of powerful EMMO querying facilities. We have finished the implementation of an EMMO container environment and an EMMA query execution engine, and are currently in the process of evaluating the query algebra in several case studies.

1 Introduction

For the presentation and rendering of multimedia content, there exist several multimedia content formats, such as HTML [1], SMIL [2], or SVG [3]. Those approaches have all in common that they merely focus on the encoding of the content, but neglect the information the content conveys. As those multimedia content formats are limited to the modeling of presentation-related issues of multimedia content, only the generation of inflexible hard-wired multimedia presentations can be realized. As a prerequisite for advanced operations, such as retrieval and reuse of content, automatic composition, and adaptation of content to a user's needs, additional information about the content's semantics has to be provided. Triggered by the research in the context of the Semantic Web initiative [4], several attempts have been undertaken to integrate semantics into the modeling of multimedia content. For recent publications on multimedia semantics see [5], [6], [7], [8], [9], [10], an up-to-date overview is given in [11].

To facilitate the semantic modeling of multimedia content in content sharing and collaborative applications, we have developed *Enhanced Multimedia Meta Objects (EMMOs)* [12] in the context of the EU-funded CULTOS project[1]. An EMMO establishes a self-contained unit of multimedia content indivisibly unifying three aspects of multimedia content: The *media aspect* aggregates the basic media objects of an EMMO, the *semantic aspect* enables the specification of semantic associations between an EMMO's media objects, and finally the *functional aspect* permits EMMOs to define arbitrary, domain-specific operations that can be invoked by applications. Moreover, by providing *versioning* support, EMMOs can be modified concurrently within a distributed environment. As all three aspects of multimedia content and the versioning information can be bundled into one unit and serialized into an exchangeable format, EMMOs establish *tradeable*, semantically enriched units of multimedia content.

In contrast to common approaches for the representation of multimedia content, as well as existing standards for modeling the content's semantics, EMMOs establish a unique way for the semantic modeling of multimedia content. Popular standards for *multimedia document models*, such as HTML [1], XHTML+SMIL [13], HyTime [14], MHEG-5 [15], MPEG-4 BIFS and XMT [16], SMIL [2], or SVG [3], model the presentation of content by arranging basic media objects according to temporal, spatial, and interaction relationships. Therefore, they mainly cover the content's media aspect, but disregard the semantic and functional aspects of content, and provide no versioning support. Standards for *modeling semantics*, such as RDF [17, 18], Topic Maps [19], MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [20]), or Conceptual Graphs [21], clearly provide means for describing the semantic aspect of content. However, they rather neglect the media aspect and functional aspect, and also do not provide versioning support.

Within the CULTOS project, a distributed infrastructure of *EMMO containers* [22] and an *authoring tool* for the creation of EMMOs were developed. For the realization of advanced operations on EMMOs, efficient retrieval and processing of the information captured by EMMOs was still missing after the completion of the CULTOS project. Therefore, we have developed the query algebra EMMA, which provides a formal basis for querying EMMOs.

[1] CULTOS was carried out from 2001 to 2003 by partners from 11 EU countries and Israel. It aimed at providing a collaborative multimedia platform for researchers in intertextual studies enabling them to share and communicate their knowledge about the relationships between cultural artifacts. See `http://www.cultos.org` for more information.

The contribution of this paper is to introduce the formal foundation of the query algebra EMMA. The paper builds on, revises, and extends previous research work published in [12], [23], and [24]. By addressing an EMMO's media, semantic, and functional aspect, as well as its versioning information, EMMA is adequate and complete with regard to the EMMO model. EMMA comprises an extensive set of simple and orthogonal query operators (extraction operators, navigational operators, selection predicates, constructors, and a join operator), which allow the construction of more complex queries against EMMOs, thus providing the basis for efficient query rewriting and optimization.

The remainder of the paper is organized as follows. Section 2 introduces and formally defines the EMMO model, Sect. 3 discusses the requirements of a query algebra for EMMOs. Section 4 takes a look at related approaches and Sect. 5 introduces a representative selection of EMMA's formal foundation along with illustrative examples. Section 6 briefly introduces the EMMO and EMMA implementation and, finally, Sect. 7 concludes this paper and gives an outlook on future work.

2 The EMMO Model

As mentioned before, EMMOs establish tradable, knowledge-enriched units of multimedia content that indivisibly combine the content's media, semantic, and functional aspect, as well as its versioning information into one single object.

The formal components of the EMMO model are *entities*, which occur in four different kinds – *logical media parts* representing media objects or parts of media objects, *ontology objects* representing concepts of an ontology, *associations* modeling binary relationships between entities, and *EMMOs* establishing an aggregation of semantically related entities.

In the following subsections, we formally define entities and their four specializations and use real-world example EMMOs originating from the CULTOS project (see [25]) to illustrate the EMMO model. Moreover, to exemplify the integration of domain knowledge, we use an extended version of the Ontology of Intertextual Studies [26].

2.1 Entities

Each entity w is characterized by thirteen properties:

– Each entity w has a *global* and *unique object identifier (OID)* o_w represented as universal unique identifier (UUID) [27], which enables the unique identification of entities in distributed scenarios.
– As UUIDs are not really useful for humans, each entity w has also a human readable *name* n_w expressed as string value.
– For classifying whether an entity w is a logical media part, an ontology object, an association, or an EMMO, its *kind* k_w is specified accordingly.

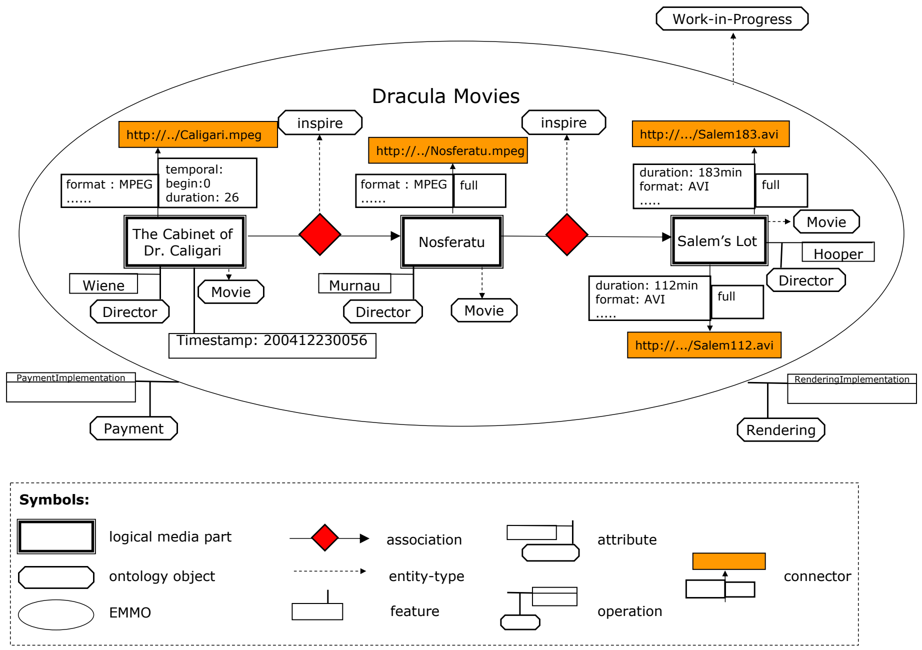


Fig. 1. EMMO "Dracula Movies" (e_{movies})

– Each entity w is described by a set of *types* T_w, i.e. a set of ontology objects, enabling the classification by concepts taken from a domain ontology, e.g. entity w might be an instantiation of the concepts "Ancient Text" and "Novel", or an instantiation of the concept "Movie". The semantics of ontology objects is specified in the underlying domain ontology, e.g. within the ontology structure represented in Fig. 2.
– Each entity possesses an arbitrary number of application-dependent *attributes* A_w. Attributes are represented as attribute-value pairs with the attribute name being a concept of a domain ontology, e.g. by attaching the value "Murnau" for the attribute "Director" to the entity representing the movie "Nosferatu" (see Fig. 1), one can express that the movie was directed by Murnau. The attribute value is per default untyped, however, typing constraints can be introduced via the domain ontology (see Fig. 2).
– For providing versioning support, a set of *preceding versions* P_w and *succeeding versions* S_w can be assigned to each entity w. Each version of w is again an entity of the same kind k_w. By also treating an entity's versions as entities, different versions of an entity can be interrelated just like any other entities, thus allowing one to establish relationships between entity versions. Figure 3 shows several versions of the EMMO "Dracula Movies" and their interrelationships.
– As it might be necessary in an implementation of the model, to augment an entity w with further low-level data, such as timestamps or status infor-

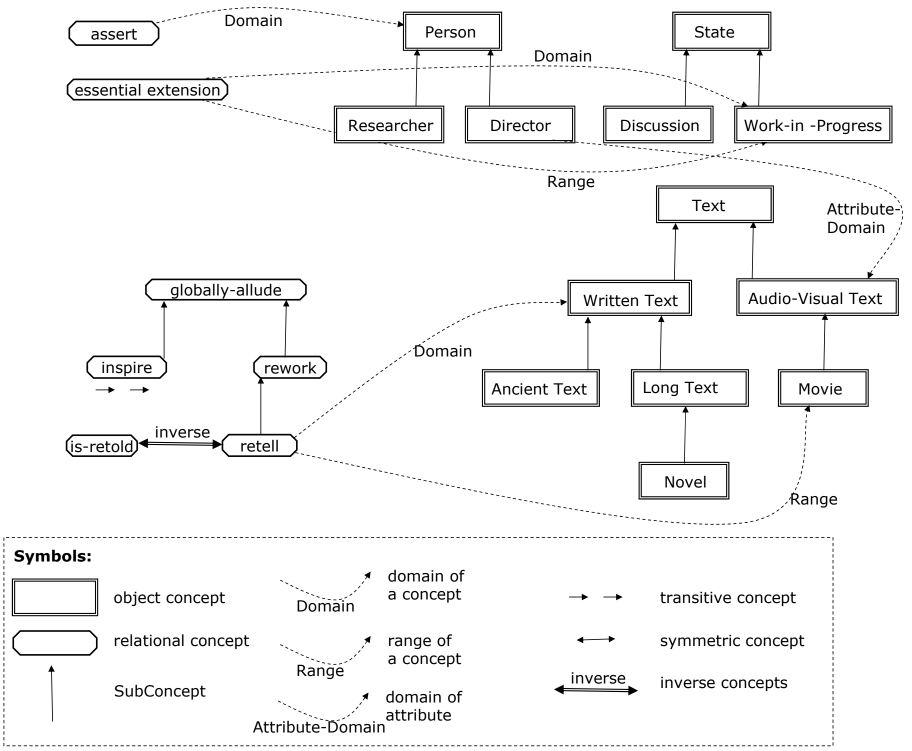


Fig. 2. Graphical representation of a part of an Ontology of Intertextual Studies

mation, in a flexible, ad-hoc manner, a set of *features* F_w, represented as feature-value pairs, can be attached to the entity. In contrast to attributes, feature names are not ontology objects but simple strings.

As the remaining properties of an entity w are only relevant for certain kinds of entities, at this point we will only provide a brief explanation as far as it is necessary for the understanding of the following definitions; we will provide more detailed definitions and examples in the following subsections.

- By specifying exactly one *source* and *target entity* s_w and t_w, an *association* establishes a directed binary relationship between those entities.
- The *connectors* C_w establish a connection to the physical media data of a *logical media part*. Each connector consists of a *media profile* which describes the storage location by either embedding the *raw media data* or by referencing the media data via a *URI*, and of a *media selector*, which provides means to address only selected parts of the media object.
- An *EMMO* constitutes a container of all entities specified in the set *nodes* N_w.

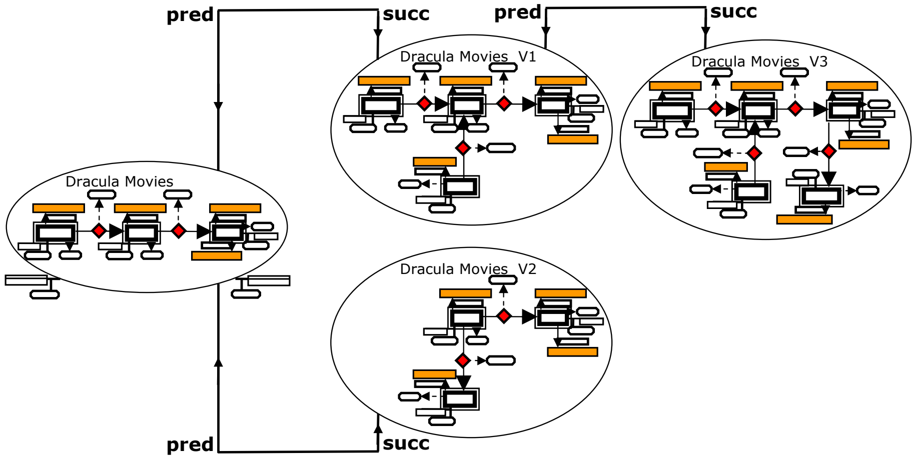


Fig. 3. The versioning information of EMMO "Dracula Movies"

– An *EMMO* offers *operations* O_w, which can be invoked by external applications. The implementation of an operation is described by a mathematical *function.*

After this informal intuitive description, we are now ready to provide a formal definition of an entity. First we define some basic symbols we will use throughout the rest of this paper.

Definition 1. *[Symbols] Let* Γ *denote the set of all logical media parts,* Θ *the set of all ontology objects,* Λ *the set of all associations,* Σ *the set of all Emmos, and* $\Omega = \Gamma \cup \Theta \cup \Lambda \cup \Sigma$ *the set of all entities. Further, let* $\mathcal{MS}$ *be the set of all media selectors,* $\mathcal{MP}$ *the set of all media profiles,* $\mathcal{OP}$ *the set of all operations. Finally, let* $\mathbb{VAL}$ *be the set of all untyped data values,* $\mathbb{UUID} \subset \mathbb{VAL}$ *the set of all universal unique identifiers,* $\mathbb{STR} \subset \mathbb{VAL}$ *the set of all strings,* $\mathbb{URI} \subset \mathbb{STR}$ *the set of all uniform resource identifiers,* $\mathbb{RMD}$ *the set of all raw media data, and* $\mathbb{FUN}$ *the set of all functions.*

On the basis of these common symbols, we define entities as follows.

Definition 2. *[Entity] An entity* $w \in \Omega$ *is a thirteen-tuple* $w = (o_w, n_w, k_w, s_w, t_w, T_w, A_w, C_w, N_w, P_w, S_w, F_w, O_w)$*, where* $o_w \in \mathbb{UUID}$ *denotes the unique object identifier (OID) of* w*,* $n_w \in \mathbb{STR}$ *the name of* w*,* $k_w \in \{\text{"}lmp\text{"}, \text{"}ont\text{"}, \text{"}asso\text{"}, \text{"}emm\text{"}\}$ *the kind of* w*,* $s_w \in \Omega \cup \{\varepsilon\}$ *the source and* $t_w \in \Omega \cup \{\varepsilon\}$ *the target entity of* w *with* $\varepsilon \notin \Omega$ *stating that such an entity is undefined,* $A_w \subseteq \Theta \times \mathbb{VAL}$ *the attributes,* $T_w \subseteq \Theta$ *the types,* $C_w \subseteq \mathcal{MS} \times \mathcal{MP}$ *the connectors,* $N_w \subseteq \Omega$ *the nodes,* $P_w \subseteq \Omega$ *the predecessors,* $S_w \subseteq \Omega$ *the successors,* $F_w \subseteq \mathbb{STR} \times \mathbb{VAL}$ *the features, and* $O_w \subseteq \mathcal{OP}$ *the operations of* w*. The following constraints hold for all entities:*

$$\forall w_1, w_2 \in \Omega : o_{w_1} = o_{w_2} \longrightarrow w_1 = w_2 \tag{1}$$

$$\forall w, v \in \Omega : v \in P_w \vee v \in S_w \longrightarrow k_w = k_v \tag{2}$$

Constraint (1) enforces that each entity has a unique identifer and Constraint (2) assures that each version of w is again an entity of the same kind k_w.

2.2 Logical Media Parts

Logical media parts are entities which enable the representation of media objects or parts of media objects at a logical level, and thus address an EMMO's *media aspect.* By decoupling the logical media part from any existing physical representation, a person who is not owing a media object can still use it within an EMMO. To express the difference between, for example, the movie "Salem's Lot" directed by Tobe Hooper and its underlying source material, the novel "Salem's Lot" written by Stephen King, the movie and the novel are modeled as two different logical media parts.

Definition 3. *[Logical media part] A logical media part $l \in \Gamma$ is an entity with $k_l = "lmp" \wedge s_l = t_l = \varepsilon \wedge N_l = O_l = \emptyset$.*

By means of *connectors* C_w, logical media parts not only model media objects at a logical level but additionally maintain connections to physical media data representing these objects, and thus provide the media aspect of multimedia content represented within the EMMO model. Connectors (see Def. 2) consist of a *media profile* representing the physical media data and of a *media selector* addressing the parts of the media data represented by the profile according to textual, spatial, and temporal criteria.

As formally defined in Def. 4, a media profile combines the storage location, which is called – following the MPEG-7 terminology – *media instance*, with low-level *metadata*, such as storage format or file size. The *media instance* either directly embeds media data or – if embedding is not feasible, e.g. because the media data is a live stream – references media data via a URI.

Definition 4. *[Media profile] A media profile $mp = (i_{mp}, M_{mp}) \in \mathcal{MP}$ is described by its media instance $i_{mp} \in \mathbb{URI} \cup \mathbb{RMD}$ and its metadata $M_{mp} \subseteq \mathbb{STR} \times \mathbb{VAL}$.*

Media selectors (see Def. 5) render it possible to address only selected parts of the physical media data, such as the introductory section of a movie from the first until the 26th minute, without having to extract that part, for instance, by putting the scene into a separate file using an audio editing tool.

Definition 5. *[Media selector] A media selector $ms = (k_{ms}, P_{ms}) \in \mathcal{MS}$ is described by its kind $k_{ms} \in \{$"spatial", "textual", "temporal", "full"$\}$ and by its parameters $P_{ms} \subseteq \mathbb{STR} \times \mathbb{VAL}$.*

In Example 1 we illustrate how the three logical media parts depicted in Fig. 1 representing the media objects "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" can be formally described within the EMMO model. The symbols $l_{caligari}$, $l_{nosferatu}$, and l_{salem} represent the three logical media parts. For example, the thirteen-tuple $l_{caligari}$ indicates that there exists an entity which is uniquely identified by the OID "l2471", is named "The Cabinet of Dr. Caligari", is of kind logical media part ("lmp"), specifies no source and target entity, is classified as "Movie", has the value "Wiene" for the attribute "Director", describes its physical media data by the connector (ms_1, mp_1), is augmented by its timestamp information, and specifies its sets of nodes, predecessors, successors, and operations as empty. The connector (ms_1, mp_1) references the temporal selection of the first 26 minutes from the MPEG-movie "`Caligari.mpeg`". (ms_2, mp_2) represents the connector of the logical media part $l_{nosferatu}$ associating the complete MPEG-movie "Nosferatu.mpeg", and, finally, (ms_3, mp_3) and (ms_4, mp_4) represent two versions of different length of the movie "Salem's Lot".

Example 1.

$$\begin{aligned}
l_{caligari} =& (\text{“}l2471\text{”}, \text{“The Cabinet of Dr. Caligari”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Wiene”})\},\\
& \{(ms_1, mp_1)\}, \emptyset, \emptyset, \emptyset, \{(\text{“timestamp”}, \text{“200412230056”})\}, \emptyset),\\
l_{nosferatu} =& (\text{“}l9462\text{”}, \text{“Nosferatu”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Murnau”})\},\\
& \{(ms_2, mp_2)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
l_{salem} =& (\text{“}l6231\text{”}, \text{“Salem's Lot”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Hooper”})\},\\
& \{(ms_3, mp_3), (ms_4, mp_4)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
ms_1 =& (\text{“temporal”}, \{(\text{“begin”}, 0), (\text{“duration”}, 26)\}),\\
mp_1 =& (\text{“}\texttt{www.../Caligari.mpeg}\text{”}, \{(\text{“format”}, \text{“MPEG”})\}),\\
ms_2 =& (\text{“full”}, \emptyset),\\
mp_2 =& (\text{“}\texttt{www.../Nosferatu.mpeg}\text{”}, \{(\text{“format”}, \text{“MPEG”})\}),\\
ms_3 =& (\text{“full”}, \emptyset),\\
mp_3 =& (\text{“}\texttt{www.../Salem183.avi}\text{”}, \{(\text{“format”}, \text{“AVI”}), (\text{“duration”}, 183)\}),\\
ms_4 =& (\text{“full”}, \emptyset),\\
mp_4 =& (\text{“}\texttt{www.../Salem112.avi}\text{”}, \{(\text{“format”}, \text{“AVI”}), (\text{“duration”}, 112)\}).
\end{aligned}$$

2.3 Ontology Objects

Ontology objects are entities that represent concepts of an ontology. By providing the basis for the description of entities and other properties by concepts taken from an ontology, ontology objects contribute to the *semantic aspect* of multimedia content modeling. Within the EMMO model, ontology objects are applied in four different ways, i.e. they are used:

– for designating the types of entities,
– for designating the attributes of attribute values,
– for designating the operations attached to EMMOs (see Def. 9),
– as nodes within the EMMO knowledge structure (see Sect. 2.5).

Definition 6. *[Ontology object] An ontology object $o \in \Theta$ is an entity with $k_o =$ "ont" $\wedge$ $s_o = t_o = \varepsilon \wedge C_o = N_o = O_o = \emptyset$.*

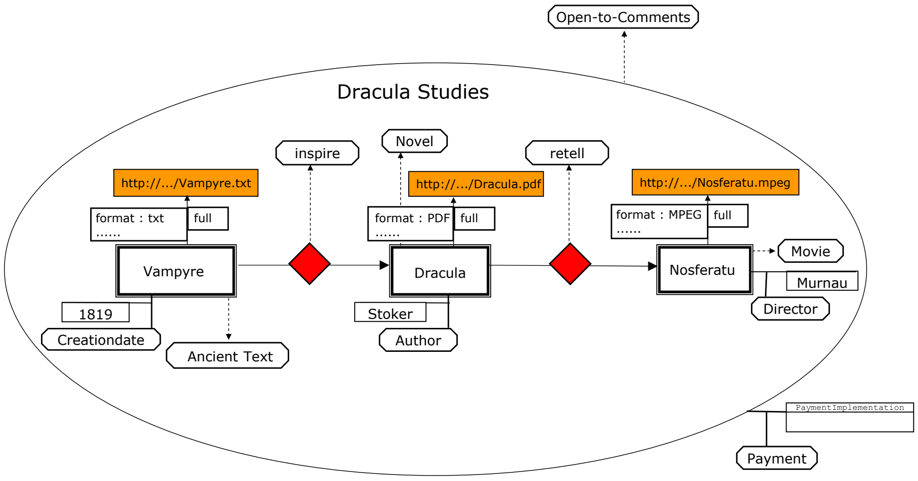


Fig. 4. EMMO "Dracula Studies"($e_{studies}$)

As can be seen from Def. 6, the types T_o of an ontology object o can be a non-empty set, i.e. ontology objects can again be classified by other ontology objects. This provides the basis for expressing ontological structures within the EMMO model. The development of a dedicated ontology engineering environment is focus of future work (see [28], [29], [30]). The final aim is the seamless integration of ontological knowledge into the EMMO model.

In Example 2 all four different ways of using ontology objects are illustrated: Within the EMMOs "Dracula Studies" and "Dracula Research"(see Fig. 4 and Fig. 5), ontology object $o_{inspire}$ represents the type of the association connecting the two logical media parts "Vampyre" and "Dracula", ontology object o_{movie} the type of the logical media part "Nosferatu"; ontology object $o_{director}$ is used as name of the attribute attached to the logical media part "Nosferatu", and ontology object $o_{payment}$ represents the designator of the operation provided by EMMO "Dracula Studies". Moreover, the ontology object o_{miller}, which represents the concept "Elizabeth Miller", is specified as node contained within EMMO "Dracula Research", and by additionally typing this ontology object with the ontology object $o_{researcher}$, "Elizabeth Miller" is classified as "Researcher".

Example 2.

$$
\begin{aligned}
o_{inspire} &= (\text{“}o8421\text{”}, \text{“inspire”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{movie} &= (\text{“}o4302\text{”}, \text{“Movie”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{director} &= (\text{“}o3418\text{”}, \text{“Director”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{payment} &= (\text{“}o6445\text{”}, \text{“Payment”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{miller} &= (\text{“}o3021\text{”}, \text{“Elizabeth Miller”}, \text{“ont”}, \epsilon, \epsilon, \{o_{researcher}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{researcher} &= (\text{“}o2166\text{”}, \text{“Researcher”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}
$$

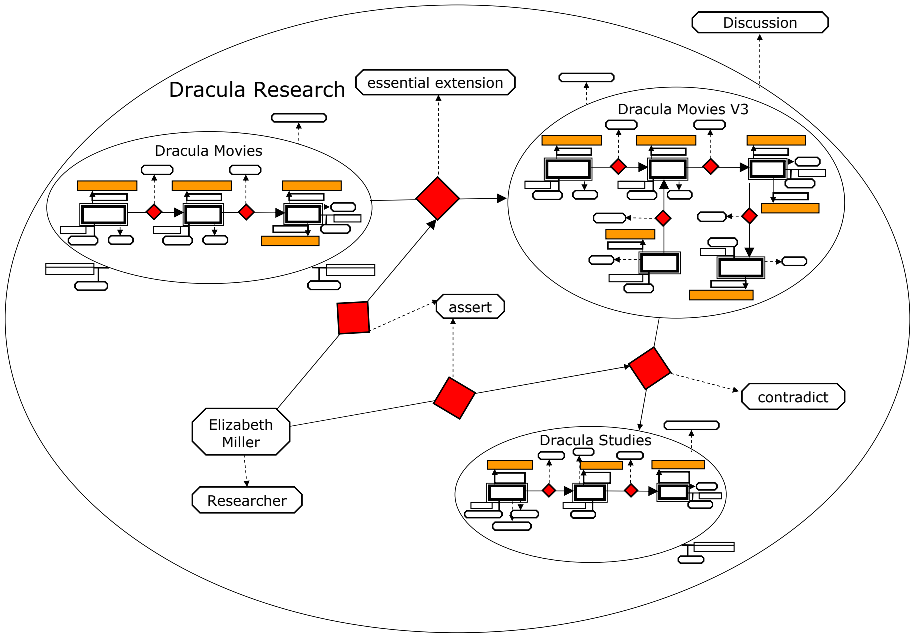


Fig. 5. EMMO "Dracula Research" ($e_{research}$)

2.4 Associations

Associations describe binary directed semantic relationships between entities. Thus, they contribute to the *semantic aspect* of multimedia content. By being modeled as entities, associations can take part in other associations, and thus facilitate the *reification* of statements in the EMMO model.

Definition 7. *[Association] An association* $a \in \Lambda$ *is an entity with* $k_a =$ "*asso*" $\wedge\ s_a \neq \varepsilon \wedge t_a \neq \varepsilon \wedge C_a = N_a = O_a = \emptyset \wedge |T_a| = 1$.

Similar to other entities, an association's type is represented by an ontology object and determines the kind of semantic relationship. Different from other entities, however, an association can only associate one type because it is supposed to represent only a unique kind of relationship. By specifying exactly one source and one target entity s_a and t_a, each association establishes a directed binary relationship between those two entities.

Example 3 shows the formal description of the two associations $a_{ca \to no}$ and $a_{no \to sa}$ contained within EMMO "Dracula Movies" (Fig. 1) and of the four associations $a_{mo \to moV3}$, $a_{moV3 \to st}$, $a_{mi \to (mo \to moV3)}$, and $a_{mi \to (moV3 \to st)}$ contained within EMMO "Dracula Research" (Fig. 5). Association $a_{mo \to moV3}$ models that EMMO "Dracula Movies V3" is an essential extension of EMMO "Dracula Movies", and by expressing that this statement was asserted by "Elizabeth Miller", association $a_{mi \to (mo \to moV3)}$ exemplifies the reification of statements.

Example 3.

$$\begin{aligned}
a_{ca\rightarrow no} =& (\text{“}a0225\text{”}, \text{“}ca \rightarrow no\text{”}, \text{“asso”}, l_{caligari}, l_{nosferatu}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{no\rightarrow sa} =& (\text{“}a5461\text{”}, \text{“}no \rightarrow sa\text{”}, \text{“asso”}, l_{nosferatu}, l_{salem}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mo\rightarrow moV3} =& (\text{“}a6390\text{”}, \text{“}mo \rightarrow moV3\text{”}, \text{“asso”}, e_{movies}, e_{moviesV3}, \{o_{essential\text{-}extension}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{moV3\rightarrow st} =& (\text{“}a5461\text{”}, \text{“}moV3 \rightarrow st\text{”}, \text{“asso”}, e_{moviesV3}, e_{studies}, \{o_{contradict}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mi\rightarrow(mo\rightarrow moV3)} =& (\text{“}a4771\text{”}, \text{“}mi \rightarrow (mo \rightarrow moV3)\text{”}, \text{“asso”}, o_{miller}, a_{mo\rightarrow moV3}, \{o_{assert}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mi\rightarrow(moV3\rightarrow st)} =& (\text{“}a7031\text{”}, \text{“}mi \rightarrow (moV3 \rightarrow st)\text{”}, \text{“asso”}, o_{miller}, a_{moV3\rightarrow st}, \{o_{assert}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}$$

2.5 EMMOs

An EMMO constitutes the core component of our model. It is a container that combines several entities into a single unit. By aggregating media data (i.e. logical media parts) and enriching this media data by semantic data (i.e. associations and ontology objects), an EMMO addresses the *media* and *semantic aspect* of multimedia content modeling. For instance, EMMO “Dracula Movies” groups the semantic descriptions of the logical media parts “The Cabinet of Dr. Caligari”, “Nosferatu”, and “Salem's Lot” into one single unit. Since EMMOs are modeled as entities, EMMOs can be contained within other EMMOs, just as any other entity. Therefore, a structure of hierarchically nested EMMOs can be established: EMMO “Dracula Research” in Fig. 5, for example, contains the EMMOs “Dracula Movies”, “Dracula Movies V3”, and “Dracula Studies”. Furthermore, an EMMO can also take part in associations, facilitating the representation of knowledge about the EMMO. For instance, within EMMO “Dracula Research” it is stated that EMMO “Dracula Movies V3” contradicts EMMO “Dracula Studies”. Finally, by specifying operations that process its content, EMMOs address the *functional aspect* of multimedia content.

Definition 8. *[EMMO] An EMMO $e \in \Sigma$ is an entity with $k_e = \text{"}emm\text{"}$, and $s_e = t_e = \varepsilon \wedge C_e = \emptyset$, such that*

$$\forall x \in N_e : \quad k_x = \text{"}asso\text{"} \longrightarrow \{s_x, t_x\} \subseteq N_e \tag{3}$$

According to this definition, an EMMO e constitutes a container of other entities because its set of nodes N_e is not restricted to an empty set, as it is the case with other kinds of entities. The contained entities form a connected graph structure when they are interlinked by associations. Constraint 3 ensures that associations can specify only those entities as source or target entity which already belong to the EMMO's nodes, and thus, guarantees that any established relationship is fully contained within the EMMO.

A further difference between EMMOs and the other kinds of entities is that its set of *operations* is not necessarily empty, allowing an EMMO to associate arbitrary operations. Within the EMMO model, an operation is a tuple combining an ontology object acting as the operation's *designator* with the operation's *implementation*, which can be described by any mathematical function.

Definition 9. *[Operation] An operation* $op = (d_{op}, i_{op}) \in \mathcal{OP}$ *is described by its designator* $d_{op} \in \Theta$ *and its implementation* $i_{op} \in \mathbb{FUN}$*.*

In Example 4, finally, the three EMMOs “Dracula Movies”, “Dracula Studies”, and “Dracula Research” are formally described: EMMO “Dracula Movies” consists of five nodes, i.e. the three logical media parts “The Cabinet of Dr. Caligari”, “Nosferatu”, and “Salem’s Lot”, and two associations; it defines EMMO “Dracula Movies V1” and EMMO “Dracula Movies V2” as its direct successor versions (see Fig. 3), and specifies the functions f_{render} implementing a “rendering” operation and $f_{payment}$ implementing a “payment transaction” operation. EMMO “Dracula Studies” aggregates five entities, i.e. the three logical media parts “Vampyre”, “Dracula”, and “Nosferatu”, as well as two associations, and offers a payment functionality. Finally, EMMO “Dracula Research” consists of eight nodes, i.e. the EMMOs “Dracula Movies”,“Dracula Movies V3”, and “Dracula Studies”, the ontology object “Elizabeth Miller”, and four associations.

Example 4.

$$\begin{aligned}
e_{movies} =&(\text{“}e7921\text{”}, \text{“}\mathsf{Dracula\ Movies}\text{”}, \text{“}\mathsf{emm}\text{”}, \epsilon, \epsilon, \{o_{\textit{work-in-progress}}\}, \emptyset, \emptyset, \{l_{caligari}, l_{nosferatu}, l_{salem},\\
&a_{ca \rightarrow no}, a_{no \rightarrow sa}\}, \emptyset, \{e_{moviesV1}, e_{moviesV2}\}, \emptyset, \{(o_{render}, f_{render}), (o_{payment}, f_{payment})\}),\\
e_{studies} =&(\text{“}e3811\text{”}, \text{“}\mathsf{Dracula\ Studies}\text{”}, \text{“}\mathsf{emm}\text{”}, \epsilon, \epsilon, \{o_{\textit{open-to-discussion}}\}, \emptyset, \emptyset,\\
&\{l_{vampyre}, l_{dracula}, l_{nosferatu}, a_{va \rightarrow dr}, a_{dr \rightarrow no}\}, \emptyset, \emptyset, \emptyset, \{(o_{payment}, f_{payment})\}),\\
e_{research} =&(\text{“}e1411\text{”}, \text{“}\mathsf{Dracula\ Research}\text{”}, \text{“}\mathsf{emm}\text{”}, \epsilon, \epsilon, \{o_{discussion}\}, \emptyset, \emptyset, \{e_{movies}, e_{moviesV3}, e_{studies},\\
&o_{miller}, a_{mo \rightarrow moV3}, a_{moV3 \rightarrow st}, a_{mi \rightarrow (mo \rightarrow moV3)}, a_{mi \rightarrow (moV3 \rightarrow st)}\}, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}$$

The integration of ontology knowledge enables to restrict the usage of associations and attributes, i.e. to define constraints on the knowledge structures within EMMOs. For example, by specifying the ontology represented in Fig. 2 as underlying domain ontology, it is determined that associations of type “retell” describe relationships pointing from entities of type “Written Text” to entities of type “Audio-Visual Text”. Within the axioms of the ontology, it can be additionally specified that integrity constraints on associations are extended to subconcepts, i.e. specifying entities of type “Audio-Visual Text” as permitted target entities then also includes entities of type “Movie” as permitted value. Moreover, within the ontology, one can specify the permitted domain of attributes, i.e. the types of entities to which they can be attached, for instance, the concept “Director” can only be used as attribute for entities of type “Audio-Visual Content”.

3 Requirements of a Query Algebra for EMMOs

For the realization of advanced operations on EMMO structures, a formal basis for querying of EMMOs, i.e. an algebra providing a set of formal query operators suitable for the EMMO model, is needed. The EMMO model has no inherent semantics, i.e. the particular semantics of an application scenario implementing the EMMO model is derived from the integrated domain ontology. Therefore, the requirements for accessing the information captured by EMMOs result from structural and syntactical issues, and have to be seen as being independent from the semantics of any particular application scenario. In the following, we introduce the essential requirements for such a query algebra.

The most important and fundamental prerequisite of such an algebra is to provide operators for accessing an EMMO's three aspects and its versioning information. Thus, the algebra has to offer operators enabling the access to:

- an EMMO's *media aspect*, i.e. operators that give access to logical media parts and their connectors;
- an EMMO's *semantic aspect*, i.e. operators that facilitate the retrieval of all kinds of entities contained in an EMMO, the querying of the types of entities and their attribute values, as well as the traversal of associations between entities; the operators must be expressive enough to cope with the more advanced constructs of the EMMO model, such as the reification of associations and the nesting of EMMOs;
- an EMMO's *functional aspect*, i.e. operators that allow the access to and permit the execution of the operations of an EMMO;
- an EMMO's *versioning information*, i.e. operators for the querying of an entity's direct and indirect versions.

In addition, an EMMO query algebra should meet basic query algebra requirements. Its operators should be formally defined with *precise semantics* to provide the basis for query rewriting and optimization. Furthermore, the operators should be *orthogonal* and arbitrarily nestable for enabling the formulation of expressive queries.

To combine information contained within different EMMOs, the algebra should support *joins* between entities. Moreover, a suitable algebra should support some basic construction and manipulation operators, such as union, intersection, and difference. However, since we have a graphical authoring tool available, such construction and manipulation operators can be kept simple.

Finally, because the EMMO model uses concepts of an ontology (i.e. ontology objects) to describe the meaning of the entities contained in an EMMO and the associations between them, a suitable EMMO query algebra should be expressive enough to *integrate ontological knowledge* into a query. Thus, for example, it should be possible to consider supertype/subtype relationships, transitive or inverse associations, etc.

A query algebra which is sufficiently expressive to fulfill all these requirements is said to be *adequate and complete* with regard to the EMMO model.

4 Related Approaches

On the search for a suitable query algebra for EMMOs, we will first take a brief look at object-oriented query approaches and query approaches for semi-structured data and multimedia content, before we analyze query approaches for semantic standards and examine their adequacy and completeness with regard to the EMMO model.

Although object-oriented database systems establish a graph-based data model, the object-oriented data model and the EMMO model are very different from each other, i.e. the former operates on the schema level and gathers objects of a particular type within one class, whereas the EMMO model operates on the instance level, defines no object classes, and leaves the specification of the type semantics to the integrated domain ontology. However, for accessing the information captured by EMMOs, i.e. an EMMO's three aspects and versioning information, one needs an adequate EMMO query approach. Therefore, we did not analyze query languages for object-oriented databases, such as OQL [31], AQUA Algebra [32], or XSQL [33], in further detail.

The Object Exchange Model OEM [34] is the widely accepted data model for semi-structured data. Similar to the EMMO model, OEM is schema-less and describes graph structures. Nevertheless, there are many differences between the two data models. OEM does neither address the three aspects of multimedia content, nor provides versioning support. Therefore, query languages for semi-structured data, such as Lorel [35], UnQL [36], SAL [37], XQuery [38], or XPath [39], are inadequate for querying EMMOs. However, query languages for semi-structured data provide a profound basis for graph navigation by establishing regular path expressions. Thus, we could use those query languages as inspiration for the design of the regular path expressions and navigational operators in EMMA (see Sect. 5.3).

In the literature, several query algebras for multimedia content have been proposed, such as GCalculus/S [40], Algebraic Video [41], or the Multimedia Presentation Algebra (MPA) [42]. These algebras have in common that they mainly address the *media aspect* of multimedia content. They focus on querying the temporal and spatial relationships between the basic media of multimedia content and the construction of new presentations out of these media. However, they ignore semantic relationships between media as well as the functional aspect of multimedia content.

In the context of the Semantic Web, several standards have emerged that can be used to model the semantic relationships between the basic media of multimedia content addressing the content's *semantic aspect*, such as RDF [17, 18], Topic Maps [19], and MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [20]). For these standards, a variety of proposals for query languages and algebras have been made.

Since the RDF data model, compared to the EMMO model, rather neglects the media aspect of multimedia content, it does not address the functional aspect of content, and does neither provide explicit support for versioning nor a hierarchical structuring of resource descriptions; the same is generally true for RDF-based query approaches as well. There are quite a few proposals for RDF query languages, such as RQL [43], SquishQL [44], or RAL [45], which can be used for querying the semantic aspect of multimedia content, but provide no means for querying the media and functional aspect or the versioning information of multimedia content. Thus, those approaches are incomplete and inadequate with regard to the EMMO model.

The situation for Topic Maps is quite similar to RDF. The Topic Map data model focuses on the semantic aspect and – considering the EMMO model's ability to include raw media data and metadata about the media by means of media profiles within an EMMO – neglects the media and functional aspects of multimedia content. Moreover, although Topic Maps can be hierarchically nested like EMMOs, they have no explicit versioning support. Consequently, query languages for Topic Maps are also in general incomplete and inadequate with regard to the EMMO model.

Within the context of the ongoing standardization of a Topic Maps Query Language TMQL [46], several approaches, such as Tolog [47], TMPath [48], XTMPath [49], or [50] have been introduced. However, those proposals remain again on the syntactic level and do not provide formal definitions of their operators. No formal algebra as a sound foundation for the querying of Topic Maps exists so far.

Finally, concerning the querying of semantic descriptions of multimedia content on the basis of MPEG-7's `Graph` tools, there are quite a few approaches adapting XQuery for the querying of MPEG-7 media descriptions (see [51]), but these approaches do not provide specific operators that would allow a reasonable processing of the `Graph` tools.

To summarize, looking at related work, we have not been able to find a formally sound foundation that would allow an adequate querying of EMMOs. Although there are some formal algebras available for querying the media aspect of multimedia content like GCalculus/S, Algebraic Video, or MPA, as well as for querying the semantic aspect of multimedia content, such as the RDF-based RAL, those algebras are neither adequate nor complete with regard to the EMMO model, which addresses the media, semantic, and the functional aspects of multimedia content, as well as versioning support.

As a consequence, we were forced to develop a dedicated query algebra to obtain a sound foundation for querying EMMOs. At least for the design of this algebra, we were able to gain valuable insights from the approaches we examined to incorporate certain aspects of their design.

5 EMMA Query Algebra

The design of the EMMO query algebra EMMA was in the first place driven by the requirement for accessing the complete information stored within an EMMO, i.e. the access to the three aspects of an EMMO, as well as its versioning information. To enable query optimization, the query algebra's operators are of limited complexity and orthogonal. Through the combination and nesting of modular operators, complex queries can be formulated.

There are five general classes of EMMA's query operators: the *extraction operators* provide the basis for querying an EMMO's three aspects, as well as its versioning information. The *navigational operators* enable the navigation along an EMMO's semantic graph structure and provide means for the integration of ontological knowledge. The *selection predicates* facilitate the selection of only those entities satisfying a specific condition, and the *constructors* enable the modification, combination, and creation of new EMMOs. Finally, the *join operator* relates several entities or EMMOs with a join condition.

Before we present the formal basis of the five operator classes in the following subsections, we will provide some definitions required for the understanding of the definitions to follow. To guarantee the readability of the paper, we will introduce the most representative EMMA operators by giving their formal definitions accompanied with illustrative real-world example queries originating from the CULTOS project. We will omit any EMMA operator which is used for accessing only some very specific aspects and information captured by the EMMO model. The complete list of formal definitions of EMMA operators can be found in [52].

To conclude this section, we will explain in a summary subsection how these operators contribute to fulfil the requirements for an EMMO query algebra.

5.1 Basic Definitions

The input and output values of EMMA operators, i.e. their signatures, are described by *sets* and *sequences.*

Definition 10. *[Set and Sequence] Let IN denote the set of all natural numbers, I an arbitrary index set, $\mathbb{BOO} = \{true, false\}$ the Boolean set, and $\mathbb{SET}$ the set of all sets. Let A and B be arbitrary sets, then $\mathcal{P}(A) = \{x \,|\, x \subseteq A\}$ denotes the powerset of A and $A \times B := \{(x, y) \,|\, x \in A \wedge y \in B\}$ the Cartesian product over A and B. The elements of a Cartesian product are called sequences or tuples. $\mathbb{SEQ}$ denotes the set of all sequences. A sequence of length 1 is equal to its single entry element, i.e. $\forall x\, (x) = x$. Let $j \in I$ then $\pi_j : \prod_{i\in I} A_i \longrightarrow A_j$ with $\pi_j(a_1, a_2, \ldots, a_n) = a_j$ denotes the j^{th} projection of $\prod_{i\in I} A_i$.*

EMMA operators are either *functions* or *predicates.*

Definition 11. *[Function and Predicate] Let $A, B \in \mathbb{SET}$ and $f \in \mathbb{FUN}$ with $f : A \longrightarrow B$ be a function, then $\mathcal{D}(f) = A$ denotes the domain and $\mathcal{R}(f) = B$ the range of function f, $\mathbb{FUN}_A$ the set of all functions with $\mathcal{D}(f) = A$, and $\mathbb{FUN}_{[A,B]}$ the set of all functions with $\mathcal{D}(f) = A$ and $\mathcal{R}(f) = B$. Furthermore, $p \in \mathbb{FUN}_{[A,\mathbb{BOO}]}$ denotes a predicate, $\mathbb{PRE}_A = \mathbb{FUN}_{[A,\mathbb{BOO}]}$ the set of all predicates with domain A, and $\mathbb{PRE} = \{\mathbb{PRE}_A \,|\, A \in \mathbb{SET}\}$ the set of all predicates. Let $f \in \mathbb{FUN}_{\prod_{i\in I} A_i}, j \in I, x \in A_j$ and $(a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \in \prod_{i\in I\setminus\{j\}} A_i$, then the function $f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]} : A_j \longrightarrow \mathbb{SET}$ with $f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]}(x) = f(a_1, \ldots, a_{j-1}, x, a_{j+1}, \ldots, a_n)$ is called f-projection onto A_j.*

EMMA operators are designed to be modular and simple. By using modular EMMA operators in combination with the operators *Apply* and *Elements*, more complex EMMA operators can be defined, and complex queries can be formulated. The operator *Apply* takes a function and a set as input values and returns the set consisting of all return values of the specified function being applied to each element in the specified set.

Definition 12. *[Apply] Let* $A \in \mathbb{SET}$ *and* $f \in \mathbb{FUN}$*, then the operator* $\mathit{Apply} : \mathbb{FUN} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *is defined as* $\mathit{Apply}(f, A) = \{f(x) \mid x \in A \cap \mathcal{D}(f)\}$.

The operator *Elements* is used to flatten data returned by other operations, e.g. for the specified input set it returns all elements being contained in at least one element of the specified set.

Definition 13. *[Elements] Let* $A \in \mathbb{SET}$*, then the operator* $\mathit{Elements} : \mathbb{SET} \longrightarrow \mathbb{SET}$ *is defined as* $\mathit{Elements}(A) = \{x \mid \exists X \in A \wedge x \in X\}$.

Additionally, for enabling the combination and nesting of EMMA operators, their signatures are always specified in the most general way, i.e. their input and output values are specified as set of entities. Thus, operators which only return valid results if applied to specific kinds of entities can still be applied to other kinds of entities yielding an empty result.

5.2 Extraction Operators

The extraction operators render it possible to access the information stored within an EMMO. In the following, we define a representative subset of the extraction operators for the three different aspects, as well as for the versioning information.

Media Aspect. Logical media parts model media objects at a logical level and maintain connections to their physical representations, i.e. to their media profiles and media selectors. For accessing the information described by a logical media part's connectors, EMMA defines several modular operators, as well as some more complex operators defined by nesting those modular operators. For example, the operator *MediaProfiles* can be used for locating media profiles, i.e. applying the operator *MediaProfiles* to a logical media part returns the union of all its associated media profiles, e.g. (see Fig. 1) the query expression

$$\mathit{MediaProfiles}(l_{salem}) = \big\{\big(\texttt{www.../Salem183.avi}, \big\{\big(\text{``}\mathit{duration}\text{''}, \mathit{183}\big), \big(\text{``}\mathit{format}\text{''}, \text{``}\mathit{AVI}\text{''}\big)\big\}\big),$$
$$\big(\texttt{www.../Salem112.avi}, \big\{\big(\text{``}\mathit{duration}\text{''}, \mathit{112}\big), \big(\text{``}\mathit{format}\text{''}, \text{``}\mathit{AVI}\text{''}\big)\big\}\big)\big\}$$

gives a set of two media profiles, each of them consisting of a URI locating the media data and a metadata set describing the low-level characteristics of the media data. The operator *MediaProfiles* is defined as a combination of the operators *connectors* and *MediaProfile.* For a specified entity, the operator *connectors* returns its set of connectors, and the operator *MediaProfile* returns the media profile for a given connector. By using the operators *Apply* and *Elements* in its definition, the operator *MediaProfiles* can be used to access the union of associated media profiles of a logical media part.

Definition 14. *[connectors and MediaProfiles] Let* $w \in \Omega$*,* $ms \in \mathcal{MS}$*, and* $mp \in \mathcal{MP}$*, then the operator* $\mathit{connectors} : \Omega \longrightarrow \mathcal{P}(\mathcal{MS} \times \mathcal{MP})$ *is defined as* $\mathit{connectors}(w) = C_w$*, the operator* $\mathit{MediaProfile} : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MP}$ *as* $\mathit{MediaProfile}(ms, mp) = mp$*, and* $\mathit{MediaProfiles} : \Omega \longrightarrow \mathcal{P}(\mathcal{MP})$ *as* $\mathit{MediaProfiles}(w) = \mathit{Elements}(\mathit{Apply}(\mathit{MediaProfile}, \mathit{connectors}(w)))$.

Semantic Aspect. By attaching concepts of an ontology to entities, entities get meaning. The operator *types* accesses an entity's set of classifying ontology objects. For example, applying the operator *types* to the logical media part "Nosferatu" yields the set containing the ontology object "Movie" (see Fig. 1):

$$types(l_{nosferatu}) = \{o_{movie}\}.$$

Definition 15. *[types] Let $w \in \Omega$, then the operator types : $\Omega \longrightarrow \mathcal{P}(\Theta)$ is defined as $types(w) = T_w$.*

For retrieving the attributes of an entity, the operator *attributes* can be used. Requesting, for example, all attribute-value pairs of the logical media part "Nosferatu", i.e.

$$attributes(l_{nosferatu}) = \{(o_{director}, \text{“}Murnau\text{”})\},$$

yields the set including only one attribute-value pair, i.e. the ontology object "Director" with the string value "Murnau". The operator *attributes* returns the set of associated attribute-value pairs for a given entity.

Definition 16. *[attributes] Let $w \in \Omega$, then the operator $attributes : \Omega \longrightarrow \mathcal{P}(\Theta \times \mathbb{VAL})$ is defined as $attributes(w) = A_w$.*

EMMOs encapsulate a graph-like knowledge structure of entities. The algebra provides the operator *asso* for accessing all associations representing binary directed semantic relationships between other entities, e.g. the query expression

$$asso(e_{research}) = \{a_{mo \rightarrow moV3}, a_{moV3 \rightarrow st}, \\ a_{mi \rightarrow (mo \rightarrow moV3)}, a_{mi \rightarrow (moV3 \rightarrow st)}\}$$

returns the associations within EMMO "Dracula Research" (Fig. 5).

Definition 17. *[asso] Let $w \in \Omega$, then the operator asso : $\Omega \longrightarrow \mathcal{P}(\Lambda)$ is defined as $asso(w) = \{x \mid x \in N_w \cap \Lambda\}$.*

As associations are modeled as entities, they belong to an EMMO's set of nodes. The algebra provides the operator *nodes* for accessing all entities contained within an EMMO, e.g. the query expression

$$nodes(e_{research}) = \{e_{movies}, e_{moviesV3}, e_{studies}, o_{miller}, a_{mo \rightarrow moV3}, \\ a_{moV3 \rightarrow st}, a_{mi \rightarrow (mo \rightarrow moV3)}, a_{mi \rightarrow (moV3 \rightarrow st)}\}$$

yields a set consisting of all entities in EMMO "Dracula Research".

Definition 18. *[nodes] Let $w \in \Omega$, then the operator nodes : $\Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $nodes(w) = N_w$.*

As EMMOs are also entities, EMMOs can be nested hierarchically. The operator *AllEncEnt* can be used for accessing *all enc*apsulated *ent*ities of an EMMO,

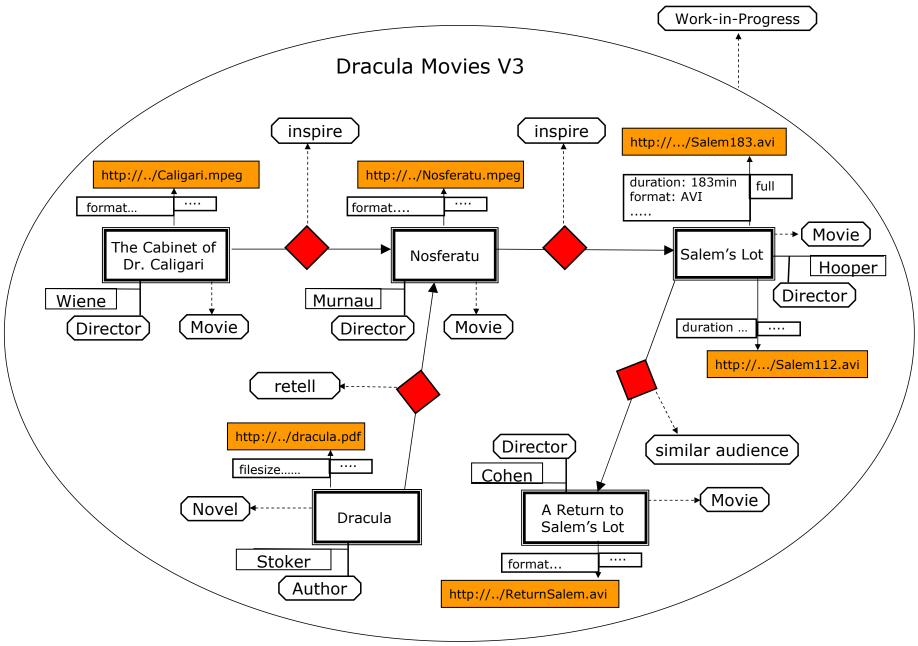


Fig. 6. EMMO "Dracula Movies V3"$(e_{moviesV3})$

i.e. it computes all entities recursively contained within an EMMO. For example, the query expression

$$\begin{aligned} AllEncEnt(e_{research}) &= nodes(e_{research}) \cup nodes(e_{movies}) \cup nodes(e_{moviesV3}) \\ &\quad \cup nodes(e_{studies}) = \\ &= \{e_{movies}, e_{moviesV3}, e_{studies}, o_{miller}, a_{mo\to moV3}, a_{moV3\to st}, \\ &\quad a_{mi\to(mo\to moV3)}, a_{mi\to(moV3\to st)}, l_{caligari}, l_{nosferatu}, \\ &\quad l_{salem}, a_{ca\to no}, a_{no\to sa}, l_{dracula}, l_{return}, \\ &\quad a_{dr\to no}, a_{sa\to re}, l_{vampire}, a_{va\to dr}\} \end{aligned}$$

unifies the nodes of EMMO "Dracula Research" with the nodes of the EMMOs "Dracula Movies" (Fig. 1), "Dracula Movies V3" (see Fig. 6), and "Dracula Studies" (Fig. 4), because these EMMOs are contained within EMMO "Dracula Research" and contain no further EMMOs themselves.

The operator *AllEncEnt* is defined by means of induction over the natural numbers *IN* and is based on the operator *EncEnt*. We say

- "entity w_1 is *contained* in EMMO w_0 *at first level*", if w_1 belongs to EMMO w_0's nodes,
- "entity w_{n+1} is *contained* in EMMO w_0 at $n{+}1^{th}$*-level*", if there exists a sequence of n EMMOs, i.e. $w_1, \ldots, w_n$, such that for all $k \in \{1, \ldots, n+1\}$ entity w_k belongs to EMMO w_{k-1}'s nodes,

– "w is *recursively contained* or *encapsulated* in EMMO w_0", if there exists a natural number n, such that w is contained in EMMO w_0 at n^{th}-level.

To provide a basis for the combination with other EMMA operators, the operators *AllEncEnt* and *EncEnt* both take entities as input value, but only return a reasonable result, if the input entity is of kind EMMO. In all other cases, the empty set is returned. In this way, the operator *EncEnt* takes an EMMO e and a natural number n as input, and returns the nodes of EMMO e at n^{th} level. By defining a unification over the operator *EncEnt*, the operator *AllEncEnt* returns, for a specified EMMO, the set of all its recursively contained entities.

Definition 19. *[AllEncEnt] Let $e \in \Omega$, then the operator $EncEnt : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ is defined inductively over IN as follows: $EncEnt(e, 1) = N_e$, and by assuming $EncEnt(e, n)$ is defined, one defines $EncEnt(e, n{+}1) = \{x \in \Omega \mid \exists y \in EncEnt(e, n) \cap \Sigma \wedge x \in N_y\}$. The operator $AllEncEnt : \Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $AllEncEnt(e) = \bigcup_{i \geq 1} EncEnt(e, i)$.*

Functional Aspect. EMMOs offer functions for dealing with their content. For the execution of an EMMO's functionality, the query algebra EMMA specifies the operator *Execute*. Applying the operator *Execute* to EMMO "Dracula Movies" (Fig. 1), the ontology object "rendering", and the parameter `HTML`, i.e.

$$Execute(e_{movies}, o_{render}, \texttt{HTML}) = f_{render}(e_{movies}, \texttt{HTML}) = \texttt{DraculaMovies.html},$$

returns an HTML-document representing the content of EMMO "Dracula Movies", for example, an HTML-document of a table with the rows being the EMMO's associations as illustrated in the left part of Fig. 7.

Applying the operator *Execute* to the same EMMO and the same ontology object, but with the parameter `SMIL`, i.e.

$$Execute(e_{movies}, o_{render}, \texttt{SMIL}) = f_{render}(e_{movies}, \texttt{SMIL}) = \texttt{DraculaMovies.smil},$$

yields a SMIL-document about the EMMO's content, for example, a SMIL-document sequentially representing the EMMO's associations as illustrated in the right part of Fig. 7.

The operator *Execute* takes an EMMO, a function, and a sequence of parameters as input values and returns the result value of the execution of the function with the specified EMMO and sequence of parameters as input values. If the operator *Execute* is applied to an entity which is not of kind EMMO, or the specified operation does not belong to the operations of the specified EMMO, or the sequence of parameters constitutes no valid input value for the specified operation, the empty set is returned.

Definition 20. *[Execute] Let $e \in \Omega$, $op \in \mathcal{OP}$, and $s \in \mathbb{SEQ}$, then the operator $Execute : \Omega \times \mathcal{OP} \times \mathbb{SEQ} \longrightarrow \mathbb{SET}$ is defined as*

$$Execute(e, op, s) = \begin{cases} \pi_2(op)(e, s) & if \quad op \in O_e \wedge (e, s) \in D(\pi_2(op)) \\ \emptyset & else \end{cases}$$

```
<html>
<body>
<h1>EMMO Dracula Movies</h1>
<table border="1">
<tr><th>Source</th>
<th>Association</th>
<th>Target</th></tr>
<tr><td>
<a href=".../Caligari.mpeg">The ..Caligari</a></td>
<td>Inspire</td>
<td><a href=".../Noseferatu.mpeg">Nosferatu</a></td></tr>
<tr><td><a href=".../Noseferatu.mpeg">Nosferatu</a></td>
<td>Inspire</td>
<td><a href=".../Salem183.avi">Salem's Lot</a><br>
<a href=".../Salem112.avi">Salem's Lot</a>
</td></tr>
</table>
</body>
</html>
```

```
<smil>
<head><layout>
<root-layout height="200" width="620"/>
<region id="l" left="0" ..../> .......
</layout></head>
<body> <seq>
<par end="60s" >
<video src="./Caligari.mpeg" type="video/mpeg" region="l"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Nosfertatu.mpeg" type="video/mpeg" region="r"/>
</par>
<par end="60s" >
<video src="./Nosferatu.mpeg" type="video/mpeg" region="l"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Salem183.avi" type="video/mpeg" region="r"/>
</par>
</seq></body>
</smil>
```

Fig. 7. DraculaMovies.html and DraculaMovies.smil

Versioning. Each entity describes a set of succeeding and a set of preceding versions. The operator *successors* can be used for accessing all direct successors of an entity, e.g. the query expression

$$successors(e_{movies}) = \{e_{moviesV1}, e_{moviesV2}\}$$

returns EMMO "Dracula Movies V1" and "Dracula Movies V2", the two direct successor versions of EMMO "Dracula Movies" (see Fig. 3). For accessing all succeeding versions, the operator *AllSuccessors* is applied, e.g.

$$AllSuccessors(e_{movies}) = \{e_{moviesV1}, e_{moviesV2}, e_{moviesV3}\}.$$

The operator *successors* retrieves all direct successors of the specified entity.

Definition 21. *[successors] Let $w \in \Omega$, then the operator $successors : \Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $successors(w) = S_w$.*

The operator *AllSuccessors* is defined by means of induction over the natural numbers *IN* and returns the set of all successors for a specified entity. The operator's definition is based on the operator *successors* serving as initial step of the induction and on the operator *Successors* which returns the set of all n^{th}-successors for a specified entity and natural number n. An entity w' is called n^{th} successor of entity w, if there exists a sequence of *n-1* entities, with each entity in the sequence representing the direct successor of its subsequent entity.

Definition 22. *[AllSuccessors] Let $w \in \Omega$, then the operator $Successors : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ is defined by induction over IN as follows: $Successors(w, 1) = successors(w)$, and by assuming $Successors(w, n)$ is defined, one defines $Successors(w, n+1) = \{x \in \Omega \mid \exists y \in Successors(w, n) \wedge x \in successors(y)\}$, and the operator $AllSuccessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ as $AllSuccessors(w) = \bigcup_{i \geq 1} Successors(w, i)$.*

For the access to an entity's preceding versions, EMMA also provides the operators *predecessors*, *Predecessors*, and *AllPredecessors*, which are defined analogously.

5.3 Navigational Operators

An EMMO establishes a graph-like knowledge structure of entities with associations being labeled by ontology objects describing the edges in the graph structure. The navigational operators provide means for traversing the semantic graph structure of an EMMO. Navigation through an EMMO's graph structure is controlled by a navigation path defined as a set of sequences of ontology objects. A mapping for each ontology object in the sequence to the corresponding association of an EMMO defines the traversal path of the graph structure. We have defined *regular path expressions* over ontology objects for describing the syntax of a navigation path. The basic building blocks of regular path expressions are ontology objects which can be modified and combined using conventional regular expression operators.

Definition 23. *[Regular path expression] Given a symbol set* $S = \{\varepsilon, \_, +, *, ?, |, -, (,)\}$*, an alphabet* $\Psi = \Theta \cup S$*, and* Ψ^**, the set of words over* Ψ *(finite strings over elements of* Ψ*). Then, we define* $\mathbb{REG} \subseteq \Psi^*$ *as the smallest set with the following properties:*

$$\begin{array}{llll}
(1) & \forall o \in \Theta : o \in \mathbb{REG}\,, & (6) & \forall r \in \mathbb{REG} : \quad r? \in \mathbb{REG}\,,\\
(2) & \varepsilon \in \mathbb{REG}\,, & (7) & \forall r \in \mathbb{REG} : \quad r+ \in \mathbb{REG}\,,\\
(3) & \_ \in \mathbb{REG}\,, & (8) & \forall r \in \mathbb{REG} : \quad r* \in \mathbb{REG}\,,\\
(4) & \forall r_1, r_2 \in \mathbb{REG} : \quad r_1 \,|\, r_2 \in \mathbb{REG}\,, & (9) & \forall o \in \Theta : \quad o- \in \mathbb{REG}\,,\\
(5) & \forall r_1, r_2 \in \mathbb{REG} : \quad r_1 r_2 \in \mathbb{REG}\,, & (10) & \forall r \in \mathbb{REG} : \quad (r) = r,
\end{array}$$

and denote $\mathbb{REG}$ *as the set of regular path expressions over ontology objects.*

Navigational operators take a regular path expression as input and specify how this syntactic expression is applied to navigate the graph structure. For example, for a given EMMO, starting entity, and regular path expression, the navigational operator *JumpRight* returns the set of all entities that can be reached by traversing the navigation path in the right direction, i.e. by following associations from source to target entities. Applying the operator *JumpRight* to EMMO "Dracula Movies V3"(see Fig. 6), the starting entity "The Cabinet of Dr. Caligari", and the regular path expression consisting of only one single ontology object "$o_{inspire}$" yields the logical media part representing the movie "Nosferatu":

$$JumpRight(e_{movies\,V3}, l_{caligari}, o_{inspire}) = \{l_{nosferatu}\}.$$

As already mentioned, the basic building blocks of regular path expressions are ontology objects, which can be modified and combined using conventional regular expression operators. For example, adding the operator "$*$" to a regular path expression specifies an iteration of path expressions, which is interpreted as navigation along the same regular path expression any number of times. Applying the operator *JumpRight* to the same EMMO and starting entity as in the above query, as well as the regular path expression "$o_{inspire}*$" returns three logical media parts representing the movies "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot":

$$JumpRight(e_{movies\,V3}, l_{caligari}, o_{inspire}*) = \{l_{caligari}, l_{nosferatu}, l_{salem}\}.$$

Regular path expressions can also be concatenated or defined as optional. For example, applying the operator *JumpRight* to EMMO "Dracula Movies V3", the starting entity "Nosferatu", and the regular path expression "$o_{inspire}o_{similar}?$", yields the logical media parts "Salem's Lot" and "A Return to Salem's Lot":

$$\mathit{JumpRight}(e_{movies\,V3}, l_{nosferatu}, o_{inspire}o_{similar}?) = \{l_{salem}, l_{return}\}.$$

The choice operator "|" can be used to combine regular path expressions as alternate versions, e.g.

$$\mathit{JumpRight}(e_{movies\,V3}, l_{nosferatu}, o_{inspire} \,|\, o_{retell}) = \{l_{salem}\}.$$

By adding the operator "$-$" to a regular path expression, the inversion of the regular path expression, i.e. the change of direction of navigation, can be expressed, e.g.

$$\mathit{JumpRight}(e_{movies\,V3}, l_{nosferatu}, o_{retell}-) = \{l_{dracula}\}.$$

Traversal along the opposite direction of associations can also be expressed with the navigational operator *JumpLeft*, e.g.

$$\mathit{JumpLeft}(e_{movies\,V3}, l_{nosferatu}, o_{retell}) = \mathit{JumpRight}(e_{movies\,V3}, l_{nosferatu}, o_{retell}-).$$

Navigational operators provide the basis for the integration of ontological knowledge into queries. For example, the transitivity of association types, such as the transitivity of associations of type "inspire", can be reflected by replacing the navigation path $o_{inspire}$ with the navigation path $o_{inspire}*$ (see example above). Knowledge about inverse association types, such as the association types "retell" and "is-retold", can be integrated within the queries as well, for instance, by replacing the navigation path $o_{is-retold}$ with the navigation path $o_{is-retold} \,|\, o_{retell}-$, e.g.

$$\mathit{JumpRight}(e_{movies\,V3}, l_{nosferatu}, o_{is-retold} \,|\, o_{retell}-) = \{l_{dracula}\}.$$

The operator *JumpRight*, which is formally defined below, takes two entities and one regular path expression as input values. The first input entity – which has to be of type EMMO for the operator *JumpRight* to return a non-empty result value – determines the navigation space, the second entity specifies the starting point of navigation, and the regular path expression describes the set of all possible navigation paths.

Definition 24. *[JumpRight] For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operator JumpRight : $\Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

$$
\begin{array}{lll}
(1) & \forall r \in \Theta: & JumpRight(e,w,r) = \{x \in N_e \mid \exists y\, y \in asso(e) \wedge \\
& & \qquad \wedge\, r \in types(y) \wedge\ w = s_y \wedge x = t_y\} \\
(2) & r = \varepsilon: & JumpRight(e,w,\varepsilon) = \{w \mid w \in N_e\} \\
(3) & r = \_: & JumpRight(e,w,\_) = \{x \in N_e \mid \exists y \in asso(e) \wedge \\
& & \qquad \wedge\, w = s_y \wedge x = t_y\} \\
(4) & \forall r_1, r_2 \in \mathbb{REG}: & JumpRight(e,w,r_1 \mid r_2) = \bigcup_{x \in \{r_1,r_2\}} JumpRight(e,w,x) \\
(5) & \forall r_1, r_2 \in \mathbb{REG}: & JumpRight(e,w,r_1 r_2) = \\
& & = \bigcup_{x \in JumpRight(e,w,r_1)} JumpRight(e,x,r_2) \\
(6) & \forall r \in \mathbb{REG}: & JumpRight(e,w,r?) = \bigcup_{x \in \{r,\varepsilon\}} JumpRight(e,w,x) \\
(7) & \forall r \in \mathbb{REG}: & JumpRight(e,w,r+) = \bigcup_{n \geq 1} JR_n(e,w,r) \quad with \\
& & JR_n(e,w,r)\, defined\ by\ induction\ over\, IN: \\
& & \quad JR_1(e,w,r) = JumpRight(e,w,r) \\
& & \quad JR_n(e,w,r) = \bigcup_{x \in JR_{n-1}(e,w,r)} JumpRight(e,x,r) \\
(8) & \forall r \in \mathbb{REG}: & JumpRight(e,w,r*) = \bigcup_{x \in \{r+,\varepsilon\}} JumpRight(e,w,x) \\
(9) & \forall o \in \Theta: & JumpRight(e,w,o\text{–}) = \{x \in N_e \mid \exists y\, y \in asso(e) \wedge \\
& & \qquad \wedge\, o \in types(y) \wedge\ x = s_y \wedge w = t_y\}.
\end{array}
$$

The navigational operator *JumpLeft* is defined analogously.

5.4 Selection Predicates

The selection predicates allow the selection of only those entities that satisfy a specific condition. They basically use the result values of extraction operators to create Boolean operators. For instance, applying the operator *IsType* to the logical media part “Dracula” (Fig. 6) and the set containing one ontology object “Book” returns false:

$$IsType(l_{dracula}, \{o_{book}\}) = false.$$

By taking a set of ontology objects as second input parameter, the operator *IsType* enables the integration of supertype/subtype relationships within queries. The ontological knowledge about a subtype relationship, e.g. the subtype relationship between the ontology objects “Novel” and “Book”, can be reflected within the query expression, e.g.

$$IsType(l_{dracula}, \{o_{book}, o_{novel}\}) = true.$$

Assuming that ontological knowledge about supertype/subtype relationships is also represented within EMMOs (e.g. in an EMMO $e_{ontology}$), e.g. by means of associations of type “is_a”, the subtypes of “Book” in the previous query could also be calculated dynamically during query execution by using an appropriate *JumpLeft* expression:

$$IsType(l_{dracula}, JumpLeft(e_{ontology}, o_{book}, o_{is\_a}*)) = true.$$

Although we have not yet developed a language which governs the expression of such ontological knowledge within the EMMO model, the query algebra is sufficiently expressive to be ready for exploiting this knowledge once it becomes available.

Definition 25. *[IsType] Let $w \in \Omega$, and $O \subseteq \Omega$, then the operator* $IsType : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsType(w, O) = \begin{cases} true & if \;\; \exists o \in O \;\; o \in types(w) \\ false & else \end{cases}$$

The selection predicates can be combined with the generic *Select* operator, which takes a predicate and an arbitrary set as input values, and returns all elements of the set that satisfy the condition of the specified predicate. For instance, if we apply the *Select* operator to the selection predicate *IsType* with the set consisting of the ontology objects "Book"and "Novel" as fixed parameter value and to the set of all logical media parts contained within EMMO "Dracula Studies" (see Fig. 4), the result set consists of the logical media part representing Stoker's novel "Dracula":

$$Select(IsType_{[\$,\{o_{book},o_{novel}\}]}, lmp(e_{studies})) = \{l_{dracula}\}.$$

Definition 26. *[Select] Let $A \in \mathbb{SET}$ and $p \in \mathbb{PRE}$, then let the operator* $Select : \mathbb{PRE} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *be* $Select(p, A) = \{x \mid x \in A \cap \mathcal{D}(p) \wedge p(x)\}$.

Being based on the return values of extraction operators, the list of selection predicates has the same length as the list of extraction operators. Any information which can be accessed by the extraction operators is again used for the selection of entities. Thus, for example, selection predicates allow the selection of all logical media parts within EMMO "Dracula Movies"(see Fig. 1) associated with a media profile describing media data in AVI format, i.e.

$$Select(HasMediaProfileValue_{[\$,\text{ “format”},Equal_{[\$,\text{ “AVI”}]}]}, lmp(e_{movies})) = \{l_{salem}\}$$

yields the logical media part "Salem's Lot" specified by two media profiles which both contain the attribute "format" with value "AVI" in their sets of metadata.

The operator $HasMediaProfileValue$ takes three input parameters, i.e. an entity w, a string value s, and a predicate p, and returns true, if the entity w contains a media profile with a set of metadata including a name-value pair, with the name being equal to s and the value satisfying the condition described by the specified predicate p, e.g. in the above example the predicate *Equal* returns true if its two specified parameters are equal, otherwise false.

Definition 27. *[HasMediaProfileValue] Let $w \in \Omega$, $s \in \mathbb{STR}$, and $p \in \mathbb{PRE}$, then* $HasMediaProfileValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*

$$HasMediaProfileValue(w, s, p) = \begin{cases} true & if \;\; \exists c \in C_w \\ & \quad \exists k \in Metadata(MediaProfile(c)) \\ & \quad (\pi_1(k) = s \wedge p(\pi_2(k))) \\ false & else \end{cases}$$

5.5 Constructors

EMMA specifies five constructors for EMMOs, i.e. the operators *Difference*, *Union*, *Intersection*, *Nest*, and *Flatten*. All the constructors take at least one

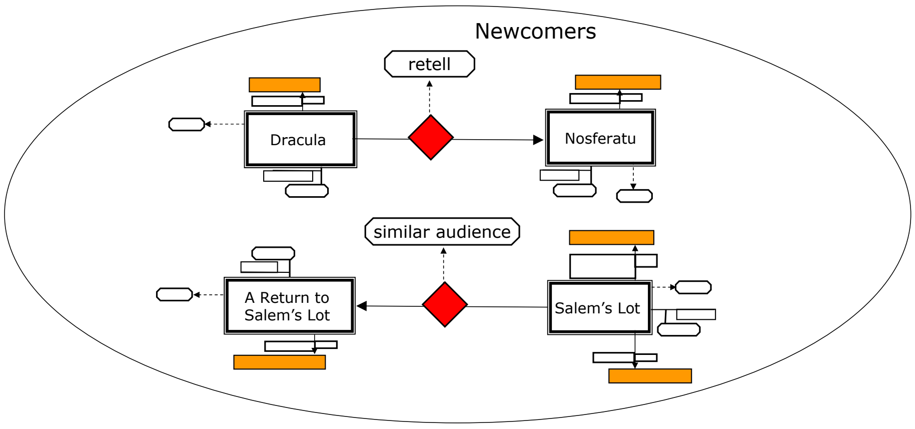


Fig. 8. EMMO "Newcomers" ($e_{newcomers}$)

EMMO and possibly other parameters as input values, and return exactly one EMMO as output value. The *Difference* operator takes two EMMOs and a string value. It creates a new EMMO which is denoted by the specified string value. The new EMMO's nodes encompass all entities belonging to the first, but not to the second EMMO, and additionally, the source and target entities of each association contained within the first EMMO.

Applying the *Difference* operator to the successor EMMO "Dracula Movies V3" (Fig. 6) and the original EMMO "Dracula Movies" (Fig. 1), generates a new EMMO "Newcomers" (see Fig. 8) consisting of the logical media parts describing the movies "Nosferatu", "Salem's Lot", and "A Return to Salem's Lot", and the novel "Dracula", as well as two connecting associations, i.e.

$$
\begin{aligned}
&\mathit{Difference}(e_{moviesV3}, e_{movies}, \text{“}\mathit{Newcomers}\text{”}) = e_{newcomers} \\
&\text{with} \qquad \mathit{nodes}(e_{newcomers}) = \{l_{dracula}, a_{dr\to no}, l_{nosferatu}, l_{salem}, a_{sa\to re}, l_{return}\}.
\end{aligned}
$$

Definition 28. *[Difference] Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$ *then the operator* $\text{Difference} : \Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as*
$\text{Difference}(e_1, e_2, s) = (o_{e_s}, \text{“}s\text{”}, \text{“}emm\text{”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} = \text{nodes}(e_1) \backslash \text{nodes}(e_2) \cup \{x \,|\, \exists y \in \text{asso}(e_1) \backslash \text{asso}(e_2) \quad x = t_y \vee x = s_y\}$.

The operators *Union* and *Intersection* are defined in a similar way, the operator *Nest* extracts the information stored within a set of associations from an EMMO, i.e. triples consisting of source entity, association, and target entity, into a new EMMO knowledge structure, and the operator *Flatten* generates a flattened EMMO, i.e. all recursively contained higher level entities are added as first level entities to the nodes of the EMMO. Due to space restriction, we omit the formal definitions.

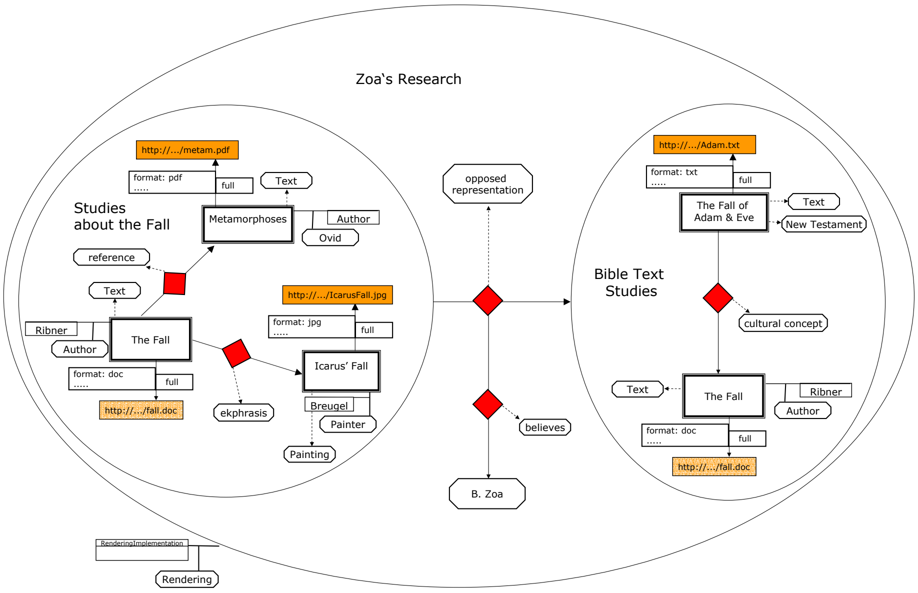


Fig. 9. EMMO "Zoa's Research"(e_{zoa})

5.6 Join Operator

The *Join* operator renders it possible to extend queries across multiple EMMOs. It specifies how to relate n sets of entities, possibly originating from different EMMOS, within a query. The *join* operator takes n entity sets, n operators, and one predicate as input values. We compute the Cartesian product of the n entity sets and select only those tuples that satisfy the predicate after applying the n operators to the n entities. The result set of tuples is projected onto the first entry. For example, asking for all entities within EMMO "Zoa's Research" (see Fig. 9) which contain within their nodes the logical media part "Icarus' Fall" corresponds to the query expression

$$Join(nodes(e_{zoa}), \{l_{icarus}\}, nodes, id, \supseteq) = \{e_{studiesfall}\}$$

and yields EMMO "Studies about the Fall", because this EMMO includes the logical media part "Icarus' Fall".

Definition 29. *[Join] Let* $i \in I = \{1, \ldots n\}, n \in I\!N, W_i \subseteq \Omega, f_i \in \mathbb{FUN}$ *and* $p \in \mathbb{PRE}$*, then the operator* $Join : \prod_{i\in I} \mathcal{P}(\Omega) \times \prod_{i\in I} \mathbb{FUN} \times \mathbb{PRE} \longrightarrow \mathbb{SET}$ *is defined as* $Join(W_1, \ldots, W_n, f_1, \ldots, f_n, p) = \{\pi_1(w_1, \ldots, w_n) \,|\, \forall i \in I$ $(w_i \in W_i \wedge f_i \in \mathbb{FUN}_{W_i} \wedge p \in \mathbb{PRE}_{\prod_{i\in I} \mathcal{R}(f_i)} \wedge p(f_1(w_1), \ldots, f_n(w_n)))\}$.

The *Join* operator is a generalization of the *Select* operator accounting for constraints defined on not only one but several entity sets. Defining the *identity* function *id*, i.e. $id(x) = x$, any select operation can be expressed by a join expression taking only one set, one operator, and one predicate p as input values, e.g.

$$Join(nodes(e_{studies}), id, p) = Select(p, nodes(e_{studies})).$$

5.7 Summary of EMMA Operators

EMMA provides operators to access the three aspects and the versioning information. The access to an EMMO's *media aspect* is realized by the operator *connectors* returning all connectors of a logical media part and the operator *MediaProfiles* returning all media profiles of a logical media part. For accessing the *semantic aspect*, EMMA provides the operator *types* accessing the types of an entity, the operator *attributes* returning an entity's attribute values, the operator *asso* retrieving all associations within an EMMO, the operator *nodes* yielding all entities within an EMMO, the operator *AllEncEnt* attaining all recursively contained entities within an EMMO, and the operators *JumpRight* and *JumpLeft* enabling the navigation of an EMMO's graph structure. The operator *Execute* addresses the *functional aspect*, and the operators *successors* (*predecessors*) and *AllSuccessors* (*AllPredecessors*) ensure the access to the versioning information.

The ability to arbitrarily nest and combine operators guarantees the high *orthogonality* of EMMA's operators. The basic *Select* operator takes a selection predicate and an arbitrary set – possibly the return set of another EMMA operation. The operator *Apply* allows one to use a specified operator not only for a single input value, but for a set of input values. As some of the operator's output values are represented in a format which cannot be directly used as input value for other operators, EMMA provides operators to transform and prepare the data for the use by other operators: the operator *Elements* allows the flattening of data sets and the *Nest* operator facilitates the nesting of an arbitrary set of associations into an EMMO knowledge container.

By extending queries across multiple EMMOs and entities, the *join* operator allows one to correlate the information contained in different EMMOs. The construction operators establish primitive operators for the construction and manipulation of EMMOs.

Finally, EMMA allows one to capture ontological knowledge within a query. Within the EMMO model, ontological knowledge is represented by ontology objects. The operator *types* accesses the classification of an entity (represented by a set of ontology objects), and the operator *IsType* selects entities of specific types. As the operators *JumpRight* and *JumpLeft* allow the specification of navigation along associations by means of powerful regular path expressions, they enable the inclusion of ontological knowledge such as transitive and inverse association types, and supertype/subtype relationships.

By fulfilling all the requirements described in Sect. 3, EMMA can be said to be *adequate* and *complete* with regard to the EMMO model.

6 Implementation

For enabling content sharing and collaborative authoring of EMMOs, the implementation had to be realized on a distributed infrastructure. Thus, we have established *EMMO containers* constituting a management component for EMMOs, i.e. the space where EMMOs "live". The EMMO containers are not intended as a centralized infrastructure realized by one single Root EMMO container running at one server. Instead we establish a decentralized infrastructure with EMMO containers of different scale and sizes running at different, distributed servers. To realize a decentralized EMMO management infrastructure two requirements need to be fulfilled:

- The users of EMMO containers are manifold, i.e. ranging from individual users running a home PC to multimedia content publishers. In other words, the systems running the EMMO containers are very heterogeneous servers with different sizes, operating systems, capabilities, and requirements. Therefore, the implementation of the EMMO container infrastructure needed to be *platform independent* and *scalable*. We have implemented the EMMO containers in *Java* and used the object-oriented DBMS *ObjectStore* for their persistent storage. By using Java we achieved platform independency and by using ObjectStore for the persistent storage of EMMOs the scalability of EMMO containers could be realized, i.e. besides a full-fledged database server implementation suitable for larger content providers, ObjectStore also provides a code-compatible file-based in-process variant *PSEPro* that better suits the limited capabilities and needs of home users.
- For enabling the sharing and collaborative authoring of multimedia content, EMMOs must be *transferable* between the different EMMO containers. Therefore, *export* and *import* facilities for EMMO containers, reflecting an EMMO's content, i.e. its three aspects and versioning information, are required. As EMMOs can describe quite complex structures, it is important for the users to specify the parts of the EMMO they want to export. They can choose between four *export options* indicating whether the EMMO is transferred with or without media objects, with or without versioning information, with or without encapsulated entities, and with or without the attached operations. EMMO containers export their EMMOs to a bundle structure, i.e. a ZIP archive that captures an EMMO's three aspects and versioning information, and indicates the specified export option.

We have implemented the *EMMA query processing architecture* with query optimization in mind, however, the realization of a query optimizer is subject of future work. The EMMA query processing architecture, which is depicted in Fig. 10, is based on the implementation of the EMMO container infrastructure described above. Its focus is the extraction and navigation of information stored within the EMMO containers. The EMMA query processing architecture takes syntactically well-defined query expressions as input. The processing of the query expressions reflects the definition of the EMMA query operators and produces a set of EMMO objects in a pre-defined output format.

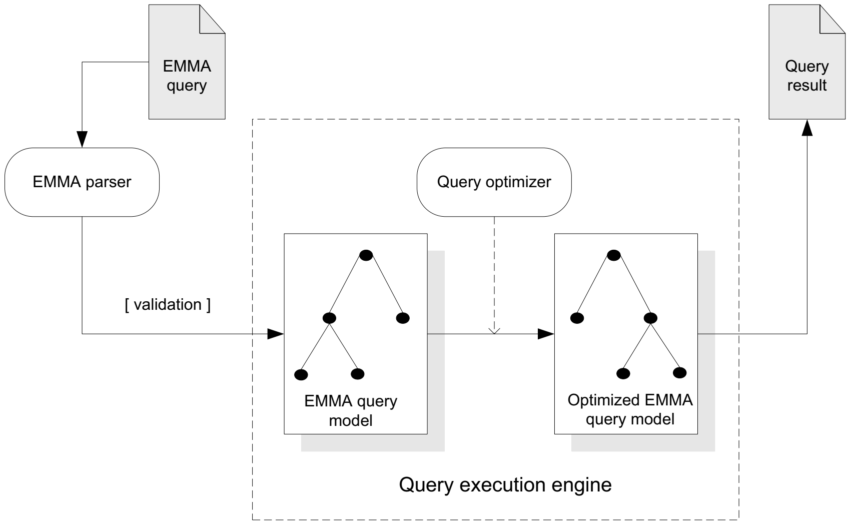


Fig. 10. The EMMA query processing architecture

For the implementation of the EMMA operators, we have chosen a functional approach, i.e. each operator has a corresponding function that evaluates according to its implementation-specific algorithm. For enabling consistency and integrity checking, each function has a signature that defines the number and types of input arguments, and, additionally, the types of the expected output values. By typing all EMMO and EMMA model constructs according to an internal hierarchy, those constructs can be used for specifying the signature of functions.

For realizing complex queries, i.e. the nesting of modular EMMA operators, the *EMMA query model* is built up. The EMMA query model is a tree consisting of nodes and leaves. Nodes represent algebraic operators, and leaves correspond to EMMO and EMMA model constructs, i.e. values of the underlying EMMO container. The EMMA query model is supplied with a built-in validation mechanism, ensuring that operators in the query tree contain only valid references to subsequent nodes, i.e. before evaluating the complex structure of the query model tree, a consistency and integrity check concerning the signature of the functions implementing the EMMA operators is performed.

By applying a *bottom-up evaluation* technique, the execution computes the final query result. This evaluation technique runs through several steps. First, any EMMO or EMMA model construct captured by the EMMO containers that represents a valid input value for the query expression is fetched. Then, all possible output values – represented as tuples – that can be derived when applying the function's algorithm reflecting the definition of the corresponding EMMA operator of the fetched input values, are computed. Going up the tree hierarchy, this process is repeated by applying functions to the set of objects in the EMMO store together with those tuples which were inferred in the previous step. This process is repeated until the root of the query tree is reached and the final result set is delivered.

Query optimization is realized by the EMMA query optimizer, which takes a query model and transforms it into an equivalent model that can evaluate more efficiently. The design of the transformation algorithm is based on the evaluation of the response time of query expressions and is subject of future work.

7 Conclusion

In this paper, we have introduced the formal basis of the query algebra EMMA, which enables the efficient retrieval of the knowledge represented by EMMOs, a novel approach to semantic multimedia meta modeling. EMMA's operators provide the access to all information and aspects stored within EMMOs and are based on precise semantics, thus offering a formal basis for query rewriting and optimization. EMMA features orthogonal, arbitrarily combinable operators that range from simple selection and extraction operators to more complex navigational operators, joins, and even rudimentary operators for the construction and manipulation of EMMOs. Moreover, EMMA renders it possible to integrate ontological knowledge within queries, such as supertype/subtype relationships, transitive or inverse association types. We have briefly sketched the implementation of the EMMO container infrastructure and the EMMA query processing architecture.

In our future work, we will focus on the realization of an eLearning scenario by means of the EMMO infrastructure. Based on real-world data gathered from this use case, we will carry out experiments for performance evaluation, in particular to achieve a detailed analysis and understanding of the effects of the various factors on query performance. We will use the application-specific data as starting point for the development of an EMMA query optimizer. Furthermore, we are in the process of developing an ontology engineering environment, consisting of an ontology description language compatible with the EMMO model, and tools that enable the seamless integration of ontological knowledge into query processing.

References

[1] Raggett, D., Hors, A.L., Jacobs, I.: HTML 4.01 Specification. W3C Recommendation, World Wide Web Consortium (W3C) (1999)

[2] Ayars, J., et al.: Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation, World Wide Web Consortium (W3C) (2001)

[3] Ferraiolo, J., Jun, F., Jackson, D.: Scalable Vector Graphics (SVG) 1.1. W3C Recommendation, World Wide Web Consortium (W3C) (2003)

[4] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)

[5] Cruz, I., Sayenko, O.: Semantically Driven Multimedia Querying and Presentation. In Srinivasan, U., Nepal, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (2005)

[6] Hunter, J.: Enhancing the Semantic Interoperability of Multimedia Through a Core Ontology. IEEE Transaction on Circuits and Systems for Video Technology **13** (2003)

[7] van Ossenbruggen, J., Nack, F., Hardman, L.: That Obscure Object of Desire: Multimedia Metadata on the Web (Part I). IEEE MultiMedia **11** (2004)

[8] Nack, F., van Ossenbruggen, J., Hardman, L.: That Obscure Object of Desire: Multimedia Metadata on the Web (Part II). IEEE MultiMedia **12** (2005)

[9] Nack, F., Hardman, L.: Towards a Syntax for Multimedia Semantics. CWI Report INS-RO204, Centrum voor Wiskunde en Informatica (2002)

[10] Hammiche, S., et al.: Semantic Retrieval of Multimedia Data. In: Proc. of the Second ACM International Workshop on Multimedia Databases, Washington, DC, USA (2004)

[11] Srinivasan, U., Nepall, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (2005)

[12] Schellner, K., Westermann, U., Zillner, S., Klas, W.: CULTOS: Towards a World-Wide Digital Collection of Exchangeable Units of Multimedia Content for Intertextual Studies. In: Proc. of the Conference on Distributed Multimedia Systems (DMS 2003), Miami, Florida (2003)

[13] Newman, D., Patterson, A., Schmitz, P.: XHTML+SMIL Profile. W3C Note, World Wide Web Consortium (W3C) (2002)

[14] ISO/IEC JTC 1/SC 34/WG 3: Information Technology – Hypermedia/Time-Based Structuring Language (HyTime). International Standard 15938-5:2001, ISO/IEC (1997)

[15] ISO/IEC IS 13522-5: Information Technology – Coding of Hypermedia Information – Part 5: Support for Base-Level Interactive Applications. International Standard, ISO/IEC (1996)

[16] Pereira, F., Ebrahimi, T., eds.: The MPEG-4 Book. Pearson Education, California (2002)

[17] Beckett, D.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium (W3C) (2004)

[18] Brickely, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium (W3C) (2004)

[19] ISO/IEC JTC 1/SC 34/WG 3: Information Technology – SGML Applications – Topic Maps. ISO/IEC International Standard 13250:2000, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2000)

[20] ISO/IEC JTC 1/SC 29/WG 11: Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes. Final Draft International Standard 15938-5:2001, ISO/IEC (2001)

[21] ISO/JTC 1/SC 32/WG 2: Conceptual Graphs. ISO/IEC International Standard, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2001)

[22] Westermann, U., Zillner, S., Schellner, K., Klas, W.: EMMOs: Tradeable Units of Knowledge Enriched Multimedia Content. In Srinivasan, U., Nepal, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (2005)

[23] Zillner, S., Westermann, U., Winiwarter, W.: EMMA – Towards a Query Algebra for Enhanced Multimedia Meta Objects. In: Proc. of the Fourth International Conference on Computer and Information Technology (CIT 2004), Wuhan, China (2004)
[24] Zillner, S., Westermann, U., Winiwarter, W.: EMMA – A Query Algebra for Enhanced Multimedia Meta Objects. In: Proc. of the Third International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2004), Larnaca, Cyprus (2004)
[25] Billiani, F., et al.: Demonstrator of Intertextual Cultural Threads – Standard Ontology-Extended Ontology. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning (2003)
[26] Benari, M., et al.: Proposal for a Standard Ontology of Intertextuality. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning (2003)
[27] Leach, P.: UUIDs and GUIDs. Network Working Group Internet-Draft, The Internet Engineering Task Force (IETF) (1998)
[28] Zillner, S., Winiwarter, W.: Ontology-Based Query Refinement for Multimedia Meta Objects. In: Proc. of the Sixth International Conference on Information Integration and Web Based Applications & Services (iiWAS 2004), Jakarta, Indonesia (2004)
[29] Zillner, S., Winiwarter, W.: Integrating Ontology Knowledge into a Query Algebra for Multimedia Meta Objects. In: Proc. of the Fifth International Conference on Web Information Systems Engineering (WISE 2004), Brisbane, Australia (2004)
[30] Zillner, S., Winiwarter, W.: Integration of Ontological Knowledge within the Authoring and Retrieval of Multimedia Meta Objects. International Journal of Web and Grid Services (IJWGS) **1** (2005)
[31] Cattell, R., ed.: The Object Database Standard: ODMG-93. Morgan, Kaufmann, San Francisco, CA (1994)
[32] Leung, T., et al.: The Aqua Data Model and Algebra. In: Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages, Manhattan, New York City (1993)
[33] Kifer, M., Kim, W., Sagiv, Y.: Querying Object-Oriented Databases. In: Proc. of the ACM SIGMOD Conference on Management of Data, San Diego, CA (1992)
[34] Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object Exchange Across Heterogeneous Information Sources. In: Proc. of the Eleventh International Conference on Data Engineering, Taipei (1995)
[35] Abiteboul, S., et al.: The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries **1** (1997) 68–88
[36] Bruneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. The VLDB Journal – The International Journal on Very Large Data Bases **9** (2000)
[37] Beeri, C., Tzaban, Y.: SAL: An Algebra for Semistructured Data and XML. In: Proc. of the Second International Workshop on the Web and Databases (WebDB 99), Philadelphia, Pennsylvania, USA (1999)
[38] Boag, S., et al.: XQuery 1.0: An XML Query Language. W3C Working Draft, World Wide Web Consortium (W3C) (2005)
[39] Berglund, A., et al.: XML Path Language (XPath). W3C Working Draft Version 2.0, World Wide Web Consortium (W3C) (2005)
[40] Lee, T., et al.: Querying Multimedia Presentations Based on Content. IEEE Transactions on Knowledge and Data Engineering **11** (1999)

[41] Duda, A., Weiss, R., Gifford, D.: Content Based Access to Algebraic Video. In: Proc. of the IEEE First International Conference on Multimedia Computing and Systems, Boston, MA, USA (1994)
[42] Adali, S., Sapino, M., Subrahmanian, V.: A Multimedia Presentation Algebra. In: Proc. of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA (1999)
[43] Karvounarakis, G., et al.: RQL: A Functional Query Language for RDF. In Gray, P.M.D., et al., eds.: The Functional Approach to Data Management. Springer, Heidelberg, Germany (2003)
[44] Miller, L., Seaborn, A., Reggiori, A.: Three Implementations of SquishQL, a Simple RDF Query Language. In: Proc. of the First International Semantic Web Conference (ISWC2002), Sardinia, Italy (2002)
[45] Frasincar, F., et al.: RAL: An Algebra for Querying RDF. In: Proc. of the Third International Conference on Web Information Systems Engineering (WISE 2000), Singapore (2002)
[46] ISO/IEC JTC1 SC34 WG3: New Work Item Proposal, Topic Map Query Language (TMQL). New Proposal, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2000)
[47] Garshol, L.: Tolog 0.1. Ontopia Technical Report, Ontopia (2003)
[48] Bogachev, D.: TMPath – Revisited. Online Article, available under http://homepage.mac.com/dmitryv/TopicMaps/TMPath/TMPath Revisited.html (2004)
[49] Barta, R., Gylta, J.: XTM::Path – Topic Map Management, XPath Like Retrieval and Construction Facility. Online Article, available under http://cpan.uwinnipeg.ca/htdocs/XTM/XTM/Path.html (2002)
[50] Widhalm, R., Mück, T.: Topic Maps (in German). Springer, Berlin Heidelberg, Germany (2002)
[51] Manjunath, B., Salembier, P., Sikora, T., eds.: Introduction to MPEG-7. John Wiley & Sons, West Sussex, UK (2002)
[52] Zillner, S.: A Query Algebra for Ontology-enhanced Management of Multimedia Meta Objects. PhD thesis, Vienna University of Technology (2005)

Comparing and Transforming Between Data Models Via an Intermediate Hypergraph Data Model

Michael Boyd[1] and Peter McBrien[2]

[1] PSA Parts Ltd, London SW19 3UA
mb@psaparts.co.uk
[2] Dept. of Computing, Imperial College, London SW7 2AZ
pjm@doc.ic.ac.uk

Abstract. Data integration is frequently performed between heterogeneous data sources, requiring that not only a schema, but also the data modelling language in which that schema is represented must be transformed between one data source and another.

This paper describes an extension to the hypergraph data model (HDM), used in the AutoMed data integration approach, that allows constraint constructs found in static data modelling languages to be represented by a small set of primitive constraint operators in the HDM. In addition, a set of five equivalence preserving transformation rules are defined that operate over this extended HDM. These transformation rules are shown to allow a bidirectional mapping to be defined between equivalent relational, ER, UML and ORM schemas.

The approach we propose provides a precise framework in which to compare data modelling languages, and precisely identifies what semantics of a particular domain one data model may express that another data model may not express. The approach also forms the platform for further work in automating the process of transforming between different data modelling languages. The use of the both-as-view approach to data integration means that a bidirectional association is produced between schemas in the data modelling language. Hence a further advantage of the approach is that composition of data mappings may be performed such that mapping two schemas to one common schema will produce a bidirectional mapping between the original two data sources.

Keywords: conceptual data modelling, mappings, transformations, multiple representations.

1 Introduction

The AutoMed data integration system [8,24] distinguishes itself as being an approach which has a clear methodology for handling a wide range of static data modelling languages in the integration process [28], as opposed to the other approaches that assume integration is always performed in a single common data model. This is achieved by allowing a user to relate the modelling constructs of a higher level modelling language such as ER, relational, UML, or ORM, to the constructs in a single lower level common data modelling language called the **hypergraph data model** (**HDM**) [39], *i.e.* using the terminology of model management [5] we perform a ModelGen to convert schemas in

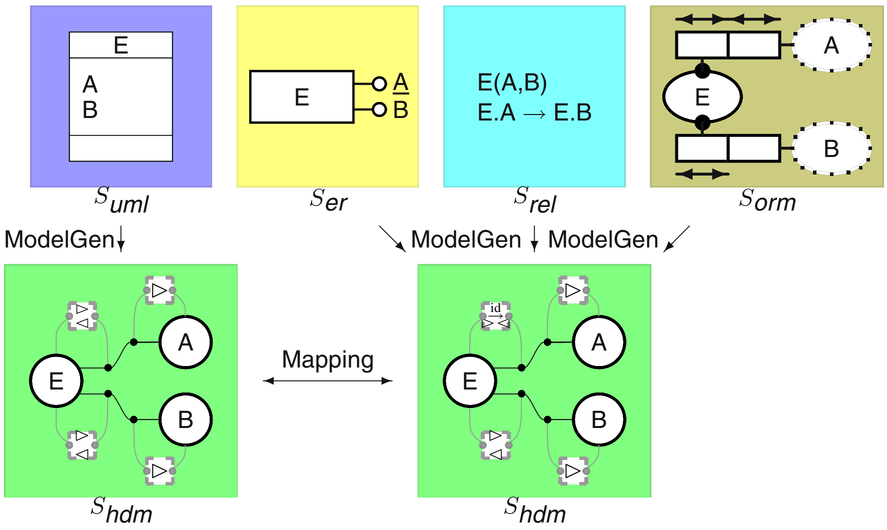


Fig. 1. Conceptual modelling languages represented in the HDM

the higher level modelling languages into the HDM. Figure 1 illustrates this concept being applied to four schemas which might appear to be equivalent.

In [28] a general approach was proposed showing how the data aspects of higher level modelling languages were modelled as nodes and edges in the HDM, with the constraints of the higher level modelling language being represented by writing constraint formulae over the HDM. For example, the ER schema in Figure 1 has an entity labelled E that we represent by a node E in the HDM schema (represented by a black outlined circle), and the attributes A and B of E are also represented as nodes in the HDM, together with edges (the thick black lines) associating them to the node representing E. The fact that each entity instance has only one associated attribute instance in each of A and B is represented by constraint rules in the HDM (the dash grey boxes, introduced in [9] and extended in this paper), as is the fact that A is a key attribute of E. Using the rules for ModelGen presented in Section 2, we will see that the ER, relational and ORM schemas in Figure 1 all produce the same HDM schema, and the UML schema produces an HDM schema with one difference in the constraints. In Section 3 we will describe equivalence preserving rules for the HDM that can be used to map between equivalent HDM schemas: this example is a case where the two HDM schemas shown are not equivalent, and hence the rules do not permit us to 'lose' the extra $\xrightarrow{\text{id}}$ constraint.

The concept of using graphs as an underlying representation for higher level modelling languages has been used in modelling relational schemas [51], and for OO and ER schemas [49], and we argue is an intuitive assumption to make. It also reduces all schemas to an irreducible form [19], and in the context of relational databases has recently been identified as a sixth normal form [13,12].

This paper extends the approach of [28] to represent the high level modelling language constraints using a set of primitive constraint operators on the HDM. This paper also shows how we may relate the ER, relational, UML and ORM higher level modelling languages — perform **intermodel transformations** — by the application of five types of equivalence rules on the HDM and its primitive constraint operators. This work is a considerable enhancement of our earlier work presented in [9] in that:

1. We give a formal definition of the constraint operators in an extension of the HDM definition found in [39], and have added an additional constraint operator.
2. We give a set of mapping rules to exactly define how the higher level modelling languages are converted into these new constraint operators, and in addition we now review how ORM is modelled using the constraints. We also consider more advanced modelling language features, such as generalisation hierarchies, look-here and look-across cardinality constraints, candidate keys, and n-ary relationships.
3. We define how **both-as-view** (**BAV**) [30] data integration rules can be generated, and use the properties of these BAV rules to demonstrate when we have equivalent higher level schemas, and when there is information loss in the mapping process.

Our approach differs from other work in the area (reviewed in Section 5) in that we use hypergraphs as the common data modelling language combined with producing BAV transformations that exactly define the nature of the relationship (equivalence or non-equivalence) between schemas on a construct by construct basis. Our approach forms a framework, where the constraints we propose in the HDM, and the equivalence preserving transformations, can be extended to handle new modelling languages. Thus we provide a platform for the conversion between any data modelling language, with the limitation that our current work assumes that the data model must have set-based semantics. We also leave for future work the consideration of types in the data models: we make the crude assumption for our current work that the data types match in the data models being compared. This paper demonstrates the approach being applied by converting between the major construct types of ER, relational, UML class and ORM modelling languages, and hence has wide applicability in data modelling.

In addition to providing a mechanism for comparing the expressiveness of modelling languages, the proposed primitive constraints and set of equivalence rules also forms the basis for a method of automating the translation between modelling languages, based on descriptions of their constructs. This would involve further development of an algorithm that would determine which equivalence rules need to be applied to the HDM schema of one higher level schema to form a valid HDM schema of another higher level schema.

The remainder of this paper is structured as follows. Section 2 reviews how to describe higher level data modelling languages by relating them to the HDM schema. The HDM language is extended with a set of constraint operators that form a language used to model the constraints in higher level data modelling languages. Section 3 details how we approach the transformation between schemas in different modelling languages by applying equivalence rules to the HDM schemas, thereby relating basic constructs of the higher level modelling languages with each other. Section 4 considers how some extended operators of these languages are related. Section 5 discusses some related work, and we give a summary and discuss future work in Section 6.

2 Describing a Data Modelling Language

We now review the HDM (first defined in [39]), giving slightly modified definitions that reflect the syntax used in later work on the HDM and the AutoMed implementation. Then we present an example **universe of discourse** (**UoD**), and use it to illustrate in general how the HDM may be used to represent ER, relational, UML and ORM schemas. To keep the initial discussion reasonably concise, we leave some advanced higher level modelling features until Section 4.

We define first the notion of a HDM **schema**, which is the structure in which data may be held.

Definition 1 HDM Schema
Given a set of $Names$ that we may use for modelling the real world, an HDM **schema**, S, is a triple $\langle Nodes, Edges, Cons\rangle$ where:

- $Nodes \subseteq \{\langle\langle n_n\rangle\rangle \mid n_n \in Names\}$
  i.e. $Nodes$ is a set of nodes in the graph, each denoted by its name enclosed in double chevron marks.
- $Schemes = Nodes \cup Edges$
- $Edges \subseteq \{\langle\langle n_e, s_1, \ldots, s_n\rangle\rangle \mid$
  $n_e \in Names \cup \{\_\} \wedge s_1 \in Schemes \wedge \ldots \wedge s_n \in Schemes\}$
  i.e. $Edges$ is a set of edges in the graph where each edge is denoted by its name, together with the list of nodes/edges that the edge connects, enclosed in double chevron marks.
- $Cons \subseteq \{c(s_1, \ldots, s_n) \mid c \in Funcs \wedge s_1 \in Schemes \wedge \ldots \wedge s_n \in Schemes\}$
  i.e. $Cons$ is a set of boolean-valued functions (*i.e.* constraints) whose variables are members of $Schemes$ and where the set of functions $Funcs$ forms the HDM constraint language. □

The first three items in Definition 1 define a labelled, directed, nested hypergraph (a **hyperedge** is an edge that connects more than two nodes in a graph, and these edges are **nested** in the sense that hyperedges can themselves participate in hyperedges). The last item in Definition 1 will be refined into the constraint language which is one of the contributions of this paper. We list in Example 1 the contents of an example HDM schema that we shall later, in Figure 3, show to be equivalent to an ER schema. Note how the names of edges are sometimes given as the character '_' representing an unnamed edge, and also note that one of the edges is a nested edge, connecting the node ⟨⟨result:grade⟩⟩ to another edge ⟨⟨result,student,course⟩⟩.

Example 1 An HDM schema
$Nodes$ = {⟨⟨ug⟩⟩, ⟨⟨ug:ppt⟩⟩, ⟨⟨student⟩⟩, ⟨⟨student:name⟩⟩, ⟨⟨student:sid⟩⟩,
⟨⟨result:grade⟩⟩, ⟨⟨course⟩⟩, ⟨⟨course:code⟩⟩, ⟨⟨course:dept⟩⟩}
$Edges$ = {⟨⟨_,ug,ug:ppt⟩⟩, ⟨⟨_,student,student:sid⟩⟩, ⟨⟨_,student,student:name⟩⟩,
⟨⟨result,student,course⟩⟩, ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩,
⟨⟨_,course,course:dept⟩⟩, ⟨⟨_,course,course:code⟩⟩} □

The fourth component of the HDM schema definition states any extra constraints that all instances of the schema must satisfy. Note that the definition of constraints in [39] makes no restrictions on what the constraint language is. This papers proposes a restricted constraint language, which we show is able to represent constraints in higher level modelling languages, and forms a basis for writing transformations between schemas in those higher level modelling languages. Before defining these constraints we give in Definition 2 a definition of an **instance** of a schema, simplified from the definition in [39].

Definition 2 HDM instance
Given an HDM schema S, an **instance** I of S is a structure for which there exists a function $\mathit{Ext}_{S,I} : Schemes \rightarrow P(Seq(Vals))$ where $Vals$ is the set of values we wish to model as being in the domain of our data model, Seq gives any sequence of those values, and P forms the power set of those sequences. We also have the restriction that:

1. each tuple $\langle a_1, \ldots, a_n \rangle \in \mathit{Ext}_{S,I}(\langle\langle n_e, s_1, \ldots, s_n \rangle\rangle)$, $a_i \in \mathit{Ext}_{S,I}(s_i)$ for all $1 \leq i \leq n$;
2. for every $c \in Cons$, the expression $c(v_1/\mathit{Ext}_{S,I}(v_1), \ldots, v_n/\mathit{Ext}_{S,I}(v_n))$ evaluates to true, where $v_1, \ldots, v_n$ are the variables of c.

We call $Ext_{S,I}(s)$ the **extent** of scheme $s \in Schemes$. □

Note that item (1) of Definition 2 enforces that the extent of an edge is drawn from values present in the extents of nodes and other edges it connects. Note that no restriction is put on the extent of nodes: this would form the basis of **typing** in the HDM, where a type $T \subseteq P(Seq(Vals))$ is associated to each $s \in Schemes$ such that for all I $\mathit{Ext}_{S,I}(s) \subseteq T$. Also note that the semantics of the schema are set based, and hence we at present can only use the HDM to accurately model data modelling languages with set based semantics. Dealing with typing of data, and bag or list based semantics will be the subject of our future work.

The combination of HDM schema and instance, together with an extension mapping function Ext forms what we will term an HDM data source in Definition 3.

Definition 3 HDM data source
A **data source** is a triple $\langle S, I, Ext_{S,I} \rangle$ where S is a schema, I is an instance of S and $Ext_{S,I}$ is an extension mapping from S to I. □

We now introduce to the HDM a set of six primitive constraints that we are proposing as a practical solution to model the constraints of the higher level modelling language in an analogous manner to how the nodes and edges of the HDM models the data aspects of higher level modelling language. The set of six constraint operators might need to be extended in the future to handle a wider range of high level modelling language constructs, but we will demonstrate in this paper that our six constraints can deal with a wide range of modelling concepts used in practice.

In the following descriptions, any variable beginning with s denotes a member of $Schemes$. For each definition we give both a functional form (*e.g.* $\mathbf{inclusion}(s_1, s_2)$, useful for some of our mapping rules that talk in general about a constraint $op(\ldots)$)

and an equivalent infix form (*e.g.* $s_1 \subseteq s_2$, which is used in the diagrams and in most of the descriptions). When using these definitions later, we assume that where a tuple of schemes $\langle s_1, \ldots, s_m \rangle$ is used in a constraint definition, then s_1 may be used as the singleton tuple $\langle s_1 \rangle$. For example, we can write $s_1 \lhd s$ as a shorthand for $\langle s_1 \rangle \lhd s$. Several of the constraint definitions use a **project** function defined in Definition 4 that provides a method of producing a kind of view of an HDM edge restricted to contain a subset of the nodes or edges that the edge connects.

Definition 4 HDM project
The HDM **project** function $\pi(\langle s_x, \ldots, s_y \rangle, s, t)$, which takes a tuple of schemes $\langle s_x, \ldots, s_y \rangle$ that must appear in edge s, together with a tuple t that appears in the extent of s, returns the values in tuple t that corresponds to the schemes $\langle s_x, \ldots, s_y \rangle$:

$$\pi(\langle s_x, \ldots, s_y \rangle, \langle\langle n_e, s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle\rangle, \\ \langle a_1, \ldots, a_x, \ldots, a_y, \ldots, a_n \rangle) = \langle a_x, \ldots, a_y \rangle$$

If the tuple t is omitted, a set of values is obtained:

$$Ext_{S,I}(\pi(\langle s_x, \ldots, s_y \rangle, \langle\langle n_e, s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle\rangle)) = \{\langle s_x, \ldots, s_y \rangle \mid \\ \langle s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle \in Ext_{S,I}(\langle\langle n_e, s_1, \ldots, s_x, \ldots, s_y, \ldots, s_n \rangle\rangle)\} \quad \square$$

Definition 5 HDM Constraints
The minimum HDM constraint language should comprise of $Funcs$={inclusion, exclusion,union,mandatory,unique,reflexive}, where the six functions have the following semantics:

1. $\mathbf{inclusion}(s_1, s_2) \equiv s_1 \subseteq s_2$
   States that the extent of s_1 is always a subset of the extent of s_2, *i.e.* for all I, $Ext_{S,I}(s_1) - Ext_{S,I}(s_2) = \emptyset$
2. $\mathbf{exclusion}(s_1, \ldots, s_n) \equiv s_1 \not\cap \ldots \not\cap s_n$
   for all $1 \leq x < y \leq n$, and for all I, $Ext_{S,I}(s_x) \cap Ext_{S,I}(s_y) = \emptyset$
3. $\mathbf{union}(s, s_1, \ldots, s_n) \equiv s = s_1 \cup \ldots \cup s_n$
   for all I, $Ext_{S,I}(s) = Ext_{S,I}(s_1) \cup \ldots \cup Ext_{S,I}(s_n)$
4. $\mathbf{mandatory}(\langle s_1, \ldots, s_m \rangle, s) \equiv \langle s_1, \ldots, s_m \rangle \rhd s$
   States that every combination of the values in extents of $s_1, \ldots, s_m$ must appear at least once in the extent of edge s that connects them, *i.e.* for all I
   $$\{\langle a_1, \ldots, a_m \rangle \mid a_1 \in Ext_{S,I}(s_1) \wedge \ldots \wedge a_m \in Ext_{S,I}(s_m)\} \\ - \{\langle \pi(s_1, s, t), \ldots, \pi(s_m, s, t) \rangle \mid t \in Ext_{S,I}(s)\} = \emptyset$$
5. $\mathbf{unique}(\langle s_1, \ldots, s_m \rangle, s) \equiv \langle s_1, \ldots, s_m \rangle \lhd s$
   States that every combination of the values in extents of $s_1, \ldots, s_m$ must appear at most once in the extent of edge s that connects them, *i.e.* for all I
   $$\{t \mid t \in Ext_{S,I}(s) \wedge t' \in Ext_{S,I}(s) \wedge t \neq t' \wedge \\ \pi(s_1, s, t) = \pi(s_1, s, t') \wedge \ldots \wedge \pi(s_m, s, t) = \pi(s_m, s, t')\} = \emptyset$$
6. $\mathbf{reflexive}(s_1, s) \equiv s_1 \xrightarrow{\text{id}} s$
   If an instance of scheme s_1 appears in edge s, then one of the instances of s must be an identity tuple, *i.e.* for all I:
   $$\{\pi(s_1, s, t) \mid t \in Ext_{S,I}(s)\} - \\ \{\pi(s_1, s, t) \mid t \in Ext_{S,I}(s) \wedge t = \langle \pi(s_1, s, t), \pi(s_1, s, t) \rangle\} = \emptyset \quad \square$$

Example 2 may be combined with Example 1 to give a complete HDM as represented diagrammatically in Figure 3(b). The constraint language might appear too fine grain, but we will show in Section 3 that when writing inter model transformations it is useful to be able to test for, to add or to delete constraints expressed at this level of detail. Also, the six constraint operators defined here are not meant to be definitive: handling other modeling language constructs in addition to those presented later in this section might require us to introduce additional constraint primitives.

Example 2 Constraints in an HDM schema

$Cons = \{\langle\langle \mathsf{ug} \rangle\rangle \lhd \langle\langle \mathsf{\_,ug,ug{:}ppt} \rangle\rangle, \langle\langle \mathsf{ug} \rangle\rangle \rhd \langle\langle \mathsf{\_,ug,ug{:}ppt} \rangle\rangle,$
$\langle\langle \mathsf{ug{:}ppt} \rangle\rangle \rhd \langle\langle \mathsf{\_,ug,ug{:}ppt} \rangle\rangle, \langle\langle \mathsf{ug} \rangle\rangle \subseteq \langle\langle \mathsf{student} \rangle\rangle,$
$\langle\langle \mathsf{student} \rangle\rangle \lhd \langle\langle \mathsf{\_,student,student{:}sid} \rangle\rangle, \langle\langle \mathsf{student} \rangle\rangle \rhd \langle\langle \mathsf{\_,student,student{:}sid} \rangle\rangle,$
$\langle\langle \mathsf{student{:}sid} \rangle\rangle \rhd \langle\langle \mathsf{\_,student,student{:}sid} \rangle\rangle, \langle\langle \mathsf{student} \rangle\rangle \lhd \langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle,$
$\langle\langle \mathsf{student} \rangle\rangle \rhd \langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle, \langle\langle \mathsf{student} \rangle\rangle \xrightarrow{\text{id}} \langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle,$
$\langle\langle \mathsf{student{:}name} \rangle\rangle \rhd \langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle,$
$\langle\langle \mathsf{result{:}grade} \rangle\rangle \rhd \langle\langle \mathsf{\_,}\langle\langle \mathsf{result,student,course} \rangle\rangle\mathsf{,result{:}grade} \rangle\rangle,$
$\langle\langle \mathsf{result,student,course} \rangle\rangle \lhd \langle\langle \mathsf{\_,}\langle\langle \mathsf{result,student,course} \rangle\rangle\mathsf{,result{:}grade} \rangle\rangle,$
$\langle\langle \mathsf{course} \rangle\rangle \lhd \langle\langle \mathsf{\_,course,course{:}dept} \rangle\rangle, \langle\langle \mathsf{course} \rangle\rangle \rhd \langle\langle \mathsf{\_,course,course{:}dept} \rangle\rangle,$
$\langle\langle \mathsf{course{:}dept} \rangle\rangle \rhd \langle\langle \mathsf{\_,course,course{:}dept} \rangle\rangle, \langle\langle \mathsf{course} \rangle\rangle \lhd \langle\langle \mathsf{\_,course,course{:}code} \rangle\rangle,$
$\langle\langle \mathsf{course} \rangle\rangle \rhd \langle\langle \mathsf{\_,course,course{:}code} \rangle\rangle, \langle\langle \mathsf{course} \rangle\rangle \xrightarrow{\text{id}} \langle\langle \mathsf{\_,course,course{:}code} \rangle\rangle,$
$\langle\langle \mathsf{course{:}code} \rangle\rangle \rhd \langle\langle \mathsf{\_,course,course{:}code} \rangle\rangle\}$ □

Most of these constraint operators have been used before in the context of describing single modelling languages. In particular, mandatory and unique constraints (though more limited in definition) have been used in a hypergraph model for relational schemas in [51], and inclusion constraints appear in [42]. However the combination of our mandatory, unique and reflexive constraints give a rich framework in which to express various notions of cardinality constraints and keys found in higher level modelling languages.

To illustrate these constraints, consider from the HDM schema in Example 1 the nodes $\langle\langle \mathsf{student} \rangle\rangle$ and $\langle\langle \mathsf{student{:}name} \rangle\rangle$ connected by edge $\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle$. As will be discussed in depth later, this might be used to model an ER entity called student with an attribute name. Suppose we have five data sources with the same schema S but different instances I_1, I_2, I_3, I_4, I_5 for which:

$x \in 1,2,3,4 : Ext_{S,I_x}(\langle\langle \mathsf{student} \rangle\rangle) = \{1,2\}$
$Ext_{S,I_5}(\langle\langle \mathsf{student} \rangle\rangle) = \{\text{‘Peter’}, \text{‘Mike’}\}$
$x \in 1,2,3,4,5 : Ext_{S,I_x}(\langle\langle \mathsf{student{:}name} \rangle\rangle) = \{\text{‘Peter’}, \text{‘Mike’}\}$
$Ext_{S,I_1}(\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle) = \{\langle 1, \text{‘Peter’}\rangle, \langle 1, \text{‘Mike’}\rangle, \langle 2, \text{‘Mike’}\rangle\}$
$Ext_{S,I_2}(\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle) = \{\langle 2, \text{‘Peter’}\rangle, \langle 2, \text{‘Mike’}\rangle\}$
$Ext_{S,I_3}(\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle) = \{\langle 1, \text{‘Mike’}\rangle, \langle 2, \text{‘Mike’}\rangle\}$
$Ext_{S,I_4}(\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle) = \{\langle 1, \text{‘Peter’}\rangle, \langle 2, \text{‘Mike’}\rangle\}$
$Ext_{S,I_5}(\langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle) = \{\langle \text{‘Peter’}, \text{‘Peter’}\rangle, \langle \text{‘Mike’}, \text{‘Mike’}\rangle\}$

To state that every instance of $\langle\langle \mathsf{student} \rangle\rangle$ must appear in the edge (and hence that I_2 is invalid), we use the mandatory constraint:

$\langle\langle \mathsf{student} \rangle\rangle \rhd \langle\langle \mathsf{\_,student,student{:}name} \rangle\rangle$

For the ER model, this constraint would be used when the attribute is mandatory for the entity (*i.e.* to state that each student must have a name). To state that every instance of ⟨⟨student⟩⟩ must appear no more than once in the edge (and hence that I_1 is invalid), we use the unique constraint:

⟨⟨student⟩⟩ $\lhd$ ⟨⟨_,student,student:name⟩⟩

For the ER model, this constraint would be used when the attribute is not multi-valued for the entity (*i.e.* to state that each student has no more than one name). The two above constraints together model that each instance of ⟨⟨student⟩⟩ must appear exactly once in the edge, and hence for the ER model a mandatory and single valued attribute (*i.e.* to state that each student has exactly one name).

We will illustrate as we consider different higher level modelling languages how these general notions of cardinality may be used to represent optional and mandatory attributes, and also both look-here and look-across [45,21] cardinality constraints on relationships/associations. The term **look-across** is used to denote the use of cardinality constraints where the cardinality written against entity x_i in an n-ary relationship between entities $x_1, \ldots, x_n$ restricts the participation of the other $n-1$ entities in the relationship. The term **look-here** is used to denote cardinality constraints that restrict the participation of x_i in the relationship.

Here, let us consider the general notion of a key, and how it is represented using our HDM constraints. If we want to model that ⟨⟨student:name⟩⟩ is a **candidate key** for ⟨⟨student⟩⟩ — that is to say that each value in ⟨⟨student:name⟩⟩ is in a one-to-one correspondence with a value in ⟨⟨student⟩⟩ (and thus that only I_4 and I_5 are valid out of the five instances above) — we must in addition make ⟨⟨student:name⟩⟩ also be 1:1 in the edge:

⟨⟨student:name⟩⟩ $\lhd$ ⟨⟨_,student,student:name⟩⟩

⟨⟨student:name⟩⟩ $\rhd$ ⟨⟨_,student,student:name⟩⟩

Thus in the ER case, name would be unique to each student. If we want to model that ⟨⟨student:name⟩⟩ is the **primary key** for ⟨⟨student⟩⟩ — that is to say that the values in ⟨⟨student:name⟩⟩ equal those in ⟨⟨student⟩⟩ (thus only I_5 is valid) — we must make ⟨⟨student⟩⟩ be reflexive in the edge:

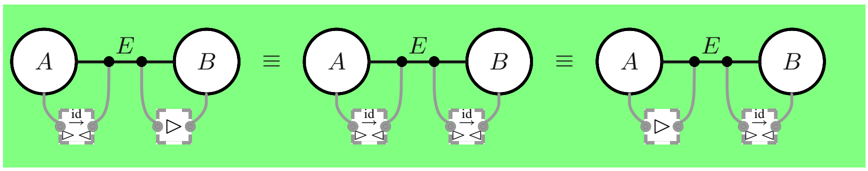
⟨⟨student⟩⟩ $\xrightarrow{\text{id}}$ ⟨⟨_,student,student:name⟩⟩

In general, reflexive, mandatory and unique together enforce that the entity has the same values as the attribute, and thus that the extent of the entity is the key values of the entity. This is because mandatory and unique at both ends of an edge enforce a one-to-one mapping, and the reflexive constraint means that this one-to-one mapping is an identity function. By contrast, lack of the reflexive constraint would be used when the entity has as its extent a set of object identifiers.

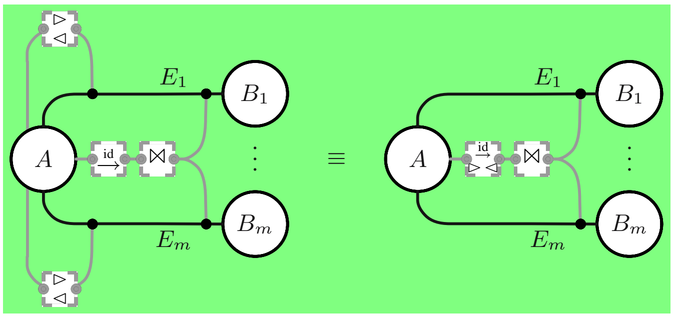
Note that we could have alternatively put the reflexive constraint on the other node as:

⟨⟨student:name⟩⟩ $\xrightarrow{\text{id}}$ ⟨⟨_,student,student:name⟩⟩

Also note that the unique constraint on the opposite end of the edge to the reflexive constraint is redundant, since it is implied by the other constraints. Hence we draw the equivalences shown in Figure 2(a) that will be used during intermodel transformation, where one higher level modelling language might happen to place constraints in a different but equivalent manner to another higher level modelling language.



(a) Transposing a reflexive constraint across an edge



(b) Mandatory-unique constraints in joins

Fig. 2. Fundamental equivalences on HDM constraints

Compound keys will require that we have a definition of an **edge natural join** given in Definition 6. This will be used to say that the reflexive constraint applies to more that one node. The introduction of the natural join means that we have a variation of the equivalence in Figure 2(a) given in Figure 2(b) that applies across an edge natural join.

Definition 6 Natural join between HDM edges
A view over HDM edges may be formed by joining edges together to form a new virtual edge:

$$\langle\langle E, A, B_1, \ldots, B_n\rangle\rangle \bowtie \langle\langle E, A, C_1, \ldots, C_m\rangle\rangle = \langle\langle E, A, B_1, \ldots, B_n, C_1, \ldots, C_m\rangle\rangle$$

The extent of the virtual edge is defined by a natural join over the extent of the two joined edges:

$$\begin{aligned} Ext_{S,I}(\langle\langle E, A, B_1, \ldots, B_n, C_1, \ldots, C_m\rangle\rangle) = \{&\langle x, y_1, \ldots, y_n, z_1, \ldots, z_m\rangle \\ &\mid \langle x, y_1, \ldots, y_n\rangle \in Ext_{S,I}(\langle\langle E, A, B_1, \ldots, B_n\rangle\rangle) \land \\ &\langle x, z_1, \ldots, z_m\rangle \in Ext_{S,I}(\langle\langle E, A, C_1, \ldots, C_m\rangle\rangle)\} \end{aligned}$$

□

2.1 An Example UoD and Four Schemas

Figures 3(a), 4(a), 5(a) and 6(a) show four data models, in ER, relational, UML and ORM data modelling languages. These are designed to cover the same UoD, and as will be shown later, three of them have the same information capacity [32]. The schemas represent a record of students, the courses that they sit, and the grades they obtain for those courses. Some students are undergraduates, and each ug has an associated personal programming tutor ppt that other students do not have. The use of underlining in the relational and ER schemas indicates what are key attributes, and a question mark follows a nullable attribute in those schemas. In the relational schema, foreign keys are shown by using an implication between the foreign key columns and the referenced table columns. In the ER schema this foreign key may either be represented by a relationship (for example the foreign keys result.name $\rightarrow$ student.name and result.code $\rightarrow$ course.code are represented in the ER result relationship) or by a subset (for example the foreign key ug.name $\rightarrow$ student.name is represented by a subset between the student and ug entities).

To describe how we map between high level modelling languages and the HDM, we use a simple production rule language as described in Definition 7.

Definition 7 HDM Production Rules
Higher level modelling language constructs are transformed into the HDM using production rules of the form:

$$\begin{array}{l}\langle\text{high level construct name}\rangle\ \langle\text{high level construct scheme}\rangle \leadsto \langle\text{HDM scheme}\rangle*\\ \quad\langle\text{condition}\rangle_1 \Rightarrow \langle\text{HDM constraint}\rangle_1\\ \quad\vdots\\ \quad\langle\text{condition}\rangle_n \Rightarrow \langle\text{HDM constraint}\rangle_n\end{array}$$

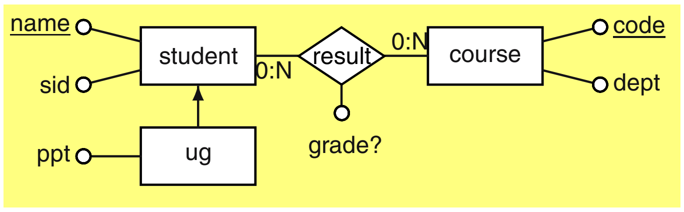
Where

- $\langle$high level construct scheme$\rangle$ is the structure used to represent a higher level model construct of type $\langle$high level construct name$\rangle$,
- $\langle$HDM scheme$\rangle$ is the list of zero or more HDM nodes or edges used to represent those aspects $\langle$high level construct scheme$\rangle$ that have an extent. Zero such schemes will be denoted using $\perp$, and the last $\langle$HDM scheme$\rangle$ is used to return the entire extent of the $\langle$high level construct scheme$\rangle$
- $\langle$condition$\rangle$ is a boolean expression over elements of of $\langle$high level construct scheme$\rangle$, which when satisfied, causes $\langle$HDM constraint$\rangle$ to be added to the $\langle$HDM scheme$\rangle$s. $\square$

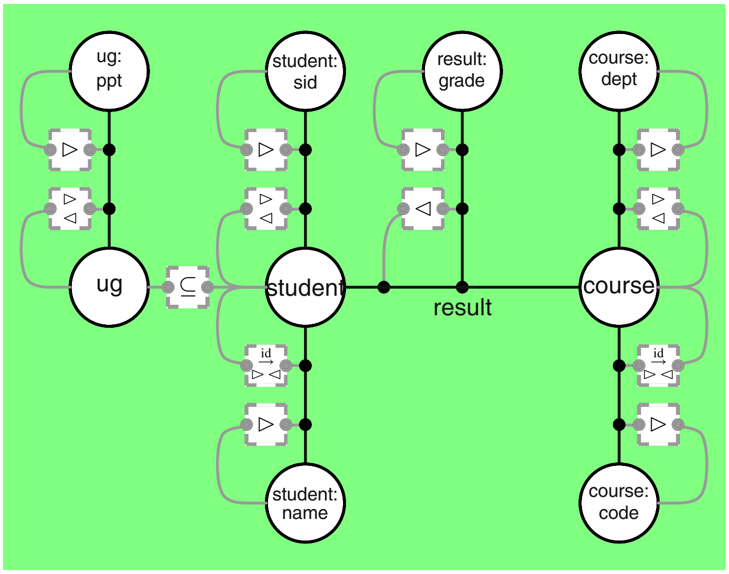
2.2 Describing an ER Modelling Language in the HDM

When the HDM is used to model a higher level modelling language, each construct in that language must be classified as being one of four types, each of which imply a different representation in the HDM. We explain how this methodology (first presented in [28]) is applied to an ER modelling language (which we describe here, see [45,35] for surveys of variations of ER modelling languages), and illustrate our discussions by showing how the methodology may take the ER schema of Figure 3(a) and produce the HDM schema of Figure 3(b). Note that in the HDM diagrams, HDM nodes are represented by white circles with thick outlines, and HDM edges are represented by thick black lines. The HDM constraint language is represented by grey dashed boxes connected by grey lines to the nodes and edges to which the constraint applies. Edges pass through black circles in a straight line, hence any edge or constraint applying to an edge meets that edge at a angle.

A summary of the AutoMed high level schemes that represent the high level ER schema in textual format are listed in Table 1, along with the class each belongs to. A similar table could be generated for the relational, UML and ORM models considered in the following sections. The production rules presented in this sub-section map these high level model schemes into the HDM schema already given in Examples 1 and 2.



(a) An ER schema of the student-course database



(b) HDM representation of the ER schema

Fig. 3. An ER schema and its equivalent HDM schema

The methodology in [28] categorises the constructs of a higher level model as being nodal, link-nodal, link, or constraint. We illustrate these categories by considering how the constructs of an ER modelling language will be handled.

Nodal. A **nodal** construct is one that may appear in isolation in a schema, such as an ER model **entity**. Using the AutoMed data integration system [8], such constructs are defined by giving a prototype **scheme** that must contain the name of a HDM node used to represent that construct. Hence we represent the ER entity student by the schema ⟪student⟫.

The production rule for an ER entity $\langle\langle E \rangle\rangle$ is very simple, since it states that each entity with scheme $\langle\langle E \rangle\rangle$ maps to a single HDM node $\langle\langle E \rangle\rangle$, and has no constraints:
entity $\langle\langle E \rangle\rangle \rightsquigarrow \langle\langle E \rangle\rangle$

Link. A **link** construct is one that associates other constructs with each other, and which has an extent which is drawn from those constructs, such as an ER **relationship**

Table 1. AutoMed high level model schemes for the ER example schema

class	construct	scheme
nodal	entity	⟨⟨student⟩⟩
link-nodal	attribute	⟨⟨student,name,notnull⟩⟩
constraint	key	⟨⟨student,name⟩⟩
link-nodal	attribute	⟨⟨student,sid,notnull⟩⟩
nodal	entity	⟨⟨ug⟩⟩
link-nodal	attribute	⟨⟨ug,ppt,notnull⟩⟩
constraint	subset	⟨⟨student,ug⟩⟩

class	construct	scheme
nodal	entity	⟨⟨course⟩⟩
link-nodal	attribute	⟨⟨course,code,key⟩⟩
constraint	key	⟨⟨course,code⟩⟩
link-nodal	attribute	⟨⟨course,dept,notnull⟩⟩
link	relationship	⟨⟨result,student,0:N,course,0:N⟩⟩
link-nodal	attribute	⟨⟨result,grade,null⟩⟩

construct. In AutoMed, we represent ER relationships by the scheme comprising of the name of the HDM edge used to represent the construct, together with pairs of the entity names and cardinality constraints. For example, we represent the ER relationship result in Figure 3(a) by the scheme $\langle\langle \mathsf{result}, \mathsf{student}, \mathsf{0{:}N}, \mathsf{course}, \mathsf{0{:}N}\rangle\rangle$. The production rule uses auxiliary rules to generate the constraints in the HDM necessary to represent the cardinality constraints in the ER schema. Assuming that our ER model uses look-here semantics [45,21], we need one line for each entity E_x that generates as appropriate $\rhd$ and $\lhd$ using a function generate_card() defined in Definition 8.

$$\mathsf{relationship}\ \langle\langle R, E_1, L_1{:}U_1, \ldots, E_n, L_n{:}U_n\rangle\rangle \rightsquigarrow \langle\langle R, E_1, \ldots, E_n\rangle\rangle$$
$$\mathsf{true} \Rightarrow \mathsf{generate\_card}(E_1, \langle\langle R, E_1, \ldots, E_n\rangle\rangle, L_1, U_1)$$
$$\vdots$$
$$\mathsf{true} \Rightarrow \mathsf{generate\_card}(E_n, \langle\langle R, E_1, \ldots, E_n\rangle\rangle, L_n, U_n)$$

Definition 8 Generation of constraints representing cardinality
This function generates cardinality constraints between a set of nodes or edges $\{NE_1, \ldots, NE_n\} \in Schemes$ and $E \in Edges$, where L may be either 0 or 1, and U may be 1 or *.

$$\mathsf{generate\_card}\ (\langle NE_1, \ldots, NE_n\rangle, E, L, U) \rightsquigarrow \bot$$
$$L = 1 \Rightarrow \langle NE_1, \ldots, NE_n\rangle \rhd E$$
$$U = 1 \Rightarrow \langle NE_1, \ldots, NE_n\rangle \lhd E$$ □

Note that the rule in Definition 8 takes as its first argument a tuple of nodes and edges $NE_1, \ldots, NE_n$ that must appear in the edge E that is its second argument, and then produces mandatory and unique constraints to determine how many times a particular combination of values from the extent of $NE_1, \ldots, NE_n$ may appear in the extent of E.

Applying our production rule to the relationship $\langle\langle \mathsf{result}, \mathsf{student}, \mathsf{0{:}N}, \mathsf{course}, \mathsf{0{:}N}\rangle\rangle$ produces an edge $\langle\langle \_, \mathsf{result}, \mathsf{student}\rangle\rangle$ to represent its extent. The auxiliary constraint rules will produce no constraints, since neither of the guards within the definition of generate_card will match $L = 0$ or $U = *$.

Link-Nodal. A **link-nodal** construct is one that has associated values, but may only exist when associated with some other construct. They are represented in the HDM by an edge associating a new node with some existing node or edge. For example, ER **attributes** are link-nodal constructs, and the name attribute of the entity student is represented in AutoMed by the scheme $\langle\langle \mathsf{student}, \mathsf{name}, \mathsf{notnull} \rangle\rangle$. The production rule for ER attributes creates a node and edge.

$\mathsf{attribute}\ \langle\langle E, A, N \rangle\rangle \rightsquigarrow \langle\langle E{:}A \rangle\rangle, \langle\langle \_, E, E{:}A \rangle\rangle$
$\quad \mathsf{true} \Rightarrow \mathsf{generate\_card}(\langle\langle E{:}A \rangle\rangle, \langle\langle \_, E, E{:}A \rangle\rangle, 1, {}^*)$

$\quad N = \mathsf{notnull} \Rightarrow \mathsf{generate\_card}(\langle\langle E \rangle\rangle, \langle\langle \_, E, E{:}A \rangle\rangle, 1, 1)$
$\quad N = \mathsf{null} \Rightarrow \mathsf{generate\_card}(\langle\langle E \rangle\rangle, \langle\langle \_, E, E{:}A \rangle\rangle, 0, 1)$

Thus the production rule when applied to the ER attribute ⟨⟨course,dept,notnull⟩⟩ produces the node ⟨⟨course:dept⟩⟩ and the edge ⟨⟨_,course,course:dept⟩⟩ to represent the extent of the attribute. The first auxiliary constraint rule produces ⟨⟨course:dept⟩⟩ $\rhd$ ⟨⟨_,course,course:dept⟩⟩, the second produces ⟨⟨course⟩⟩ $\rhd$ ⟨⟨_,course,course:dept⟩⟩ and ⟨⟨course⟩⟩ $\lhd$ ⟨⟨_,course,course:dept⟩⟩ (since both guards in the generate_card are met), and then the last rule fails to match in the guard.

Note that since the grade attribute is optional, we obtain just two constraints when the production rule is used on ⟨⟨result,grade,null⟩⟩:

⟨⟨result:grade⟩⟩ $\rhd$ ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩
⟨⟨result,student,course⟩⟩ $\lhd$ ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩

Note that in our modelling of the ER model (and relational and UML languages), the fact that attribute names are prefixed by the associated entity name reflects a deliberate choice made when defining the construct. One could alternatively say that attribute names are globally unique, which would change the HDM graph to have just one node ⟨⟨name⟩⟩ to represent both the ⟨⟨student, name, notnull⟩⟩ and ⟨⟨ug, name, notnull⟩⟩ relational columns, but this would not give the correct semantics for a normal ER model. The alternative global naming choice will be used in modelling the value types of ORM models.

In Figure 3(b) it should be noted that the syntax is not ambiguous, but does need careful reading. Each $\rhd$ or $\lhd$ always has a node or edge on its left hand side that appears in the edge on its right hand side. We use this fact to ignore which 'side' we connect $\rhd$ and $\lhd$ constraints to in the diagram. This makes the diagrams more tidy in appearance. (Note that this is different from the approach we followed in our earlier work [9]). Therefore the ⟨⟨course:dept⟩⟩ to ⟨⟨_,course,course:dept⟩⟩ mandatory constraint is drawn using $\rhd$ in the constraint box.

Constraint. A **constraint** construct is one that has no associated extent, but instead limits the extent of the constructs it connects to. An example of a constraint construct is the ER model **subset** relationship. For example, the subset between ug and student is represented in AutoMed by the scheme ⟨⟨student, ug⟩⟩.
$\mathsf{subset}\ \langle\langle E, E_s \rangle\rangle \rightsquigarrow \bot$
$\quad \mathsf{true} \Rightarrow \langle\langle E_s \rangle\rangle \subseteq \langle\langle E \rangle\rangle$

ER **generalisations** are another example of constraint constructs. For example, a generalisation that specified the children entities $\langle\langle E_1 \rangle\rangle, \ldots, \langle\langle E_n \rangle\rangle$ are disjoint subsets of some parent entity $\langle\langle E \rangle\rangle$ could be defined by the following rule:

$\mathsf{generalisation}\ \langle\langle E, E_1, \ldots, E_n \rangle\rangle \rightsquigarrow \bot$
$\quad \mathsf{true} \Rightarrow \langle\langle E_1 \rangle\rangle \subseteq \langle\langle E \rangle\rangle$
$\quad\quad \Rightarrow \vdots$
$\quad \mathsf{true} \Rightarrow \langle\langle E_n \rangle\rangle \subseteq \langle\langle E \rangle\rangle$
$\quad \mathsf{true} \Rightarrow \langle\langle E_1 \rangle\rangle \not\cap \ldots \not\cap \langle\langle E_n \rangle\rangle$

Our example ER schema in Figure 3(a) contains no generalisations, but we will discuss and compare advanced modelling constructs of various data modelling languages in Section 4.

The final constraint in our ER model is the definition of the **key** of an entity, which serves to denote the set of its attributes that may be used to identify instances of the attribute.

$\mathsf{key}\ \langle\langle E, A_1, \ldots, A_n \rangle\rangle \rightsquigarrow \bot$
$\quad \mathsf{true} \Rightarrow \langle\langle E \rangle\rangle \overset{\mathrm{id}}{\rightarrow} \langle\langle \_, E, E{:}A_1 \rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, E, E{:}A_n \rangle\rangle$

The constraint limits the instances of the entity to be an identity with the join of its key attributes (we will give an example of how this type of constraint works when looking at the relational result table in the next section).

2.3 Describing Relational Schemas in the HDM

Having reviewed the general methodology for representing higher level modelling languages in the HDM in the previous subsection, we will now apply the methodology to the relational schema. Relational model **table**s are nodal constructs, and hence we represent the table student by the scheme ⟨⟨student⟩⟩, and the table result by ⟨⟨result⟩⟩. The production rule for translating such schemes into the HDM is as follows.

$\mathsf{table}\ \langle\langle T \rangle\rangle \rightsquigarrow \langle\langle T \rangle\rangle$

Relational model **column**s are link-nodal constructs, and hence are modelled by a scheme containing a HDM node that represents the construct it depends on, followed by the name of the HDM node that represents the column, followed by the constraint on whether the attribute may be null. For example, the name column of table student is represented by the scheme ⟨⟨student, name, notnull⟩⟩. In the HDM, this becomes a node ⟨⟨student:name⟩⟩ to represent values of the column/attribute, and the nameless edge ⟨⟨_,student,student:name⟩⟩ to represent the association of these values to table/entity ⟨⟨student⟩⟩.

$\mathsf{column}\ \langle\langle T, C, N \rangle\rangle \rightsquigarrow \langle\langle T{:}C \rangle\rangle, \langle\langle \_, T, T{:}C \rangle\rangle$
$\quad \mathsf{true} \Rightarrow \mathsf{generate\_card}(\langle\langle T{:}C \rangle\rangle, \langle\langle \_, T, T{:}C \rangle\rangle, 1, *)$
$\quad N = \mathsf{notnull} \Rightarrow \mathsf{generate\_card}(\langle\langle T \rangle\rangle, \langle\langle \_, T, T{:}C \rangle\rangle, 1, 1)$
$\quad N = \mathsf{null} \Rightarrow \mathsf{generate\_card}(\langle\langle T \rangle\rangle, \langle\langle \_, T, T{:}C \rangle\rangle, 0, 1)$

The definition of relational columns and ER attributes are very similar, and as can be seen by comparing Figures 4 and 3, produce similar results in the HDM.

The **primary key** construct of the relational model is a constraint construct. The constraint specifies that the natural join between its key columns gives the extent of the table. Although a slightly unconventional notion of primary key, this definition fits well with the sixth normal form [13,12], since that normalises tables to have one table for the primary key columns, and then an additional table for each non-key column of the

ug	
name	ppt
Mary	NR
Jane	SK

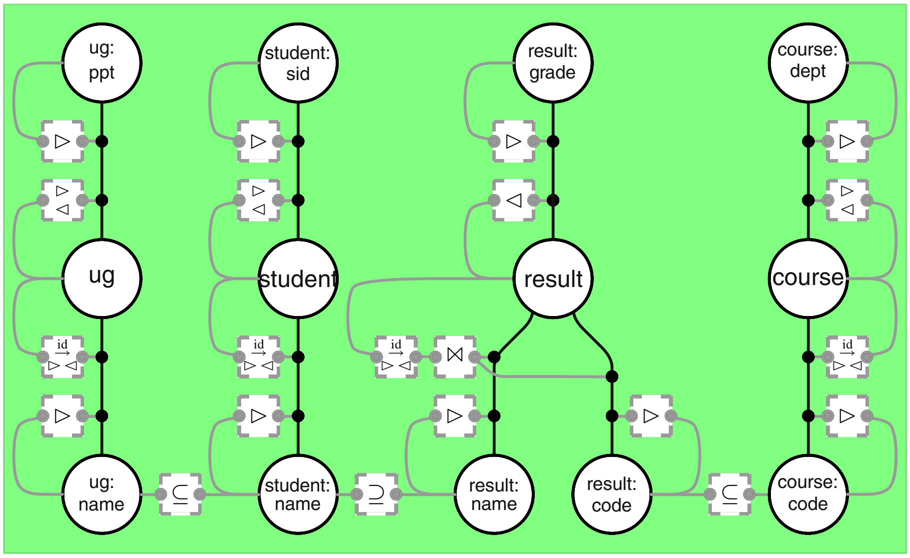
student	
name	sid
Mary	1
John	2
Jane	3
Fred	4

course	
code	dept
DB	CS
Fin	CS
Geo	Maths

result		
code	name	grade?
DB	Mary	A
Fin	Jane	C
Fin	Fred	null
Geo	Fred	A
Geo	John	B

ug.name $\rightarrow$ student.name
result.name $\rightarrow$ student.name
result.code $\rightarrow$ course.code

(a) Relational database schema and data



(b) HDM representation of relational database schema

Fig. 4. A relational schema for the student-course database

pre-normalised table. The schema of a constraint simply needs to list the table and the columns. For example, the primary key of ⟨⟨result⟩⟩ would be represented in the HDM by $\langle\langle \text{result} \rangle\rangle \xrightarrow{\text{id}} (\langle\langle \text{\_,result,result:code} \rangle\rangle \bowtie \langle\langle \text{\_,result,result:name} \rangle\rangle)$. The production rule to produce the HDM constraints is as follows:

primary_key $\langle\langle T, C_1, \ldots, C_n \rangle\rangle \rightsquigarrow \bot$

$\text{true} \Rightarrow \langle\langle T \rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, T, T{:}C_1 \rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n \rangle\rangle)$

Since any key column must also be a notnull column in a valid relational schema, this rule need only add the fact that the join of the edges leading to nodes representing key columns is reflexive. Since the table will be connected to these edges using mandatory and unique, it follows that the join is also mandatory and unique, as shown for the result node in Figure 4(b). For the primary key scheme ⟨⟨result,name,code⟩⟩ for the result table, the production rule generates the reflexive constraint $\langle\langle \text{result} \rangle\rangle \xrightarrow{\text{id}} \langle\langle \text{\_,result,result:code} \rangle\rangle \bowtie \langle\langle \text{\_,result,result:name} \rangle\rangle$. For example, with the relational data shown in Figure 4(a), this constraint along with the already stated mandatory and unique constraints enforce the following type of instantiation of the $\langle\langle \mathsf{result}\rangle\rangle$ node and key edges:

$$\begin{aligned}
&\langle\langle \mathsf{result}\rangle\rangle = \{\langle \mathsf{DB}, \mathsf{Mary}\rangle, \langle \mathsf{Fin}, \mathsf{Jane}\rangle, \langle \mathsf{Fin}, \mathsf{Fred}\rangle, \ldots\}\\
&\langle\langle \_,\mathsf{result},\mathsf{result{:}name}\rangle\rangle =\\
&\quad \{\langle\langle \mathsf{DB}, \mathsf{Mary}\rangle, \mathsf{Mary}\rangle, \langle\langle \mathsf{Fin}, \mathsf{Jane}\rangle, \mathsf{Jane}\rangle, \langle\langle \mathsf{Fin}, \mathsf{Fred}\rangle, \mathsf{Fred}\rangle, \ldots\}\\
&\langle\langle \_,\mathsf{result},\mathsf{result{:}code}\rangle\rangle =\\
&\quad \{\langle\langle \mathsf{DB}, \mathsf{Mary}\rangle, \mathsf{DB}\rangle, \langle\langle \mathsf{Fin}, \mathsf{Jane}\rangle, \mathsf{Fin}\rangle, \langle\langle \mathsf{Fin}, \mathsf{Fred}\rangle, \mathsf{Fin}\rangle, \ldots\}
\end{aligned}$$

We represent the **foreign key** constraint by the scheme made up of a name for the constraint, the table and column(s) that are the foreign key, and the table and column(s) of the referenced table.

$$\begin{aligned}
&\mathsf{foreign\_key}\ \langle\langle FK, T, C_1, \ldots, C_n, T_f, C_{f_1}, \ldots, C_{f_n}\rangle\rangle \leadsto \bot\\
&\quad \mathsf{true} \Rightarrow \pi_{\langle\langle T:C_1\rangle\rangle,\ldots,\langle\langle T:C_n\rangle\rangle}(\langle\langle \_, T, T{:}C_1\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n\rangle\rangle) \subseteq\\
&\qquad\qquad \pi_{\langle\langle T_f:C_{f_1}\rangle\rangle,\ldots,\langle\langle T_f:C_{f_n}\rangle\rangle}(\langle\langle \_, T_f, T_f{:}C_{f_1}\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T_f, T_f{:}C_{f_1}\rangle\rangle)
\end{aligned}$$

The somewhat complex constraint simply states that the join of the columns listed in T is a subset of the join of the columns in T_f. For the common case where foreign keys are not compound keys (*i.e.* $n = 1$), the constraint would simplify to $\langle\langle T{:}C_1\rangle\rangle \subseteq \langle\langle T_f{:}C_{f_1}\rangle\rangle$ For example, the foreign key between ug and $\mathsf{student}$ is represented by the scheme $\langle\langle \mathsf{ug\_fk}, \mathsf{ug}, \mathsf{name}, \mathsf{student}, \mathsf{name}\rangle\rangle$. Using the production rule, this scheme becomes $\langle\langle \mathsf{ug{:}name}\rangle\rangle \subseteq \langle\langle \mathsf{student{:}name}\rangle\rangle$ in the HDM.

Finally, the relational **candidate key** takes a similar definition to primary key, except that all that is established is a mandatory and a unique association between the table and a join of the candidate key columns.

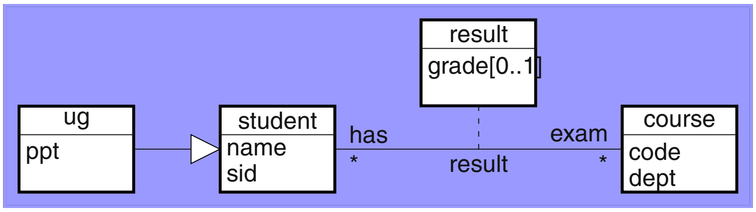
$$\begin{aligned}
&\mathsf{candidate\_key}\ \langle\langle T, C_1, \ldots, C_n\rangle\rangle \leadsto \bot\\
&\quad \mathsf{true} \Rightarrow \langle\langle T\rangle\rangle \lhd (\langle\langle \_, T, T{:}C_1\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n\rangle\rangle)\\
&\quad \mathsf{true} \Rightarrow \langle\langle T\rangle\rangle \rhd (\langle\langle \_, T, T{:}C_1\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n\rangle\rangle)\\
&\quad \mathsf{true} \Rightarrow (\pi_{\langle\langle T:C_1\rangle\rangle,\ldots,\langle\langle T:C_n\rangle\rangle}(\langle\langle \_, T, T{:}C_1\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n\rangle\rangle)) \lhd\\
&\qquad\qquad\qquad\qquad (\langle\langle \_, T, T{:}C_1\rangle\rangle \bowtie \ldots \bowtie \langle\langle \_, T, T{:}C_n\rangle\rangle)
\end{aligned}$$

The last line ensures that the combination of columns in the candidate key appears just once in the edge formed by the join of the candidate key column edges. In the common case where the candidate key is not compound (*i.e.* $n = 1$), the last constraint simplifies to $\langle\langle T{:}C_1\rangle\rangle \lhd \langle\langle \_, T, T{:}C_1\rangle\rangle$. Thus if we added the new candidate key $\langle\langle \mathsf{student},\mathsf{sid}\rangle\rangle$ to the example relational schema, then we would add to our existing relation HDM a $\langle\langle \mathsf{student{:}sid}\rangle\rangle \lhd \langle\langle \_,\mathsf{student},\mathsf{student{:}sid}\rangle\rangle$ constraint.

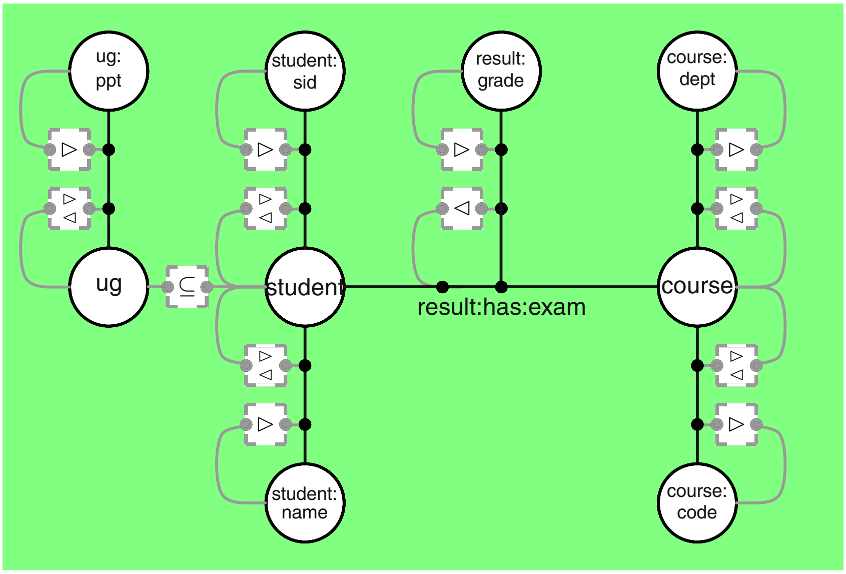
2.4 Describing UML in the HDM

UML **classes** are nodal constructs, and hence each UML class scheme $\langle\langle C\rangle\rangle$ maps to a single node $\langle\langle C\rangle\rangle$. The extent of $\langle\langle C\rangle\rangle$ is the set of unique **object identifiers** (**OID**) of the class.

$$\mathsf{class}\ \langle\langle C\rangle\rangle \leadsto \langle\langle C\rangle\rangle$$



(a) A UML class diagram of the student-course database



(b) HDM representation of the UML Schema

Fig. 5. A UML schema and its equivalent HDM schema

The definition of n-ary **associations** in UML states that the multiplicity, $L..U$, of a role, R, defines the number of instances of the class C that are associated with a particular set of values of the other classes in the association A. Thus using ER terminology, it has look-across semantics [45,21], and hence generate_card() is called for each role class with all the classes except the role class. We also make the assumption that * is simply a shorthand for 0..*, and any single number n is a shorthand for n..n (for example, in UML, one writes 1 as a shorthand for 1..1).

$$
\begin{array}{l}
\mathsf{association}\langle\!\langle A, R_1, C_1, L_1..U_1, \ldots, R_n, C_n, L_n..U_n\rangle\!\rangle \rightsquigarrow \langle\!\langle A{:}R_1{:}\ldots{:}R_n, C_1, \ldots, C_n\rangle\!\rangle \\
\quad \mathsf{true} \Rightarrow \mathsf{generate\_card}(\langle C_2, \ldots, C_n\rangle, \langle\!\langle A{:}R_1{:}\ldots{:}R_n, C_1, \ldots, C_n\rangle\!\rangle, L_1, U_1) \\
\qquad \vdots \\
\quad \mathsf{true} \Rightarrow \mathsf{generate\_card}(\langle C_1, \ldots, C_{n-1}\rangle, \langle\!\langle A{:}R_1{:}\ldots{:}R_n, C_1, \ldots, C_n\rangle\!\rangle, L_n, U_n)
\end{array}
$$

For example, the UML association between student and course has the scheme ⟨⟨result,has,student,0..*,exam,course,0..*⟩⟩, and the production rule maps this to the

HDM edge $\langle\langle$result:has:exam,student,course$\rangle\rangle$, with no constraints. Note that the label of the HDM edge is $A{:}R_1{:}\ldots{:}R_n$, which encodes the various labels HDM gives the association in a single HDM identifier. Thus the result association with role names 'has' and 'exam' gets the HDM edge name result:has:exam.

UML **attributes** are link-nodal constructs attached to CA, which is a UML class or a UML association, and hence the production rule takes a similar form to that for ER attributes or relational columns. We make the same assumptions about shorthands for the attribute multiplicity as we did for association multiplicity, as well as noting that the absence of explicit multiplicity means that 1..1 is assumed. Thus the sid attribute of student has the scheme $\langle\langle$student,sid,1..1$\rangle\rangle$.

attribute $\langle\langle CA, A, L..U\rangle\rangle \rightsquigarrow \langle\langle CA{:}A\rangle\rangle, \langle\langle \_, CA, CA{:}A\rangle\rangle$
  true $\Rightarrow$ generate_card$(\langle\langle CA{:}A\rangle\rangle, \langle\langle _, CA, CA{:}A\rangle\rangle, 1, {*})$
  $L..U \Rightarrow$ generate_card$(\langle\langle CA\rangle\rangle, \langle\langle _, CA, CA{:}A\rangle\rangle, L, U)$

Note that UML **association classes** are directly supported by these definitions of association and attribute. An association class is simply an association that has one or more attributes placed upon it, each UML attribute becoming an HDM node with a nested edge that connects that node to the HDM edge that represents the association.

UML **generalisations** have a sophisticated constraint system that specifies that the various classes or associations that are children of a parent class or association maybe overlapping or disjoint, and maybe complete or incomplete. The first and last of these keywords are 'noise' in the sense that they add nothing in addition to an unlabelled generalisation. The other two add an exclusion constraint and a union constraint.

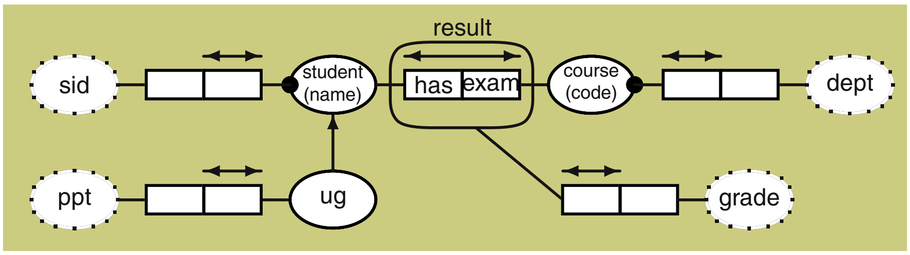
generalisation $\langle\langle C, C_1, \ldots, C_n, D\rangle\rangle \rightsquigarrow \perp$
  true $\Rightarrow \langle\langle C_1\rangle\rangle \subseteq \langle\langle C\rangle\rangle$
  $\Rightarrow \vdots$
  true $\Rightarrow \langle\langle C_n\rangle\rangle \subseteq \langle\langle C\rangle\rangle$
  disjoint $\in D \Rightarrow \langle\langle C_1\rangle\rangle \not\cap \ldots \not\cap \langle\langle C_n\rangle\rangle$
  complete $\in D \Rightarrow \langle\langle C\rangle\rangle = \langle\langle C_1\rangle\rangle \cup \ldots \cup \langle\langle C_n\rangle\rangle$

2.5 Describing the ORM in the HDM

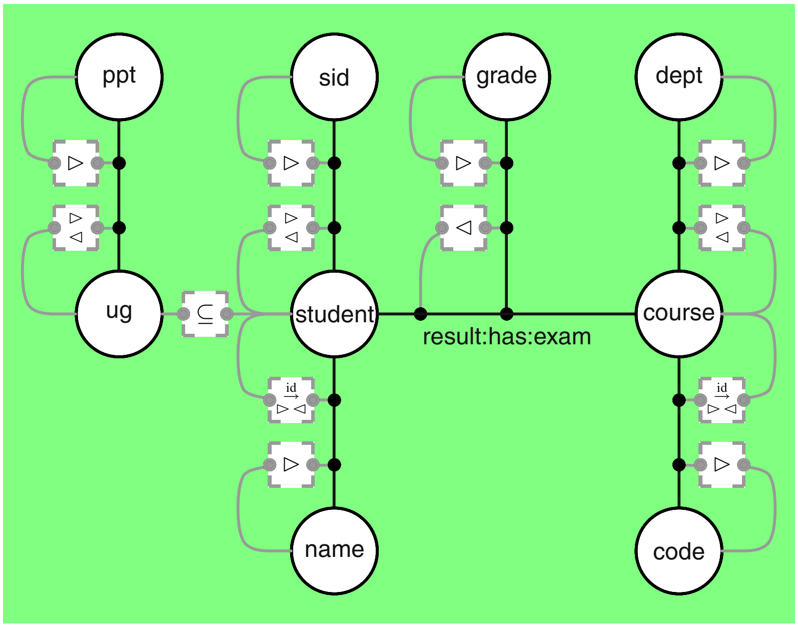
From our analysis of the ER, relational and UML modelling languages, it may seem 'obvious' that ORM entity types should be modelled as nodal constructs while value types should be modelled as link-nodal constructs. However, due to ORM's rich semantics, the similarity of value type and entity type roles in fact types, and a value-type's ability to play multiple roles in fact types, it is correct to model both value types and entity types using an HDM nodal construct type.

Each **entity type** $\langle\langle E\rangle\rangle$ maps to a single node $\langle\langle E\rangle\rangle$. The extent of entity type $\langle\langle E\rangle\rangle$ is the extent of its primary reference mode while the extent of a **value type** $\langle\langle V\rangle\rangle$ is just the ORM value type's set of values. Hence we have the simple definitions:
entity_type $\langle\langle E\rangle\rangle \rightsquigarrow \langle\langle E\rangle\rangle$
value_type $\langle\langle V\rangle\rangle \rightsquigarrow \langle\langle V\rangle\rangle$

An ORM n-ary **fact type** is an association between n objects where each object is an entity type, value type, or objectified fact type. A fact type's extent is drawn from the



(a) An ORM schema of the student-course database



(b) HDM representation of the ORM schema

Fig. 6. An ORM schema of the student-course database

objects it associates and is hence modelled in the HDM as a link construct. The scheme for the ORM fact type should describe the name FT of the fact type (if any) along with the role name N (if any), role object R, and the mandatory nature of the object type M in the role. Hence the fact type between ⟨⟨student⟩⟩ and ⟨⟨course⟩⟩ has the scheme ⟨⟨result,has,student,_,exam,course,_⟩⟩ and the fact type between ⟨⟨student⟩⟩ and ⟨⟨sid⟩⟩ has scheme ⟨⟨_,student,_,•,sid,_,_⟩⟩ (where $M = \bullet$ corresponds to the black circle used in ORM on objects to denote mandatory roles). These then map into the HDM using the following production rule:

$$\mathsf{fact\_type}\ \langle\langle FT, N_1, R_1, M_1, \ldots, N_n, R_n, M_n\rangle\rangle \leadsto \langle\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n\rangle\rangle$$

$$M_1 = \bullet \Rightarrow \mathsf{generate\_card}(R_1, \langle\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n\rangle\rangle, 1, *)$$

$$\vdots$$

$$M_n = \bullet \Rightarrow \mathsf{generate\_card}(R_n, \langle\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n\rangle\rangle, 1, *)$$

The names of fact types and roles are encoded into a single HDM edge label in a similar manner to that used to encode UML association and role names into a single HDM edge label. Note that all fact types have an implied uniqueness constraint across all n roles, and HDM has a similar edge constraint because the extent of an edge is a set of tuples. In ORM one can specify uniqueness constraints across $n-1$ roles of an n role fact type. Hence we define the scheme of **uniqueness** to take both a fact type and the role R_x that is uniquely identified by the other roles:

$$\begin{array}{l}\mathsf{uniqueness}\ \langle\!\langle\langle\!\langle FT, N_1, R_1, M_1, \ldots, N_n, R_n, M_n\rangle\!\rangle, R_x\rangle\!\rangle \rightsquigarrow \bot \\ \quad \mathsf{true} \Rightarrow \mathsf{generate\_card}(\langle R_1, \ldots, R_{x-1}, R_{x+1}, \ldots, R_n\rangle, \\ \qquad\qquad\qquad\qquad \langle\!\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n\rangle\!\rangle, 0, 1\ \ )\end{array}$$

The production rule implements the cardinality constraint using $\mathsf{generate\_card}()$ being called with all roles of the fact type accept the R_x being uniquely identified.

We note in passing that ORM also has a general **frequency constraint** type across any number of roles. We have not needed this in our examples, but could have modelled the mandatory and unique constraints using the more general frequency constraint; but as there are implicit unique and mandatory constraints in an ORM schema and these constraints are used heavily in the rules regarding a schema's well formedness, it is useful to model mandatory and unique as we have here.

ORM can express **subtype** relationships between fact roles as well as entity types. We only use subtyping between entity types in our examples, hence we shall restrict ourselves to just defining that below, together with the notion of disjointness and totality of such subtypes which ORM also supports:

$$\begin{array}{l}\mathsf{subset}\ \langle\!\langle EV, EV_s\rangle\!\rangle \rightsquigarrow \bot \\ \quad \mathsf{true} \Rightarrow \langle\!\langle EV_s\rangle\!\rangle \subseteq \langle\!\langle EV\rangle\!\rangle \\ \mathsf{disjoint}\ \langle\!\langle EV_1\rangle\!\rangle, \langle\!\langle EV_2\rangle\!\rangle \rightsquigarrow \bot \\ \quad \mathsf{true} \Rightarrow \langle\!\langle EV_1\rangle\!\rangle \not\cap \langle\!\langle EV_2\rangle\!\rangle \\ \mathsf{total}\ \langle\!\langle EV, EV_1\rangle\!\rangle, \langle\!\langle EV, EV_2\rangle\!\rangle \rightsquigarrow \bot \\ \quad \mathsf{true} \Rightarrow \langle\!\langle EV\rangle\!\rangle = \langle\!\langle EV_1\rangle\!\rangle \cup \langle\!\langle EV_2\rangle\!\rangle\end{array}$$

Our simple UoD does not use the above ORM constructs, but applying the above definitions to the ORM diagram in Figure 6(a) produces the HDM in Figure 6(b), almost the same HDM we arrived at using the ER schema bar some trivial renaming of nodes and edges. Note that we have assumed that the value classes implied by the primary reference modes for each entity class have been made explicit before applying the definitions (ORM allows these to be implicit, and not stated in the ORM schema).

3 Inter Model Transformations

We now introduce five general purpose equivalence mappings that may be used on our HDM constraint operators, and which allow us to transform between different modelling languages. In particular, the relational HDM schema in Figure 4(b) may be transformed into the ER HDM schema in Figure 3(b) by applying a sequence of transformations using four of the equivalence relationships.

Section 3.6 describes the fifth general purpose equivalence preserving rule, and shows how it is used as part of the transformation of the UML HDM schema in Figure 5(b) to the ER HDM schema. However, the UML to ER transformation as a whole will be demonstrated to be non-equivalence preserving.

In order to give our mappings a rigorous basis, we define them using the BAV transformation language [39,28,30,31] which allows the specification of bidirectional mappings between equivalent data sources, and also the specification of the situation where one data source has greater information capacity than another. Hence, we first review BAV primitive transformations for the HDM in the next subsection, before giving the five mappings in Sections 3.2–3.6 defined using those transformation primitives. Finally we discuss in Section 3.7 how non-equivalent data sources are identified and handled in our approach.

3.1 HDM BAV Transformations

In the BAV approach, schemas are incrementally transformed by applying to them a **pathway** of primitive schema transformations $t_1, \ldots, t_r$. Each primitive transformation t_i makes a 'delta' change to the schema by adding, deleting or renaming just one HDM node, edge or constraint. The model management [5] concept of a Mapping between S_1, S_2 is implemented by having a well-formed pathway [47] between the two schemas. Note that the model management concept of Compose is directly supported by the BAV approach, since it allows a pathway between S_2, S_3 to be appended to the pathway between S_1, S_2 to give a pathway between S_1, S_3. Details of how pathways can be analyzed for their impact on information capacity can be found in [27], and work in using BAV to perform the model management Match is found in [40] and Merge is found in [41].

Table 2 lists those primitive transformations of the BAV language that we use in this paper, and we now briefly review their semantics.

Table 2. BAV primitive transformations applied to HDM schema $S = \langle Nodes, Edges, Cons \rangle$, to generate a new schema $S' = \langle Nodes', Edges', Cons' \rangle$

primitive transformation $S \rightarrow S'$	reverse transformation $S' \rightarrow S$	conditions onS, S'	information capacity
addNode(n,q)	deleteNode(n,q)	$n \notin Nodes, n \in Nodes'$	$S \equiv S'$
addEdge(e,q)	deleteEdge(e,q)	$e \notin Edges, e \in Edges'$	$S \equiv S'$
addCons(c)	deleteCons(c)	$c \notin Cons, c \in Cons'$	$S \equiv S'$
renameNode(n,n')	renameNode(n',n)	$n \in Nodes, n \notin Nodes'$, $n' \notin Nodes, n' \in Nodes'$	$S \equiv S'$
renameEdge(e,e')	renameEdge(e',e)	$e \in Edges, e \notin Edges'$, $e' \notin Edges, e' \in Edges'$	$S \equiv S'$
extendNode(n,q_l,q_u)	contractNode(n,q_l,q_u)	$n \notin Nodes, n \in Nodes'$	$S \subset S'$
extendEdge(e,q_l,q_u)	contractEdge(e,q_l,q_u)	$e \notin Edges, e \in Edges'$	$S \subset S'$
extendCons(c)	contractCons(c)	$c \notin Cons, c \in Cons'$	$S \supset S'$

The primitive transformation that adds a node n to a schema S in order to generate new schema S' is $\textsf{addNode}(n, q)$, where q is a query over S specifying the extent of n in terms of the existing constructs of S. The logical semantics of this kind of transformation are

$$\forall I. Ext_{S,I}(n) = q \tag{1}$$

and for this reason we categorise addNode as an **exact transformation** [25], and the two schemas S, S' have equivalent information capacity, summarised in Table 2 by putting $S \equiv S'$ in the information capacity column.

When it is not possible to specify the exact extent of the new node n in terms of the existing schema constructs, we must instead of addNode use $\textsf{extendNode}(n, q_l, q_u)$, where q_l gives the lower bound on the extent of n, and q_u gives the upper bound. The logical semantics of this kind of transformation are

$$\forall I. q_u \supseteq Ext_{S,I}(n) \supseteq q_l \tag{2}$$

and so we term extend a **sound transformation** [25] when considering q_l and a **complete transformation** [25] when considering q_u. The query q_l may just be the constant Void, indicating no values in the extent can be derived from other constructs in the schema. The query q_u may just be the constant Any, indicating that no limit of the values in the extent can be derived from other constructs in the schema. If $q_l =$ Void and $q_u =$ Any then the two queries may be omitted (and only this form of the transformation is used in the paper). Note that S has an information capacity which is a subset of that of S', which in Table 2 by putting $S \subset S'$ in the information capacity column.

The exact transformation $\textsf{deleteNode}(c, q)$ when applied to schema S' generates a new schema S with node n removed. The extent of n may be recovered using the query q on S, and Equation 1 above holds. Note that this implies that from a primitive transformation deleteNode(n,q) used to transform $S' \rightarrow S$ we can automatically derive that addNode(n,q) transforms $S \rightarrow S'$, and *vice versa*.

When it is not possible to specify the exact extent of node n being deleted from S' in terms of the remaining schema constructs, contractNode(n,q_l,q_u) must be used instead of deleteNode, where Equation 2 above holds. Again, it is possible that sound query q_l may just be Void, and the complete query q_u be Any, indicating that the extent of n cannot be specified even partially, in which case the queries can be omitted from the transformation. Note that from a primitive transformation contractNode(n,q_l,q_u) used to transform $S' \rightarrow S$ we can automatically derive that extendNode(n,q_l,q_u) transforms $S \rightarrow S'$, and *vice versa*.

The last type of transformation dealing with nodes is $\textsf{renameNode}(n, n')$ causes a node n in a schema S to be renamed to n' in a new schema S', where in logical terms

$$\forall I. Ext_{S,I}(n) = Ext_{S,I}(n') \tag{3}$$

Note that this definition implies that from $\textsf{renameNode}(n, n')$ used to transform $S \rightarrow S'$ we can automatically derive that $\textsf{renameNode}(n', n)$ transforms $S' \rightarrow S$, and *vice versa*.

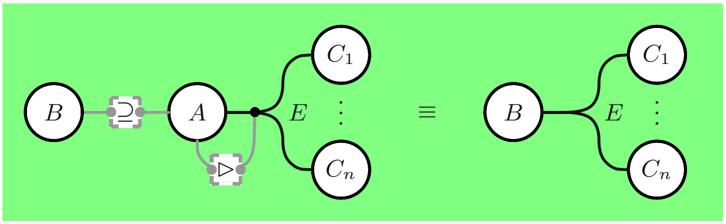


Fig. 7. Equivalence Relationships: Inclusion Merge

Entirely analogous arguments will give the definitions of the primitive transformations in Table 2 that handle manipulation of edges. For constraints, there are three differences. Firstly, since constraints have no extent, there is no query in the primitive transformations handling constraints. Secondly, constraints have no name, and hence there is no renameCons transformation. Thirdly, the extendCons transformation causes S' to be more restrictive than S, and hence the information capacity of S' is less than that of S.

Note that the ORM HDM schema in Figure 6(b) is the same as the ER HDM schema in Figure 3(b), except for trivial renaming of constructs. Hence we can apply equivalence preserving rename transformations to make the ORM HDM schema match those in the ER HDM schema:

renameEdge($\langle\langle$result:has:exam,student,course$\rangle\rangle$, $\langle\langle$result,student,course$\rangle\rangle$)

Hence the pathway of transformations describing the transformation from ER to relational schemas in the following sub-sections also defines the mapping from relational to ORM schemas, with this extra transformation step appended to the pathway.

We now move on to describe in the subsequent subsections the five mappings that we propose as a solution to transforming between the four models detailed in the previous section.

3.2 Inclusion Merge

The **Inclusion Merge** equivalence in Figure 7 allows us to merge two nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ together where $\langle\langle A\rangle\rangle$ is a subset of $\langle\langle B\rangle\rangle$ and there is a mandatory constraint from $\langle\langle A\rangle\rangle$ to an edge $e = \langle\langle E, A, C_1, \ldots, C_n\rangle\rangle$. The mandatory constraint is dropped as we merge $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$. Any edges or constraints that applied to $\langle\langle B\rangle\rangle$ remain, and any other (unillustrated) edges on $\langle\langle A\rangle\rangle$ are also redirected to $\langle\langle B\rangle\rangle$. Definition 9 gives a pseudo code definition of this equivalence, that generates primitive transformations on the HDM. The pseudo code first deletes the constraint between $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$, and then checks that the subset node $\langle\langle A\rangle\rangle$ is not associated in any other subset, union or exclusion with any other nodes, and raises an exception if that is the case. The pseudo code then iterates over all edges that connect to node $\langle\langle A\rangle\rangle$ and removes any mandatory constraints involving $\langle\langle A\rangle\rangle$. Then move_dependents function (defined in Definition 10) is used to move all edges on $\langle\langle A\rangle\rangle$ to connect to $\langle\langle B\rangle\rangle$. Note that this will include e and cause a new edge $e' = \langle\langle E, B, C_1, \ldots, C_n\rangle\rangle$ to now exist. The final line of Definition 9 then deletes $\langle\langle A\rangle\rangle$, giving a query that can restore the values of $\langle\langle A\rangle\rangle$ from the new non-mandatory edge e'.

Definition 9 Inclusion Merge

```
inclusion_merge($\langle\langle B\rangle\rangle$,$\langle\langle E, A, C_1, \ldots, C_n\rangle\rangle$)
    deleteCons($\langle\langle A\rangle\rangle \subseteq \langle\langle B\rangle\rangle$);
    if $op(\langle\langle A\rangle\rangle, d) \in Cons \wedge op \in \{\subseteq, \not\cap, \cup\}$ then
        exception
    endif;
    foreach $e \in Edges$ forwhich $e = \langle\langle E_a, A, \ldots\rangle\rangle$
        deleteCons($\langle\langle A\rangle\rangle \rhd e$)
    endforeach;
    move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$,id $\langle\langle A\rangle\rangle$);
    deleteNode($\langle\langle A\rangle\rangle$,$\{\langle x\rangle \mid \langle x, y_1, \ldots, y_n\rangle \in \langle\langle E, B, C_1, \ldots, C_n\rangle\rangle\}$);    □
```

The last line of inclusion_merge projects out the single arity tuples $\langle x\rangle$ that form the extension of $\langle\langle A\rangle\rangle$ from edge $\langle\langle E, B, C_1, \ldots, C_n\rangle\rangle$.

The definition of move_dependents takes three arguments, the first two (a, b) of which must be a node or edge, and the third a mapping set that maps instances of a to instances of b. For use in inclusion merge, the third argument should be an identity function id, defined as $id(\langle\langle A\rangle\rangle) = \{\langle a, a\rangle \mid a \in \langle\langle A\rangle\rangle\}$.

Definition 10 Move Dependents

```
move_dependents($a$,$b$,$map$)
    foreach $op(a, d) \in Cons$ forwhich $b \neq d$
        addCons($op(b, d)$);
        deleteCons($op(a, d)$)
    endforeach;
    foreach $op(d, a) \in Cons$ forwhich $b \neq d$
        addCons($op(d, b)$);
        deleteCons($op(d, a)$)
    endforeach;
    foreach $e \in Edges$ forwhich $e = \langle\langle E, a, C_1, \ldots, C_n\rangle\rangle$
        let $e'$=$\langle\langle E, b, C_1, \ldots, C_n\rangle\rangle$;
        addEdge($e'$,$\{\langle y, z_1, \ldots, z_n\rangle \mid \langle x, z_1, \ldots, z_n\rangle \in e \wedge \langle x, y\rangle \in map\}$);
        move_dependents($e$,$e'$,
            $\{\langle\langle x, z_1, \ldots, z_n\rangle, \langle y, z_1, \ldots, z_n\rangle\rangle \mid \langle x, y\rangle \in map \wedge \langle x, z_1, \ldots, z_n\rangle \in e\}$);
        deleteEdge($e$,$\{\langle x, z_1, \ldots, z_n\rangle \mid \langle y, z_1, \ldots, z_n\rangle \in e' \wedge \langle x, y\rangle \in map\}$)
    endforeach;    □
```

In Example 3, the series of transformations that will convert the HDM schema in Figure 4(b) into that in Figure 3(b) are listed. The first two steps in the series are applications of inclusion merge, which after ② result in the intermediate HDM schema shown in Figure 8.

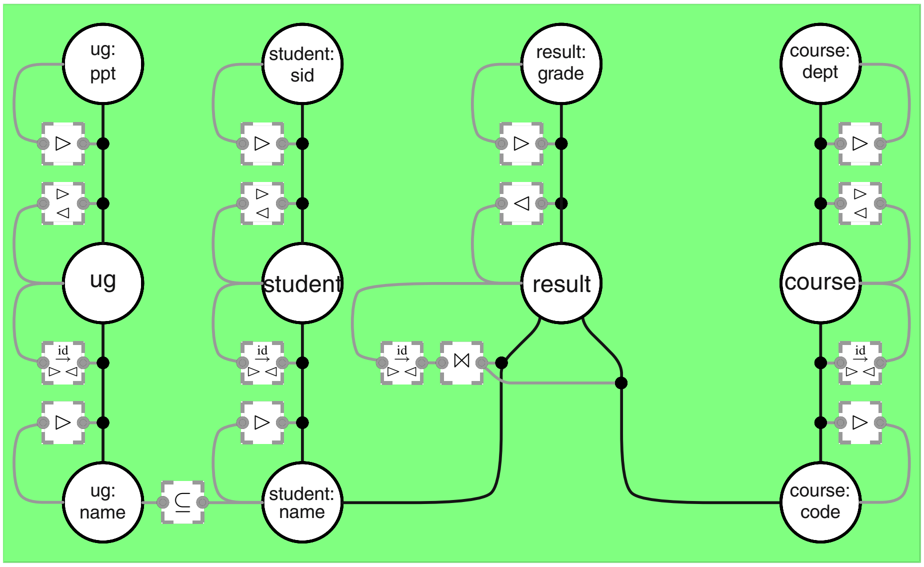


Fig. 8. Intermediate HDM schema in relational to ER conversion, after steps ① and ②

Example 3 Transforming between relational and ER HDM schemas

① inclusion_merge(⟨⟨student:name⟩⟩, ⟨⟨_,result:name,result⟩⟩)
② inclusion_merge(⟨⟨course:code⟩⟩, ⟨⟨_,result:code,result⟩⟩)
③ identity_node_merge(⟨⟨_,ug:name,ug⟩⟩)
④ unique_mandatory_redirection(⟨⟨_,student:name,result⟩⟩,
 ⟨⟨_,student:name,student⟩⟩)
⑤ unique_mandatory_redirection(⟨⟨_,course:code,result⟩⟩,
 ⟨⟨_,course:code,course⟩⟩)
⑥ identity_edge_merge(⟨⟨_,result,student⟩⟩, ⟨⟨_,result,course⟩⟩) □
⑦ move_dependents(⟨⟨student:name⟩⟩, ⟨⟨student⟩⟩, ⟨⟨_,student:name,student⟩⟩)

Taking transformation step ① and applying Definition 9, we may expand the steps into a series of primitive transformation steps shown in Example 4. Step (1.1) is a result of the first foreach loop in Definition 9, Steps (1.2) and (1.3) result from the call to move_dependents, and (1.4) and (1.5) result from the last two lines of Definition 9.

Example 4 Primitive steps associated with transformation ①

(1.1) deleteCons(⟨⟨result:name⟩⟩ $\triangleright$ ⟨⟨_,result:name,result⟩⟩)
(1.2) addEdge(⟨⟨_,student:name,result⟩⟩,
 $\{\langle b, c\rangle \mid \langle a, c\rangle \in$ ⟨⟨_,result:name,result⟩⟩ $\wedge \langle a, b\rangle \in$ id⟨⟨result:name⟩⟩$\}$
(1.3) deleteEdge(⟨⟨_,result:name,result⟩⟩,
 $\{\langle a, c\rangle \mid \langle b, c\rangle \in$ ⟨⟨_,student:name,result⟩⟩ $\wedge \langle a, b\rangle \in$ id⟨⟨result:name⟩⟩$\}$
(1.4) deleteCons(⟨⟨result:name⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩)
(1.5) deleteNode(⟨⟨result:name⟩⟩,
 $\{\langle b\rangle \mid \langle b, c\rangle \in$ ⟨⟨_,student:name,result⟩⟩$\}$ □

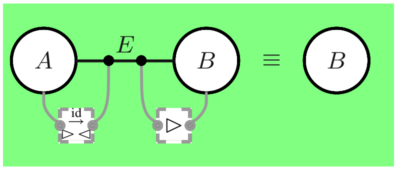


Fig. 9. Equivalence Relationships: Identity Node Merge

The expansion of transformations ① illustrates that inclusion merge is a data preserving transformation, since it only uses add and delete BAV transformations. In particular the edge ⟨⟨_,result,student:name⟩⟩ may be recovered by the query in (1.3) (which in turn uses the query of (1.5) to find the extent of ⟨⟨result:name⟩⟩), and the new edge ⟨⟨_,result,student:name⟩⟩ may be derived from existing data in (1.2). Note that a very similar expansion into primitive steps may be performed for ②, with a similar argument about data preservation.

3.3 Identity Node Merge

The **Identity Node Merge** in Figure 9 allows us to merge the two nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ together because they are identical. The constraints $\langle\langle A\rangle\rangle \xrightarrow{\text{id}} \langle\langle E,A,B\rangle\rangle$, $\langle\langle A\rangle\rangle \rhd \langle\langle E,A,B\rangle\rangle$, and $\langle\langle A\rangle\rangle \lhd \langle\langle E,A,B\rangle\rangle$ taken together mean that every instance of the edge $\langle\langle E,A,B\rangle\rangle$ is an identity mapping for $\langle\langle A\rangle\rangle$, and there is exactly one such mapping in $\langle\langle E,A,B\rangle\rangle$ for every element of $\langle\langle A\rangle\rangle$. As each element in $\langle\langle E,A,B\rangle\rangle$ is an identity mapping, each element in $\langle\langle A\rangle\rangle$ must be in $\langle\langle B\rangle\rangle$. Conversely because we also have $\langle\langle B\rangle\rangle \rhd \langle\langle E,A,B\rangle\rangle$, each element in $\langle\langle B\rangle\rangle$ must be in $\langle\langle A\rangle\rangle$, and so $\langle\langle A\rangle\rangle = \langle\langle B\rangle\rangle$. This implies both $\langle\langle B\rangle\rangle \xrightarrow{\text{id}} \langle\langle E,A,B\rangle\rangle$ and $\langle\langle B\rangle\rangle \lhd \langle\langle E,A,B\rangle\rangle$, and thus the equivalences illustrated in Figure 2(a) hold. Because we have identified them as equal, nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ can be merged together and the edge $\langle\langle E,A,B\rangle\rangle$ dropped, using Definition 11. Note any node can have this transformation applied in reverse, copying instances into a new node and linking the old node to the new node via an edge containing the identity instances.

Definition 11 Identity Node Merge

```
identity_node_merge($\langle\langle E,A,B\rangle\rangle$)
    let $e$=$\langle\langle E,A,B\rangle\rangle$;
    move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$,$e$);
    foreach $c \in Cons$ forwhich contains($e$,$c$)
        deleteCons($c$);
    endforeach;
    deleteEdge($e$,$\{\langle x,x\rangle \mid \langle x\rangle \in \langle\langle B\rangle\rangle\}$);
    deleteNode($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$);                □
```

This identity mapping comes about by the way some modelling languages specify a certain attribute as being an entity's identifying attribute (such as the primary key constraint in the relational schema).

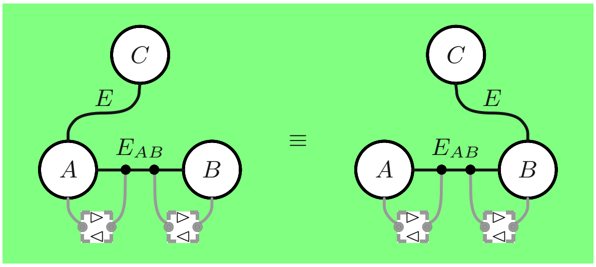


Fig. 10. Equivalence Relationships: Unique-Mandatory Redirection

In Figure 8 we can use identity node merge to merge nodes ⟪ug:name⟫ and ⟪ug⟫ by step ③ in Example 3. Note that the constraint ⟪ug:name⟫ $\subseteq$ ⟪student:name⟫ is not lost, but becomes ⟪ug⟫ $\subseteq$ ⟪student:name⟫. Figure 11 is partially derived by applying this merge.

3.4 Unique-Mandatory Redirection

The **Unique-Mandatory Redirection** equivalence in Figure 10 allows us to move an edge $\langle\langle E, A, C_1, \ldots, C_n \rangle\rangle$ from node $\langle\langle A \rangle\rangle$ to node $\langle\langle B \rangle\rangle$ because both $\langle\langle A \rangle\rangle$ and $\langle\langle B \rangle\rangle$ have a unique and mandatory constraint on the common edge $\langle\langle E_{AB}, A, B \rangle\rangle$. These constraints together are equivalent to stating that there is a one to one correspondence between the elements of $\langle\langle A \rangle\rangle$ and $\langle\langle B \rangle\rangle$ so whatever is related to an element of $\langle\langle A \rangle\rangle$ through $\langle\langle E, A, C_1, \ldots, C_n \rangle\rangle$ is equally related to the corresponding element in $\langle\langle B \rangle\rangle$. Moving the edge requires us to rewrite the elements of the edge, replacing in each the value that came from $\langle\langle A \rangle\rangle$ with the corresponding value from $\langle\langle B \rangle\rangle$ (via $\langle\langle E_{AB}, A, B \rangle\rangle$).

Definition 12 Unique-Mandatory Redirection

```
unique_mandatory_redirection($\langle\langle E, A, C_1, \ldots, C_n \rangle\rangle$,$\langle\langle E_{AB}, A, B \rangle\rangle$)
    let $e = \langle\langle E, A, C_1, \ldots, C_n \rangle\rangle$;
    let $map = \langle\langle E_{AB}, A, B \rangle\rangle$;
    if $(A \xrightarrow{\text{id}} e) \in Cons$ then exception endif;
    let $e' = \langle\langle E, B, C_1, \ldots, C_n \rangle\rangle$;
    addEdge($e'$,$\{\langle y, z_1, \ldots, z_n \rangle \mid \langle x, z_1, \ldots, z_n \rangle \in e \wedge \langle x, y \rangle \in map\}$);
    move_dependents($e$,$e'$,$map$)
    deleteEdge($e$,$\{\langle x, z_1, \ldots, z_n \rangle \mid \langle y, z_1, \ldots, z_n \rangle \in e' \wedge \langle x, y \rangle \in map\}$);    □
```

For the HDM schema in Figure 8, we can apply ④ in Example 3 to move the edge ⟪_,result,student:name⟫ from node ⟪student:name⟫ to node ⟪student⟫, becoming edge ⟪_,result,student⟫. This transformation does not lose information, because of the constraints on the edge ⟪_,student:name,student⟫, in particular ⟪student:name⟫ $\lhd$ ⟪_,student,student:name⟫ is implied by the other constraints present on the edge, as illustrated in Figure 2(a). Similarly we can apply ⑤ to move edge ⟪_,result, course:text⟫ to become ⟪_,result,course⟫. Applying these two edge redirections in addition to the previous identity node merge results in Figure 11.

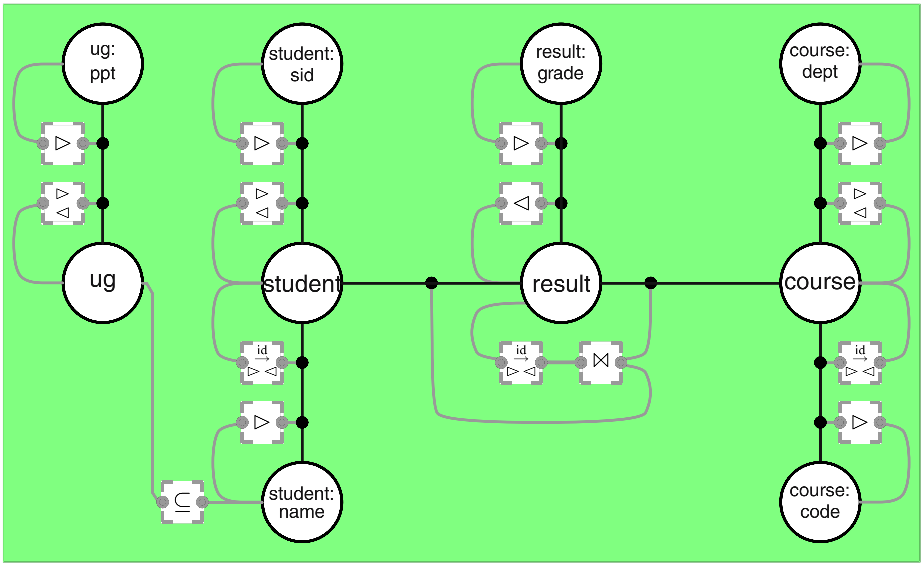


Fig. 11. Intermediate HDM schema in relational to ER conversion, after steps ①–⑤

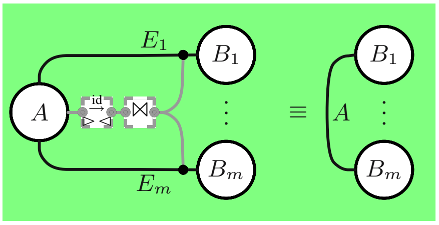


Fig. 12. Equivalence Relationships: Identity Edge Merge

3.5 Identity Edge Merge

The **Identity Edge Merge** in Figure 12 allows us to replace the node $\langle\langle A\rangle\rangle$ and edges $\langle\langle E_1, A, B_1\rangle\rangle \dots \langle\langle E_m, A, B_m\rangle\rangle$ with the single edge $\langle\langle A, B_1 \dots B_m\rangle\rangle$. The constraints $\overset{\text{id}}{\rightarrow}$, $\rhd$, and $\lhd$ between $\langle\langle A\rangle\rangle$ and the natural join of $\langle\langle E_1, A, B_1\rangle\rangle \dots \langle\langle E_m, A, B_m\rangle\rangle$ mean that for each instance of node $\langle\langle A\rangle\rangle$ there is exactly one instance of the join of edges $\langle\langle E_1, A, B_1\rangle\rangle \dots \langle\langle E_m, A, B_m\rangle\rangle$. The extent of the hyper edge $\langle\langle A, B_1 \dots B_m\rangle\rangle$ is obtained from the corresponding values in $\langle\langle B_1\rangle\rangle \dots \langle\langle B_m\rangle\rangle$ for each instance of the node $\langle\langle A\rangle\rangle$. Because of the identity mapping, these are the same values was in $\langle\langle A\rangle\rangle$, and hence there is no information in the node $\langle\langle A\rangle\rangle$ that is not in this new edge.

Definition 13 Identity Edge Merge

```
identity_edge_merge($\langle\langle E_1, A, B_1\rangle\rangle, \dots, \langle\langle E_m, A, B_m\rangle\rangle$)
    let $a = \langle\langle A, B_1, \dots, B_m\rangle\rangle$;
    addEdge($a, \{\langle b_1, \ldots, b_m\rangle \mid$
                $\langle a, b_1\rangle \in \langle\langle E_1, A, B_1\rangle\rangle \wedge \ldots \wedge \langle a, b_m\rangle \in \langle\langle E_m, A, B_m\rangle\rangle\}$);
    foreach $(A\ op\ e) \in Cons$ forwhich $e \in \{\langle\langle E_1, A, B_1\rangle\rangle, \ldots, \langle\langle E_m, A, B_m\rangle\rangle\}$
        deleteCons($A\ op\ e$)
    endforeach;
    move_dependents($\langle\langle A\rangle\rangle$,$a$,id $\langle\langle$A$\rangle\rangle$);
    deleteNode($\langle\langle A\rangle\rangle$,$a$)                                            □
```

In Figure 11 we can use identity node merge to replace the node ⟨⟨result⟩⟩ with the edge ⟨⟨result,student,course⟩⟩, in step ⑥ of Example 3. In this case the new edge is binary because the natural join was between two edges. Note that as part of this process, the edge ⟨⟨_,result,result:grade⟩⟩ from ⟨⟨result⟩⟩ to ⟨⟨result:grade⟩⟩ becomes ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩.

All that is left for us to do in order to obtain the HDM ER schema is to move the constraint ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩ to ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student⟩⟩. This is correct to do for similar reasons to the unique-mandatory redirection being correct for edges, but here we are moving a constraint between two nodes for which, in addition, we know the extent to be identical. This redirection is achieved by using the move_dependents subroutine in ⑦, and the result is Figure 3(b).

3.6 Node Reidentify

Object orientation introduces the concept of there being a unique **object identifier** (**OID**) that is associated to instances of a class, and that OID is not represented as an attribute. Thus when we look at the HDM representation of the UML shown in Figure 5, although similar to those for the relational, ER and ORM schemas, there is no use of the $\overset{\text{id}}{\rightarrow}$ constraint made between nodes representing the UML class, such as ⟨⟨student⟩⟩, and edges to nodes representing UML attributes, such as ⟨⟨_,student,student:name⟩⟩. This is because ⟨⟨student⟩⟩ has as its extent the object identifiers of the student UML class, whilst ⟨⟨student:name⟩⟩ has as its extent the names of students.

Definition 14 Node Reidentify

```
node_reidentify($\langle\langle A\rangle\rangle$,$map$)
    addNode($\langle\langle A'\rangle\rangle$,$\{\langle b\rangle \mid \langle a\rangle \in \langle\langle A\rangle\rangle \wedge \langle a, b\rangle \in map\}$);
    foreach $(\langle\langle A_s\rangle\rangle \subseteq \langle\langle A\rangle\rangle) \in Cons$
        node_reidentify($\langle\langle A_s\rangle\rangle$,$map$)
    endforeach
    move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle A'\rangle\rangle$,$map$);
    deleteNode($\langle\langle A\rangle\rangle$,$\{\langle a\rangle \mid \langle b\rangle \in \langle\langle A'\rangle\rangle \wedge \langle a, b\rangle \in map\}$)
    renameNode($\langle\langle A'\rangle\rangle$,$\langle\langle A\rangle\rangle$)                                    □
```

Example 5 Transforming between UML and ER HDM schemas
⑧ extendCons(⟨⟨student:name⟩⟩ $\lhd$ ⟨⟨_,student,student:name⟩⟩)
⑨ inverse_identity_node_merge(⟨⟨student⟩⟩, ⟨⟨student:oid⟩⟩)
⑩ deleteCons(⟨⟨student⟩⟩ $\overset{\text{id}}{\rightarrow}$ ⟨⟨_,student,student:oid⟩⟩)

⑪ $\mathsf{node\_reidentify}(\langle\langle \mathsf{student}\rangle\rangle, \{\langle x, y\rangle \mid$
$\quad\langle o, x\rangle \in \langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle \wedge \langle o, y\rangle \in \langle\langle \_, \mathsf{student}, \mathsf{student{:}name}\rangle\rangle\})$
⑫ $\mathsf{addCons}(\langle\langle \mathsf{student}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{student}, \mathsf{student{:}name}\rangle\rangle)$
⑬ $\mathsf{extendCons}(\langle\langle \mathsf{course{:}code}\rangle\rangle \lhd \langle\langle \_, \mathsf{course}, \mathsf{course{:}code}\rangle\rangle)$
⑭ $\mathsf{inverse\_identity\_node\_merge}(\langle\langle \mathsf{course}\rangle\rangle, \langle\langle \mathsf{course{:}oid}\rangle\rangle)$
⑮ $\mathsf{deleteCons}(\langle\langle \mathsf{course}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
⑯ $\mathsf{node\_reidentify}(\langle\langle \mathsf{course}\rangle\rangle, \{\langle x, y\rangle \mid$
$\quad\langle o, x\rangle \in \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle \wedge \langle o, y\rangle \in \langle\langle \_, \mathsf{course}, \mathsf{course{:}code}\rangle\rangle\})$
⑰ $\mathsf{addCons}(\langle\langle \mathsf{course}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{course}, \mathsf{course{:}code}\rangle\rangle)$
⑱ $\mathsf{renameEdge}(\langle\langle \mathsf{{:}has{:}exam}, \mathsf{student}, \mathsf{course}\rangle\rangle, \langle\langle \mathsf{result}, \mathsf{student}, \mathsf{course}\rangle\rangle)$
⑲ $\mathsf{deleteCons}(\langle\langle \mathsf{course}\rangle\rangle \lhd \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
⑳ $\mathsf{deleteCons}(\langle\langle \mathsf{course}\rangle\rangle \rhd \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
㉑ $\mathsf{deleteCons}(\langle\langle \mathsf{course{:}oid}\rangle\rangle \lhd \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
㉒ $\mathsf{deleteCons}(\langle\langle \mathsf{course{:}oid}\rangle\rangle \rhd \langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
㉓ $\mathsf{contractEdge}(\langle\langle \_, \mathsf{course}, \mathsf{course{:}oid}\rangle\rangle)$
㉔ $\mathsf{contractNode}(\langle\langle \mathsf{course{:}oid}\rangle\rangle)$
㉕ $\mathsf{deleteCons}(\langle\langle \mathsf{student}\rangle\rangle \lhd \langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle)$
㉖ $\mathsf{deleteCons}(\langle\langle \mathsf{student}\rangle\rangle \rhd \langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle)$
㉗ $\mathsf{deleteCons}(\langle\langle \mathsf{student{:}oid}\rangle\rangle \lhd \langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle)$
㉘ $\mathsf{deleteCons}(\langle\langle \mathsf{student{:}oid}\rangle\rangle \rhd \langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle)$
㉙ $\mathsf{contractEdge}(\langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle)$
㉚ $\mathsf{contractNode}(\langle\langle \mathsf{student{:}oid}\rangle\rangle)$

□

When transforming between between an OO model such as UML, and key based models such as ORM, ER or relational, we must overcome the fundamental difference in data modelling based on OIDs and natural keys. This will require us finding attributes or associations of the UML class that can be used to identify instances of the UML class.

Comparing the UML schema in Figure 5 with the ER schema in Figure 3, the HDM schemas of the two appear similar. One difference is trivial: the edge between $\langle\langle \mathsf{student}\rangle\rangle$ and $\langle\langle \mathsf{course}\rangle\rangle$ has a different name in the two schemas. The other difference is between the use of OIDs and natural keys. The ER HDM schema, using natural keys, has $\langle\langle \mathsf{student}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{student}, \mathsf{student{:}name}\rangle\rangle$ and $\langle\langle \mathsf{course}\rangle\rangle \xrightarrow{\text{id}} \langle\langle \_, \mathsf{course}, \mathsf{course{:}code}\rangle\rangle$, whereas the UML HDM, using OIDs, does not have these constraints. Example 5 lists a sequence of transformations that converts the UML schema into an 'ER compatible' HDM schema that has explicit attributes for the OIDs, and uses a natural key to identify the ER entity instances. The following steps explain the example:

1. Missing from the UML schema is any definition of natural keys for the UML classes. Hence step ⑧ introduces a new constraint that indicates that name is a candidate key for student.
2. The inverse of identity node merge in step ⑨ generates a new node $\langle\langle \mathsf{student{:}oid}\rangle\rangle$, connected to $\langle\langle \mathsf{student}\rangle\rangle$ by a new edge $\langle\langle \_, \mathsf{student}, \mathsf{student{:}oid}\rangle\rangle$. If for example

the node ⟨⟨student⟩⟩ had the extent {⟨&1⟩, ⟨&2⟩, ⟨&3⟩, ⟨&4⟩} before this step, then the new edge will have as its extent
{⟨&1, &1⟩, ⟨&2, &2⟩, ⟨&3, &3⟩, ⟨&4, &4⟩}.

3. Transformations ⑩–⑫ have the net effect of repopulating the ⟨⟨student⟩⟩ node with values of the ⟨⟨student:name⟩⟩ attribute, and changing its key from oid to name. For example, if in the schema that results from ⑨ ⟨⟨_,student, student:name⟩⟩ had extent
   {⟨&1, 'Mary'⟩, ⟨&2, 'John'⟩, ⟨&3, 'Jane'⟩, ⟨&4, 'Fred'⟩}
   then the *map* generated would be the same list, and in the schema after ⑪, the node ⟨⟨student⟩⟩ would have the extent {⟨'Mary'⟩, ⟨'John'⟩, ⟨'Jane'⟩, ⟨'Fred'⟩}, and the edge ⟨⟨_,student,student:oid⟩⟩ would have the extent
   {⟨'Mary', &1⟩, ⟨'John', &2⟩, ⟨'Jane', &3⟩, ⟨'Fred', &4⟩}
   The result of after ⑫ is shown in Figure 13.
4. Transformations ⑬–⑰ perform a similar conversion of the ⟨⟨course⟩⟩ node into a natural key based construct, using code as the key.
5. Transformation ⑱ deals with the trivial problem of renaming the edge between ⟨⟨student⟩⟩ and ⟨⟨course⟩⟩ to match the name in the ER schema.
6. Transformations ⑲–㉚ delete the ⟨⟨student:oid⟩⟩ and ⟨⟨course:oid⟩⟩ nodes (with their associated constraints and edges). Note that the use of contract transformations represents the fact that you are unable to derive the oid values from the ER schema.

If the transformations in Example 5 are compared with the BAV transformation summary Table 2, we see that the UML schema has higher information capacity than the ER schema, due to its use of OIDs and lack of key constraints. Note that alternatively, step (6) could be omitted, and the ER schema could be enhanced with oid attributes, if it was intended to use the ER schema to more fully represent the UML schema. However, the UML would still lack the key information present in the ER schema.

3.7 Non-equivalent Schemas

The examples in Figures 3–6 were deliberately chosen to illustrate how we could draw an equivalence between schemas with the same information capacity (with the exception of the UML object identifiers and the other models use of keys). In practice, modelling languages have different expressive powers, and hence there may be no equivalent schema.

For example, changing the cardinality constraint in Figure 3(a) of student being associated with result from 0:N to 1:N would result the addition in Figure 3(b) of a HDM constraint ⟨⟨student⟩⟩ $\triangleright$ ⟨⟨result,student,course⟩⟩. If we review the arguments outlined in Sections 3.2–3.5 with this extra constraint in place then we would run into a problem. The reversed edge redirection from ⟨⟨_,result,student⟩⟩ in Figure 11 to ⟨⟨_,result, student:name⟩⟩ in Figure 8 carries the mandatory constraint introduced by 1:N, giving an HDM constraint ⟨⟨student:name⟩⟩ $\triangleright$ ⟨⟨result,student,course⟩⟩. When we come to reverse the inclusion merge that merged ⟨⟨result:name⟩⟩ into ⟨⟨student:name⟩⟩ to enable the relationship between ⟨⟨result⟩⟩ and ⟨⟨student:name⟩⟩ to be represented as a foreign key, we are unable to carry this mandatory constraint down to ⟨⟨result:name⟩⟩. This is because the relational schema in Figure 4(a) cannot be altered to express the fact

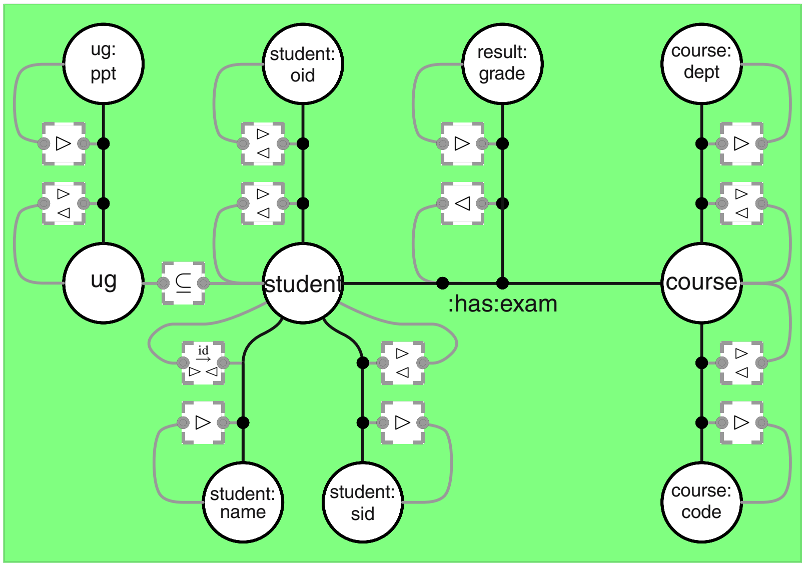


Fig. 13. UML to ER mapping after ⑫

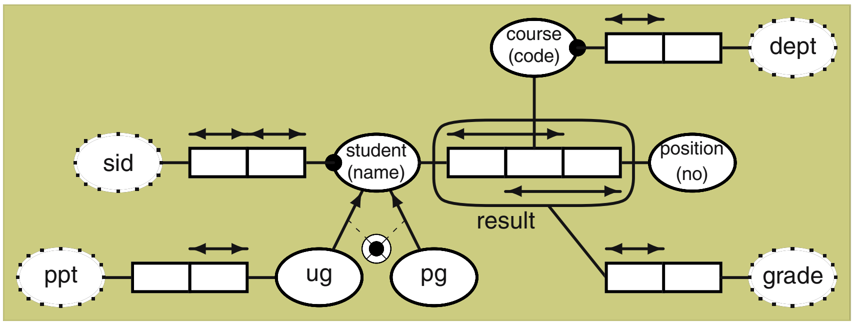
that every student.name must be referenced by at least one result.name. This lost constraint is, therefore, not a weakness in the approach, but an example of the approach formally identifying what information from the ER schema cannot be represented in the relational schema. In this particular case, it might appear that we could repair the relational schema by adding the foreign key constraint student.name → result.name, but this would not be legal since result.name is not a candidate key of result.

4 Handling Additional Modelling Concepts

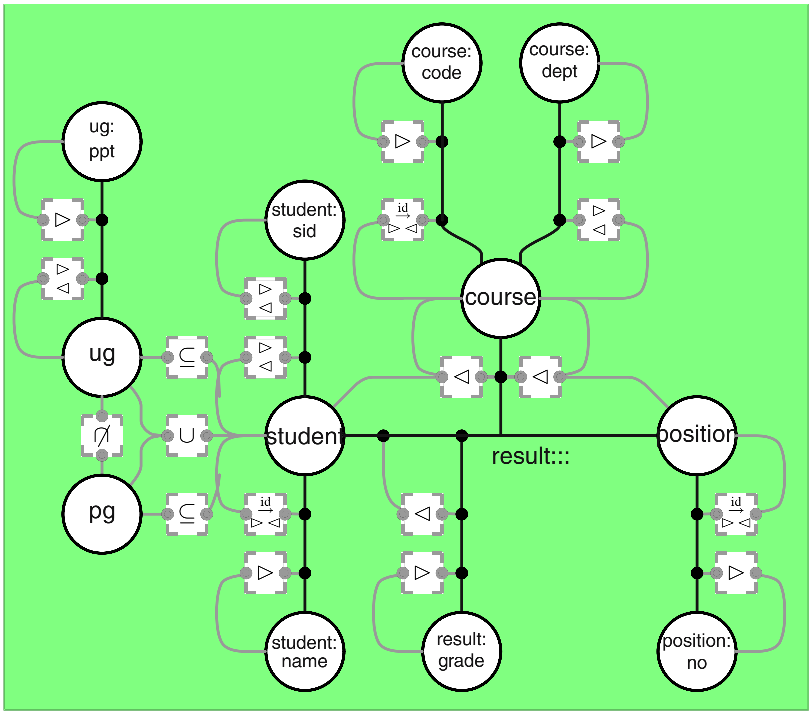
Figure 14(a) illustrates an ORM schema of an extended version of the student-course database where, as we will show, the ORM schema is able to represent some aspect of the UoD that one or more of our other three data models is unable to represent. The additions made to the ORM schema of Figure 6 are described in the following paragraphs under headings which indicate the category of modelling concept they fall under, and we discuss the extent to which the relational, ER and UML models may handle these types of concepts.

Candidate Keys. The ORM model of Figure 14(a) has reference/predicate between value ⟨⟨sid⟩⟩ and entity ⟨⟨student⟩⟩ where both roles are key. This implies that we can identify ⟨⟨student⟩⟩ by either ⟨⟨name⟩⟩ or by ⟨⟨sid⟩⟩. This concept can be represented in a relational schema with the ⟨⟨student,sid⟩⟩ attribute being a candidate key. Neither the ER nor UML models have a method of representing this concept however.

Typically, ER models do not make explicit the relationships between an entity and its attributes, but instead use some sort of syntax with an attribute's name to indicate



(a) ORM schema of an extend student-course database



(b) HDM representation of the ORM schema

Fig. 14. An extended student-course database

that it is (or is part of) the primary identifier: no other uniqueness constraints can be expressed. With the more elaborate ER syntax where attribute/entity cardinality constraints are explicit, a one-to-one relationship is synonymous with the primary identifier and therefore can only be expressed once per entity.

Because of UML's reliance on object identifiers, it does not require classes to have value-based reference schemes and indeed requires nonstandard extensions to its notation to express an attribute's uniqueness in its association with its class.

Note that in our HDM production rules for UML there is no rule that generates a uniqueness constraint from an attribute to the edge associating it to its class. In our HDM production rules for ER the only way to generate this constraint is in conjunction with a reflexive constraint.

If we were to convert our extended ORM schema into an HDM schema that could have been produced by an equivalent ER schema, we would have to drop the uniqueness constraint from either ⟨⟨sid⟩⟩ or ⟨⟨student⟩⟩, or extend the ER language to handle candidate keys. For UML, both uniqueness constraints must be dropped, since it has no support for any keys (except by using its constraint language).

Disjointness between entities. The ORM model of Figure 14(a) has an additional subclass entity ⟨⟨pg⟩⟩ that is disjoint from ⟨⟨ug⟩⟩, and in addition, ⟨⟨pg⟩⟩ and ⟨⟨ug⟩⟩ are total w.r.t. entity ⟨⟨student⟩⟩.

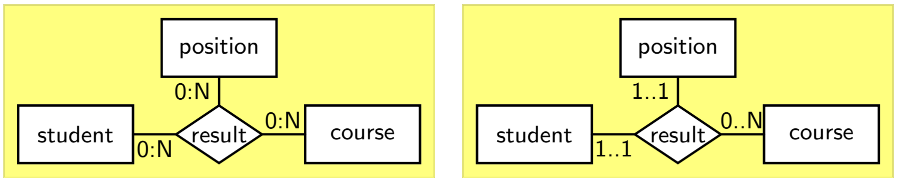
The disjointness and totality can not be represented in the relational model, since the relational model has no constructs that make use of either the HDM disjoint or union constraints. If we wanted to model our extended ORM example using a relational schema we would have to drop the exclusion constraint between ⟨⟨ug⟩⟩ and ⟨⟨pg⟩⟩ as well as the union constraint between these subsets and ⟨⟨student⟩⟩.

There are several well known ways we may attempt to model the total partition.

- We could represent each subset with a table containing the columns common to just that subset, and a primary key that is also a foreign key to the superclass table ⟨⟨student⟩⟩. The problem is that there is no way to enforce that at most one instance of some referring foreign key exists for each instance of a primary key, *i.e.* that the subclass tables are disjoint.
- With some more elaborate transformations we could change the subclass identification into a column of the superclass telling us which subclass table we should join each instance to. This would enforce the exclusion constraint, and if we made the column not nullable it would also enforce the union constraint. The problem is that the relational model has no way of specifying that a value in a column in the superclass table means that the superclass table joins with a particular subclass table.
- With yet more transformations we could use the subclass-identifying column as in (4) and make each column of each subtype a nullable column of the supertype table where it is set to null when not applicable, and have no subtype tables. Again, the relational model has no way of specifying that certain columns must, or must not, be null depending on the value held in another column.

Therefore, despite there being several ways to model subclasses in the relational model, we can not fully represent the exclusion and union constraints involving ⟨⟨ug⟩⟩ and ⟨⟨pg⟩⟩. The UML and ER modelling languages are able to represent these constraints.

Cardinality constraints of n-ary relationships. The ORM model of Figure 14(a) has a ternary ORM fact type between entities ⟨⟨student⟩⟩, ⟨⟨course⟩⟩, and ⟨⟨position⟩⟩. This fact type has two overlapping keys, the first states that any pair of ⟨⟨student⟩⟩



(a) Look here semantics (b) Look across semantics

Fig. 15. Alternative ER ternary-relationship cardinality constraints

and ⟨⟨course⟩⟩ instances may appear at most once in the fact, and the second that any pair of ⟨⟨position⟩⟩ and ⟨⟨course⟩⟩ instances may appear at most once in the fact.

Whilst it is well known that look-across and look-here cardinality constraints have equivalent expressive power for binary relationships, it is rather less well known that this equivalence does not extend to n-ary relationships [21]. In particular, the use of keys on the ORM ternary fact type is not representable in the ER modelling language as we defined it in Section 2.2, since we choose look-here cardinality constraints. Look-here constraints are restricted to express the cardinality of just a single entity in the relationship. The use of HDM unique constraints in Figure 14(b) has no representation as ER model look-here constraints. Hence the nearest representation of the ternary relationship we can achieve is shown in Figure 15(a), where all cardinality constraints are removed (*i.e.* 0:N is used, meaning that the relationship is unrestricted).

The UML model is capable of describing this concept, since it uses look-across cardinality constraints. Many ER modelling languages also use look-across semantics, and hence we could represent the ORM in such languages as shown in Figure 15(b), where the use of the symbol .. between lower and upper bounds for cardinality constraints indicates the use of look-across semantics[1].

To generalise, for **look-here** semantics, we have the restriction on the use of HDM mandatory and unique constraints from Definition 5 to $m = 1$ (*i.e.* only one node or edge is restricted as mandatory or unique), and for **look-across** semantics the restriction is that $m = n - 1$ for an n-ary relationship.

5 Related Work

Graphs and graph transformations are a subject of fundamental interest to computer science, and therefore have been very widely studied [48], and in particular have been studied with application to schema transformation between different modelling languages. What we will now do is set our work in the context of other work (and thus

[1] Whilst this example might indicate that look-across semantics have advantages, there will be other modelling situations that would be better modelled with look-here constraints. There are also operational issues to consider, in that look-across constraints on n-ary relationships are difficult to implement: for example, if mandatory were used in n-ary relationships, then inserting or deleting an instance from one entity will require more than one relationship instance to be inserted or deleted. However, since we are in this paper only studying the logical aspects of data models, we will not study this issue further.

demonstrate it to be distinctive) by discussing the answer to three general questions one might ask about work in the area of 'intermodel transformation'. Note that we exclude from the discussion issues associated with modelling languages that are not set oriented, and also the typing of data, since these matters are outside the scope of the work presented in this paper.

What is being modelled, and how is it modelled? The first distinction is between graphs which model the **dynamic behaviour** of a computer system (a recent survey may be found in [3], and general tools to model software have been constructed such as the **PROGRES** system [33]), and graphs which model the **static** (*i.e.* data) aspect of the system, in which area our work falls into. There is a degree of overlap between the two, in that programs must manipulate data to perform useful tasks (for example in the PROGRES system [34]).

Considering work that is focused on static data modelling via graphs, the next distinction we can make is between systems that are data **model specific**, or those which handle **multiple models**. This is not a binary distinction, but denotes a spectrum. At one extreme we find systems that map between schemas in a single modelling language: normally the relational model such as in the self-documenting data models of [26], or the initial version of Clio [50]. In the middle of the spectrum, much work has concentrated on converting between just a few specific modelling languages: for example between relational and ER [1,36], ORM to UML or relational [20], relational and generic object oriented models [11,23], XML and relational data [37], *etc*. Our work, along with [18,2,10,7,44], attempts to provide a more flexible framework, and demonstrate how the framework is adaptable to a wide range of data modelling languages.

Within approaches that deal with multiple models, there is then the distinction to make as to how the approaches handle the requirements of multiple data modelling languages. Invariably, some **common data model** (**CDM**) [43] is used as a intermediate language, into which schemas of all other data modelling languages are translated. There is then the choice of how expressive with CDM language should be [17]. Some approaches take a union of all modelling constructs to form a **high level** CDM which has all the features of the models being represented (for example, by having constructs for key, aggregation, function dependency, and so on). **DB-MAIN** [18,22,17] and **MDM** [2] are examples of tools that take this approach. Our work, and [10,7,44], instead use a **low level** CDM, where the CDM has a few simple modelling constructs, plus the ability to express some constraints over those modelling constructs. Our work differs from other approaches in having developed a small set of constraint primitives that may be used instead of general logical expressions. The constraint primitives are relatively fine grained, which has the advantage that our transformations can deal with just those aspects of the constraints that are relevant to the particular transformation.

What type of Graph Language is used? The notion of a 'graph' is a very general one, but we will restrict ourselves to saying that we are considering languages that have the concept of there being nodes, along with some method of defining relationships between those nodes. From that basis, there have been a wide range of variations of what semantics are attached to the nodes and relationships. A major distinction is between graph models that are **schema models** where data is just the values associated with the extent of the nodes and relationships. The HDM, and [26,2,44,18,10,16,46] fall into this category. Alternatively, the graph models may be **two-level models**, modelling as nodes and associations between nodes both the schema and the data instances. Examples of this approach include [7,23].

Another important feature to distinguish is what relationships are provided in the graph model. Using the terminology of the HDM, we can distinguish between relationships that model (1) edges (*i.e.* data that is restricted in extent to values that appear in the nodes that the edge connects), (2) constraints (*i.e.* just restrict the values that may appear in the nodes that the relationship connects) or (3) edge-constraints (*i.e.* are edges, with some implied constraints).

Perhaps surprisingly, some graph models have just nodes and constraints. For example, **MDM** has three types of node — abstract, aggregation and lexicals — and six types of constraints (which they call edges) that represent functional dependency, multivalued functional dependency, components of aggregation, keys of aggregation, and keys of abstract. Also, the **ULD** [7] representation language has nodes representing sets of tuples called 'constructs' which may have first order logic constraints placed on their extent, and **WOL** [16] models nodes as classes, with the notion of keys used to identify class instances, and general purpose constraint language to use over the class instances.

A common type of graph language is to have nodes and binary edges, such as in Clio, possibly with additional constraints, such as [26,10]. In [44] the **reserved graph grammar** (**RGG**) is used to represent data models, where an RGG is comprised of two types of node, and binary edges between these nodes, and some constraint relationships.

A hypergraph model was used in the specific case of relational schema transformations in [51], and nested hypergraphs have been studied as a general framework for modelling complex objects [38]. The **higher-order ER model** (**HERM**) [46] is similar to the HDM, in that it is also a nested hypergraph model: it allows for n-ary relationships which may connect to entities or other relationships. However, the HERM still distinguishes between attributes and entities, and it uses relatively course grained constraints. In particular, cardinality constraints are not decomposed into two primitives as they are in this paper, nor is there the equivalent of our reflexive constraint. Although there is some discussion in [46] about mapping HERM to relational and network models, we are not aware of other work that uses nested hypergraphs as a general basis for mapping between schemas in different modelling languages. An advantage of the HDM over other approaches is that it clearly separates those constructs that have an extent (nodes and edges) from those which restrict the extent of other constructs (constraints). Another advantage of the use of a hypergraph based model is that it has a natural mapping to all higher level modelling languages we have studied to date.

How are the transformations specified? The notion of transforming schemas is widely studied, and surveys of database transformations can be found in [4,15]. A distinction that can be drawn between most previous work and ours is the level of granularity at which the transformations between schemas are presented. Most work [18,26,44,16] takes a **coarse grain** approach, where a transformation will specify a semantic mapping between equivalent structures in the high level modelling language, such as the mapping between a many-many relationship in one schema to an entity with two one-many relationships. In our approach and in [10,6], the aim is to specify transformations at a **fine grain** level. Our approach differs in that our transformations are schema oriented, in the sense that we incrementally add, delete or rename single constructs in the HDM, rather than be query oriented as in [10,6] where they specify the query that is used to map sets of constructs in one model to sets of constructs in another. The use of BAV transformations has the advantage that it establishes a bidirectional mapping between schemas.

6 Acknowledgments, Summary and Future Work

The original HDM and its implementation in the AutoMed project was developed in collaboration between Imperial College and Birkbeck College, and the AutoMed project was funded by the EPSRC. We acknowledge the contribution of all the AutoMed project members to many discussions of the work presented in this paper, and also acknowledge the reviewers of this paper for many helpful suggestions and for detecting various errors in the early drafts of this paper.

In this paper we have extended the **hypergraph data model** (**HDM**) [39]. The HDM differs from normal graphs in that edges may connect together any number of nodes or edges rather than just two nodes. HDM nodes and edges have an extent, with the extent of an edge being constrained to take values that must appear in the extent of the nodes or other edges it connects.

The HDM in [39] allows arbitrary constraints. Our extension in this paper is to define six fundamental constraints that we believe cover the majority of features in the popular high level data modelling languages. We have used these to precisely define mappings between the HDM schema and a representative set of features from the relational, ER, ORM and UML-class data models.

We have taken an example UoD, and given schemas for that UoD in four high level data models named above. Then using our mappings we have derived an equivalent HDM representation for each. The similarity of each HDM graph with its high level data model's counterpart illustrates the fact that many of the semantics of these high level data models implied by their graphical structures are similar across the spectrum of data models, and this commonality is captured by the graphical structure of HDM. Although each HDM schema was shown to be equivalent (with the exception of some aspects of the UML model) in the sense that each has the same information capacity, they were syntactically different.

We reviewed the set of primitive reversible transformations on HDM graphs that we have used extensively elsewhere in our schema integration and evolution work [27,39,29,30]. We then gave five equivalence preserving graph transformations predicated on using only our six fundamental constraints, and defined in terms of these primitive HDM transformations.

We then showed how the three equivalent HDM graphs from the ER, relational and ORM schemas can be converted into each other through a sequence of these equivalence preserving transformations. The mappings from the high level data models to their HDM representations, along with the equivalence transformations between them show that these three schemas are equivalent, by virtue of the fact that only add, delete and rename transformations were required to implement the mapping. As the equivalence transformations are built from HDM's reversible primitive transformations, queries can be rewritten from one schema to another. In principle, we could therefore migrate data from an instance of a schema in one data modelling language to an instance of an equivalent schema in a different data modelling language.

We also discussed how our approach identifies when two HDM schemas are not equivalent by virtue of any 'left over' constraints from one schema when we try to convert it into the other schema (after transformations are applied, there are some constraints in one schema that do not appear in the other). This was used to show exactly how the UML schema from the example UoD differs from the other three schemas. We also outlined how our approach might be used to demonstrate that some features of a high level data modelling language may not be representable in another. Note this does not prove that the two schemas are not equivalent: to realise this goal we would need to prove that our equivalence transformations are adequate to convert between any two equivalent HDM schemas, and also prove that every feature of the data models in question are mapped into an equivalent set of HDM model constructs.

Our future work will expand our approach to take account of type information in a data source, and also to model list and bag based data models, such as XML (which has list based semantics) and SQL (*i.e.* the relational model with bag semantics). We will also investigate heuristic search techniques to determine automatically which equivalence preserving transformations are required to map a schema in one modelling language into a schema in another modelling language.

References

1. M. Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. In *Proc. ER'94*, LNCS, pages 403–419. Springer, 1994.
2. P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *Proc EDBT'96*, volume 1057 of *LNCS*, pages 79–95. Springer-Verlag, 1996.
3. L. Baresi and R. Heckel. Tutorial introduction to graph transofrmation: A software engineering perspective. In A. Corradini *et al*, editor, *Proc. ICGT*, volume 2505 of *LNCS*, pages 402–429. Springer-Verlag, 2002.
4. C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
5. P.A. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR'03*, 2003.
6. S. Bowers and L. Delcambre. On modeling conformance for flexible transformation over data models. In *Knowledge Transformation for the Semantic Web*, pages 34–48. IOS Press, 2003.
7. S. Bowers and L. Delcambre. The uni-level description: A uniform framework for representing information in multiple data models. In *Proc. ER'03*, volume 2813 of *LNCS*, pages 45–58. Springer-Verlag, 2003.
8. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 82–97. Springer-Verlag, 2004.
9. M. Boyd and P.J. McBrien. Towards a semi-automated approach to intermodel transformations. In *Proc. EMMSAD 04, CAiSE Workshop Proceedings Volume 1*, pages 175–188, 2004.
10. K.T. Claypool and E.A. Rundensteiner. Sangam: A framework for modeling heterogeneous database transformations. In *Proc. ICEIS 03*, pages 219–224, 2003.

11. C.J. Date. Object identifiers vs. relational keys. In *Relational Database: Selected Writings 1994–1997* [14], chapter 12, pages 457–476.
12. C.J. Date. *An Introduction to Database Systems*. Addison-Wesley, 8th edition edition, 2004.
13. C.J. Date, H. Darwen, and N.A. Lorentzos. *Temporal Data and the Relational Model*. Morgan Kaufmann, 2003.
14. C.J. Date, H. Darwen, and D. McGoveran. *Relational Database: Selected Writings 1994–1997*. Addison-Wesley, 1998.
15. S.B. Davidson, P. Buneman, and A.S. Kosky. Semantics of database transformations. In *Semantics in Databases*, LNCS, 1998.
16. S.B. Davidson and A.S. Kosky. WOL: A language for database transformations and constraints. In *Proc. ICDE97*, pages 55–65, 1997.
17. J-L. Hainaut. Transformation-based database engineerig. In *Transformation of Knowledge, Information, and Data* [48], chapter 1, pages 1–28.
18. J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, and D. Roland. Database evolution: the DB-MAIN approach. In *Proc. ER'94*, LNCS, pages 112–131. Springer, 1994.
19. P. Hall, J. Owlett, and S.J.P. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Data Base Management Systems*. North-Holland, 1975.
20. T. Halpin. *Information Modeling and Relational Databases*. Academic Press, 2001.
21. S. Hartmann. Reasoning about participation constraints and Chen's constraints. In *Proc. 14th Australasian database conference*, pages 105–113. Australian Computer Society, 2003.
22. J-M. Hick and J-L. Hainaut. Strategy for database application evolution: The DB-MAIN approach. In *Proc. ER'03*, volume 2813 of *LNCS*, pages 291–306. Springer-Verlag, 2003.
23. J.H. Jahnke and A. Zündorf. Apply graph transformations to database re-engineering. In *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 2, chapter 6. World Scientific, 1999.
24. E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL queries and migrating data in the AutoMed toolkit. Technical Report No. 20, AutoMed, 2003.
25. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.
26. L. Mark and N. Roussopoulos. Integration of data, schema and meta-schema. In *Proc. ER83*, pages 585–602, 1983.
27. P.J. McBrien and A. Poulovassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998.
28. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer, 1999.
29. P.J. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proc. CAiSE'02*, volume 2348 of *LNCS*, pages 484–499. Springer, 2002.
30. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.
31. P.J. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Proc. DBISP2P, at VLDB'03*, pages 91–107, 2003.
32. R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, 19(1):3–31, 1994.
33. M. Munch. Programmed graph rewriting system PROGRES. In *In Proc. AGTIVE'99*, volume 1779 of *LNCS*, pages 441–448. Springer-Verlag, 2000.
34. M. Munch, A. Schurr, and A.J. Winter. Integrity constraints in the multi-paradigm language progres. In H. Ehrig *et al*, editor, *Graph Transformation*, volume 1764 of *LNCS*, pages 338–352. Springer-Verlag, 2000.
35. S. Patig. Measuring expressiveness in conceptual modeling. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 127–141. Springer-Verlag, 2004.

36. J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proc. ICDE'96*, pages 218–227, 1996.
37. L. Popa, M.A. Hernandez, and Y. Velegrakis *et al*. Mapping XML and relational schemas with Clio. In *Proc. ICDE'02*, pages 498–499, 2002.
38. A. Poulovassilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Trans. on Information Systems*, 12(1):35–68, 1994.
39. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
40. N. Rizopoulos. Automatic discovery of semantic relationships between schema elements. In *Proc. of 6th ICEIS*, 2004.
41. N. Rizopoulos and P.J. McBrien. A general approach to the generation of conceptual model transformations. In *Proc. CAiSE'05*, volume 3520 of *LNCS*. Springer-Verlag, 2005.
42. K. Schewe. Design theory for advanced datamodels. In *Proc. 12th Australasian Conf. on Database Technologies*, pages 3–9, 2001.
43. A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.
44. G. Song, K. Zhang, and J. Kong. Model management through graph transformations. In *Proc. Visual Languages and Human-Centric Computing*, pages 75–82. IEEE, 2004.
45. I.Y. Song, M. Evans, and E.K. Park. A comparative analysis of entity-relationship diagrams. *Journal of Computer & Software Engineering*, 3(4):427–459, 1995.
46. B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer, 2000.
47. N. Tong. Database schema transformation optimisation techniques for the AutoMed system. In *Proc. BNCOD'03*, volume 2712 of *LNCS*, pages 157–171. Springer, 2003.
48. P. van Bommel. *Transformation of Knowledge, Information, and Data*. Idea Group, 2005.
49. R. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, 1998.
50. L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *Proc. SIGMOD'01*, pages 485–496, 2001.
51. C. Zaniolo and M. Melkanoff. A formal approach to the definition and the design of conceptual schemata for database systems. *ACM TODS*, 7(1):24–59, 1982.

iASA: Learning to Annotate the Semantic Web[*]

Jie Tang[**], Juanzi Li, Hongjun Lu, Bangyong Liang, Xiaotong Huang, and Kehong Wang

Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China
{j-tang02, liangby97}@mails.tsinghua.edu.cn
{ljz, x.huang, wkh}@keg.cs.tsinghua.edu.cn

Abstract. With the advent of the Semantic Web, there is a great need to upgrade existing web content to semantic web content. This can be accomplished through semantic annotations. Unfortunately, manual annotation is tedious, time consuming and error-prone. In this paper, we propose a tool, called iASA, that learns to automatically annotate web documents according to an ontology. iASA is based on the combination of information extraction (specifically, the Similarity-based Rule Learner—SRL) and machine learning techniques. Using linguistic knowledge and optimal dynamic window size, SRL produces annotation rules of better quality than comparable semantic annotation systems. Similarity-based learning efficiently reduces the search space by avoiding pseudo rule generalization. In the annotation phase, iASA exploits ontology knowledge to refine the annotation it proposes. Moreover, our annotation algorithm exploits machine learning methods to correctly select instances and to predict missing instances. Finally, iASA provides an explanation component that explains the nature of the learner and annotator to the user. Explanations can greatly help users understand the rule induction and annotation process, so that they can focus on correcting rules and annotations quickly. Experimental results show that iASA can reach high accuracy quickly.

1 Introduction

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [4, 5]. In recent years, semantic web has made significant progress, in particular through the development of infrastructure such as: ontology language like RDF and OWL, ontology editor like Protégé, and reasoning engine like Racer.

In order to provide semantic web with ‘understandable’ data, it is necessary to conduct annotation for at least two kinds of metadata. Specifically, commonly used ontologies for semantic web need to be created; and existing web contents need to be upgraded to semantic web content, i.e. semantic annotation. The later issue is exactly the problem addressed in this paper.

[*] Supported by the National Natural Science Foundation of China under Grant No. 60443002.
[**] Corresponding author.

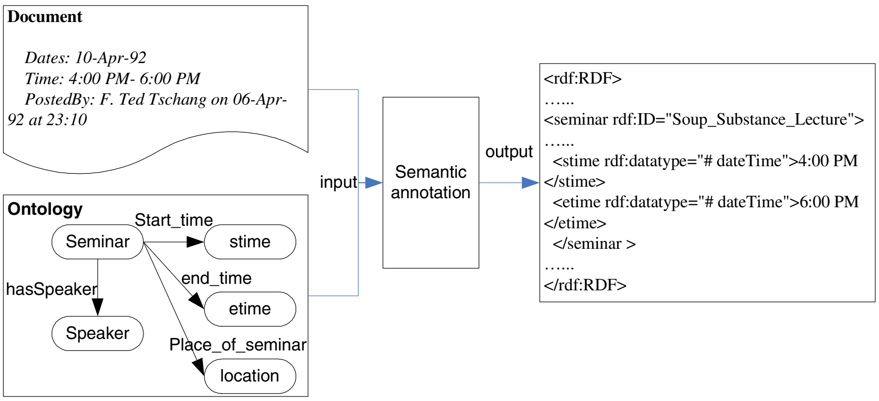


Fig. 1. An example of semantic annotation

Semantic annotation aims to markup the web pages by an ontology, which defines the meaning of contents in the pages. Figure 1 shows an example of semantic annotation. The inputs of semantic annotation are web document and ontology, the output is the annotated result. In this example, the text “4:00 PM” and “6:00 PM” are annotated as “stime” and “etime” respectively.

Many existing tools have semantic annotation features. However, most of them support only manual annotation [26, 27]. Manual annotation is tedious, time consuming and error-prone. More recent efforts make use of existing wrapping method (e.g. Amilcare [9] and Rapier [7]) and disambiguation technology to automate this process [1, 17, 24, 28, 42, 49]. In information extraction, many models are proposed, such as: Hidden Markov Model [21, 44], Maximum Entropy Model [3, 8], Support Vector Machines [13], and Conditional Random Field [32]. See section 8.4 for details. The methodologies proposed in the previous work can be used in semantic annotation. However, they seem not sufficient for the task.

In this paper, we try to address semantic annotation in a new approach. In our approach, the annotation mainly consists of two stages: learning and annotation.

In learning, we generalize the learned rules. For each rule, we define the extracted text and its context text as features and assign a label. The label represents which type of metadata the extracted text should be annotated. We use the labeled documents to generalize the learned rule set in advance.

In annotation, we identify, extract, and annotate the string in given documents using learned rules.

We propose a method called Similarity based Rule Learner (SRL) to generate the rules. We utilize an empirical method to select the optimal dynamic window size for the rule. We make use of machine learning techniques to improve the annotation results by selecting the correct annotated instances and by predicting the missing annotated instances. Finally, we provide a mechanism for explaining the nature of the rule learner and annotator. The explanation can be very useful in system analysis both for development and usage scenario.

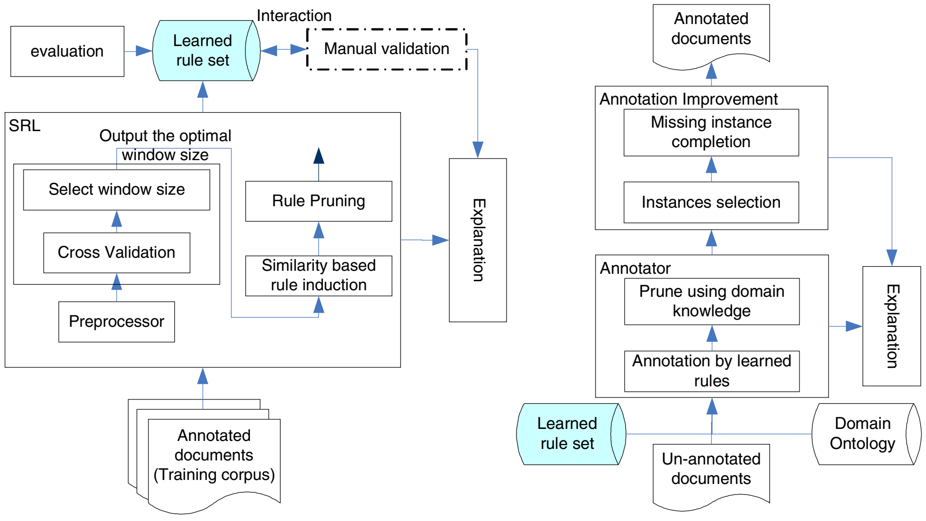
We have developed a tool based on the approach that is call iASA. iASA is targeting structured web data. In iASA, we learn the annotation rules by SRL, and apply the learned rules to un-annotated documents. We also make use of the explanation to help users refine the learned rules or to correct the annotation results.

We conducted the experiments on two data sets, and performed the comparison with existing methods. The experimental results indicate that the proposed method performs well for semantic annotation. We applied the method in a practical project: TIPSI. In TIPSI, we are aimed to extract the semi-structured information from company annual reports for Stock Exchange. Both of the results of the analysis on a user feedback and the result of an analysis on annotation results show that the features in iASA are helpful. We are trying to apply the tool to Contact Search on internet.

The rest of the paper is organized as follows. In section 2, we give an overview of the architecture of iASA and introduce the terminology and notations used throughout the paper. In section 3, we describe our rule learning method: similarity based Rule Learner (SRL). In section 4, we introduce the annotator. In section 5, we propose to improve the annotation results by using machine learning methods and in section 6, we provide a mechanism for explaining the nature of the rule learner and annotator. Section 7 gives our experimental results. Finally, before making concluding marks, we give the survey of related works.

2 iASA: An Automated Semantic Annotator

We perform semantic annotation in four main passes of procedures: learning, annotation, refinement, and the explanation. These procedures correspond to the following four components: SRL, Annotator, Annotation Improvement and Explanation (as shown in figure 2).



(a) Rule Learning (b) Annotator

Fig. 2. The architecture of iASA

In Rule Learning, the input is the annotated documents. We preprocess the annotated documents and construct an initial rule sets. We then use an empirical method to find the optimal window size for each concept's/property's rules. Next, we employ the similarity based rule induction on the initial rule set and obtain a set of annotation rules. After the pruning procedure, the output is a learned rule set. On the learned rule set, explanation component supports a user interaction, which helps the user to refine the learned rule sets.

In Annotation, the input is un-annotated documents. We apply the learned rules to the un-annotated documents and annotate them according to an ontology. The ontology is defined to represent the annotated or un-annotated documents. After that, we try to refine the annotation results by using machine learning techniques. The output is the annotated documents (an example of annotated document is shown in figure 9). In the procedure, the explanation component supports a user interaction to help the user understand the annotation and the refinement process.

In the rest of this section, we will present the necessary terminologies and definitions.

2.1 Terminology and Notation

The following terminologies are used throughout this paper.

For short, we use entity to denote concept and property in ontology hereafter. Let $E=\{e_i|i\in[1,m]\}$ be a set of entities, where m is the number of entities. In annotated documents, the texts that are annotated as entity e_i are called the instances of e_i. An instance's content includes one or more words/phrases. Let i be an instance and let I denote a set of instances. Notation $< I_i, C_i >= \{< i_{i1}, c_{i1} >, < i_{i2}, c_{i2} >, \cdots, < i_{in_i}, c_{in_i} >\}$ denotes a set of instance-occurring time pairs, in which I_i is the set of instances of e_i and i_{i1} is one possible value of the instance and c_{i1} is the corresponding occurring time of the value in the instance set I_i. n_i is the number of entity in I_i.

2.1.1 Token Definition

1. Token

In both Rule Learning and Annotator, the annotated and un-annotated documents are split into a sequence of tokens $\{t_0, t_1, \ldots, t_n\}$. Each token can be a word (A word is a set of contiguous upper or lowercase letters), a gazetteer entry (e.g. person's first name, currency unit, location, etc) or a name entity (e.g. organization, person' name and date, etc).

Each token is associated with linguistic attributes. Specifically, linguistic attributes of word include: "kind", "orth", "type", "pos", and "name"; linguistic attributes of gazetteer entry (lookup) include: "name" and "type"; and linguistic attribute of name entity is its "name". Table 1 shows the details of the three types of tokens and their attributes. The columns respectively represent token, its attributes, description of the attribute, example of the attribute's value, and description of the example value.

2. Token similarity

According to the attributes of tokens, we define the similarity of two tokens as:

$$Token\_S(t_i, t_j^{'}) = \sum_k w_k match(a_{ik}, a_{jk})$$

where a_{ik} and a_{jk} respectively represent the k-th attribute vaule of token t_i and $t_i^{'}$. Function *match*() is a zero-one function, which ends up with 1 when a_{ik} equals to a_{jk}, 0 otherwise. w_k is the weight of the k-th attribute. (In experiments, the weight of attributes "name", "pos", "kind", "type", and "orth" for word are tentatively set as 0.45, 0.25, 0.1, 0.1, and 0.1, respectively; the weight of attributes "name" and "type" for gazetteer are tentatively set as 0.7 and 0.3.)

Table 1. Details of the three types of tokens and their attributes

Token	Attribute	Comment	Examples	Value Description
Word	Name	Original string	"Time"	
	Orth	Orthography of the word	"OneHyphen"	Indicates the word is a simple hyphen ("-" or "_")
			"AllCap"	Indicates all letters in the word are capitalized ("CHINA")
			"Capitalized"	Indicates the first letter is capitalized ("Time")
	Kind	Kind of the word	"punctuation"	Indicates the word is a punctuation ("," or ".")
			"word"	Normal word ("Time")
			"number"	Indicates the word is numeric ("486")
	POS	Part-Of-Speech of the word	"NN"	Singular common noun with word initial capital
			"CD"	Cardinal number (one, 100)
	Type	Indicate whether word is a space token or not	"Spacetoken"	Indicates the word is a enter or space
			"Token"	Other words except whose type is "spacetoken"
Lookup (gazetteer entry)	Name	Major type of the gazetteer entry	person's first name	E.g. "Jeanne"
			Organization	E.g. "Motorola"
			Location	E.g. "Oregon"
	Type	Minor type of the gazetteer entry	Female	Minor type of person's first name
			company	Minor type of organization ("Motorola")
			Province	Minor type of location ("Oregon")
Name entity	Name	Recognized over the original words.	Organization	Includes company or governmental organization ("Oracle", "Intel")
			person's name	E.g. "Jeanne Heembrock"
			date	E.g. "17 Nov 1996"

3. Pattern definition

A pattern is a sequence of tokens with length *n* (n could be zero). Formally, a pattern can be written as:

$$pattern = \{t_0, t_1, \cdots, t_n\}$$

where t_i is the *i*-th token in the pattern.

As the example in figure 1, the string “4:00 PM” is annotated as the property *stime*. It can be viewed as a body pattern (defined in the following section) containing one token: name entity “date” or a body pattern containing four word tokens: “4”, “:”, “00”, and “PM”).

We also define similarity between the two patterns. See section 3.3 for details.

2.1.2 Rule Definition

Semantic annotator needs abstract patterns that are defined based on all the features potentially helpful for the ontology annotation. These abstract patterns are defined as rules. In iASA, the rule is represented in XML.

Each rule has an entity name and consists of three parts: 1) left pattern: it is used to match the text that immediately precedes the instance (*w* tokens to the left); 2) body pattern: it corresponds to the instance of an entity (tokens contained); 3) right pattern: it is used to match the text that immediately follows the instance (*w* tokens to the right).

Figure 3 shows an example of a rule of entity *etime*. In the rule, Left, body and right pattern are denoted by “leftpattern”, “bodypattern” and “rightpattern”, respectively, and the token in these patterns is denoted by “tag”. The attribute “indicator” of “tag” denotes the type of the token. The type can be name entity, gazetteer entry (lookup), or word. Attributes “kind”, “name”, “orth”, “type”, and “pos” represent the attributes of corresponding token (as shown in table 1). We use “<tag indicator=“unknown”/>” to denote a placeholder.

```
<rule name="etime" no="12">
  <leftpattern>
    <tag indicator="word" kind="word" name="Time" orth="O: Capitalized" type="token" />
    <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
    <tag indicator="unknown" />
    <tag indicator="word" kind="punctuation" name="-" orth="O: OneHyphen" pos=":" type="token" />
  </leftpattern>
  <bodypattern>
    <tag indicator="nameentity" name="date" />
  </bodypattern>
  <rightpattern>
    <tag indicator="word" name="" type="spacetoken" />
    <tag indicator="word" kind="word" orth="O: Capitalized" type="token" />
    <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
  </rightpattern>
</rule>
```

Fig. 3. An example of rule

In order to learn the target rules, the user typically needs to provide a set of annotated documents (also called training documents). In the training documents, we label a set of training instances in advance. Each instance is enclosed by a start tag (e.g. <stime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">) and an end tag (e.g. </stime>). Context of annotated instance (viz. leftpattern, bodypattern and rightpattern) is constructed according to the start and end tags. Specifically, the text that enclosed by a pair of tags is viewed as an instance and is transformed into a bodypattern; *w* tokens that precedes the instance are transformed into leftpattern; and *w* tokens that follows the instance are transformed into rightpattern.

3 SRL—Similarity Based Rule Learner

SRL has four main modules: preprocessor, rule set initialization, rule induction, and rule set pruning. The input is annotated documents. Preprocessor exploits Natural Language Processing (NLP) techniques to process the documents. Rule set initialization takes the preprocessed corpus as input and generates the initial rule set using dynamic window size based context. Rule induction performs rule generalization in an iterative mode. In each iteration, we select pairs of rules with high similarity in the initial rule set according to the rule similarity (see section 3.3 for definition of rule similarity), generalize new rules, evaluate each new rule, and then insert the new rule into the learned rule set if it survives the pruning phrase. Finally, SRL outputs the learned rule set.

3.1 Preprocessor

In preprocessing, we use NLP techniques to process the document, which has been proved to be useful for machine learning and information processing [7, 9]. NLP associates additional knowledge to each word in the document.

We make use of GATE as the NLP toolkit [14]. GATE is a general toolkit for text processing. It integrates many tools for NLP, including morphological analyzer, a POS tagger, gazetteer lookup, and named entity recognition (recognition of person name, dates, number and organization names, etc). For example, in the document snippet: "…; Patrick Stroh, assistant professor, SDS…", "Patrick Stroh" is annotated as an instance of entity *speaker*. After processing by GATE, each word is associated with linguistic knowledge: part of speech (POS), token kind (Kind), lookup, and name entity, etc. Table 2 gives an example about how GATE provides the linguistic knowledge.

Table 2. An example of preprocessed result with NLP knowledge

Instance with NLP Knowledge					**Annotated as**
Word	**POS**	**Kind**	**Lookup**	**Name Entity**	
;	:	Punctuation			
Patrick	NNP	Word	Person's first name	Person	Speaker
Stroh	NNP	Word			
,	,	Punctuation			
assistant	NN	Word	Jobtitle		
professor	NN	Word			
,	,	Punctuation			
SDS	NNP	Word			

The linguistic knowledge seems very useful in rule induction. For example, tags in the rule can be relaxed by substituting constraints on words by constraints on some parts of the linguistic knowledge. In this way, we can generalize the rules by name entity (e.g. "person's first name (male)") or POS (e.g. "NNP") instead of only flat words.

3.2 Rule Set Initialization

For rule induction, we first need to construct the initial rule set that contains the mostly specified rules. In annotated documents, each instance is enclosed by a start tag and an end tag. We transform the instance and its context in the preprocessed document into an initial rule. Existing methods usually define a window of fixed size (size=*w*) as the instance context and build the initial rule by using *w* tokens to the left and *w* tokens to the right for a given annotated instance.

In the rule set initialization, we exploit the linguistic attributes, and we also use the dynamic window size by an empirical method.

1. Linguistic knowledge based context

Suppose *w* is "3", for the annotated instance of *stime* in figure 1, table 3 gives a comparison of linguistic knowledge based context and word based context. The second column represents the linguistic knowledge based context and the third column represents the flat word based context.

Table 3. Linguistic knowledge based context vs. word based context

Initial Rule	Linguistic Context	Words' Context	Semantic Entity
Left Pattern	Date (name entity)	"10-Apr-92"	
	"Time" (word)	"Time"	
	":" (punctuation)	":"	
Body Pattern	Date (name entity)	4:00 PM	*stime*
Right Pattern	"-" (word)	"-" (word)	
	Date (name entity)	"6:00 PM" (word)	
	Return	Return	

In table 3, we see that by the comparison of the word based context, the linguistic knowledge based context substitutes a name entity 'Date' for both '10-Apr-92' and '6:00 PM'. The linguistic knowledge based context seems more reasonable, because the rules derived from it intuitively can be applied to broader cases.

2. Optimal dynamic window size based context

Analysis on our preliminary experimental results shows that different entities prefer to contexts with different window sizes. For example, in the task of annotating CMU Seminar announcements, with the increase of window size (tested from 2 to 8), the performance (evaluated by F-measure) for entity *etime* becomes better; however for entity *stime*, the performance becomes worse.

We perform the window size selection for each entity by using cross-validation, a typical approach for experimental selection [43]. Cross validation is a commonly used technique in machine learning to prevent bias. The main idea is based on the following assumption: if the rules learned from a subset of the training data produce accurate result, it is also likely that it will produce highly accurate predictions when trained on the entire dataset.

In this method, to determine *w* for each entity, we evaluate the performance on training corpus with contexts of different window sizes using cross-validation. Specifically, let *T* be the training corpus. The examples in *T* are first randomly divided into *d* equal parts T_1, T_2, ⋯, T_d (we use d = 10 in our experiments). Next, for each part T_i, $i \in [1,d]$, we try to learn the rules from the other (*d*-1) parts, then apply the learned rules to the examples in T_i. Finally, we select the window size that performs best for each entity. Section 7 will show the comparison of dynamic window size and fixed one.

3.3 Rule Induction

The input of rule induction is an initial rule set, rule induction aims to learn rules over the initial rule set and to output a learned rule set.

Within rule induction, the following problems should be considered:

- Tokens in body patterns may be very sparse, which make it difficult for generalization. For example: instances of *address*: “Baker Hall 237B”, “room 1001”, “WeH 5403”, etc.
- Sometimes token in the rule may be only a placeholder instead of a meaning ones, e.g. in examples: “the speaker is <speaker>”, “the speaker: <speaker>”, and “the speaker, <speaker>”. A placeholder can be defined so as to match “is”, “:” and “,”.
- Each learned rule should be scored. The higher the score is, the higher probability that the rule is accepted in the final rule set.

In regular expression language, “*” usually is exploited to represent any group of characters and “?” usually is exploited to represent any character. In our rule definition, we extend this idea slightly and make use of “*” and “?” to respectively represent any group of tokens and any token. In this way, for the first problem, we use “*” to represent the body pattern and for the second problem, we use “?” to represent the placeholder.

Unfortunately, most of the existing rule induction methods do not have the feature of generalizing the wildcard “*” and “?”. In this section, we introduce how SRL solves these two problems. Moreover, we describe the evaluation metric that is used to score each learned rule.

1. Rule similarity

Rule similarity is the basic idea in SRL. SRL runs in an iterative mode. In each iteration, we always try to select the most similar pair of rules for generalization. Califf et al adopt a random strategy for the rule selection [7]. The similarity based selection method seems more reasonable. There are two reasons for this: similarity based method is more efficient than random method by avoiding the pseudo generalization and the learned rules by the random method may be inconsistent.

We then need to define the similarity of a pair of rules. Typically, a rule can have three patterns (see section 2.1.2 for the definition). We calculate the similarities of the three corresponding patterns and sum them into an aggregate one. But soon, we found that there are mainly two kinds of rules: rules with sparse body patterns and rules with non-sparse body patterns. Examples of the former include rules of entity *speaker*; examples of the later include rules of entity *platform* or entity *database*.

Our proposal is that similarity of rules with sparse body patterns are calculated by the similarities of corresponding left patterns and right patterns, and similarity of rules with non-sparse body patterns includes only the similarity of body patterns. We, therefore, define rule similarity as:

$$sim(r_1, r_2) = \begin{cases} sim(lp_1, lp_2) + sim(rp_1, rp_2), & sparse(bp) > \mu \\ sim(bp_1, bp_2), \quad sparse(bp) \le \mu \end{cases}$$

where r_1 and r_2 are two rules. $sim(lp_1, lp_2)$, $sim(bp_1, bp_2)$, and $sim(rp_1, rp_2)$ respectively represent the similarities of corresponding left patterns, right patterns, and body patterns in rules r_1 and r_2. $sparse(bp)$ is a measurement indicating whether the body pattern is sparse or not. It is calculated by $count(value)/count(instance)$. $count(value)$ is the number of instance values for the given entity, and $count(instance)$ is the total number of instances (e.g. "4:00 PM" and "4:00 PM" are two instances of *etime* with only one value). Parameter μ is a threshold (we tentatively set it as 0.5).

2. Pattern similarity

We calculate the similarity of patterns by a recursive procedure called Multiple Layer Recursive Matching (MLRM) algorithm. This idea is derived from [46].

In MLRM algorithm, the input is two patterns: *pattern*$_1$ and *pattern*$_2$. Output is the similarity score of the two patterns.

The MLRM algorithm is described in Figure 4.

```
Input: pattern1[$t_0,t_1,\dots,t_n$]
       pattern2[$t_0',t_1',\dots,t_m'$];
Output: Seq_S(pattern1,pattern2); //pattern similarity
MLRM(pattern1[$t_0,t_1,\dots,t_n$], pattern2[$t_0',t_1',\dots,t_m'$])
{
  if(either pattern1 or pattern2 is empty or no more higher layer)
    return 0;
  A=0;B=0;
  while(A<=n and B<=m)
  {
    for( i from A to n )
      for( j from B to m )
        select the most similar token pair : argmax(Token_S ( $t_i$, $t_j'$));
        {
           Seq_S(pattern1,pattern2) += Token_S ( $t_i$, $t_j'$) * weight$_K$;
           Seq_S(pattern1,pattern2) += MLRM(pattern1[$t_A,\dots,t_{i-1}$],
                                  pattern2[$t_B',\dots,t_{j-1}'$]);
           A=i+1; B=j+1; break to while;
        }
  }
  Seq_S(pattern1, pattern2) += MLRM(pattern1[$t_A,\dots,t_n$],
                                  pattern2[$t_B',\dots,t_m'$]);
  return Seq_S(pattern1, pattern2);
}
```

Fig. 4. The MLRM algorithm

MLRM uses the function *MLRM*() for estimating the similarity between two patterns. *Token_S*(t_i, t_i') calculates the similarity between token t_i and t_i' using their linguistic information (token similarity is defined in section 2). *Seq_S* is the similarity of the two patterns by aggregating the similarities of individual pairs of tokens. MLRM recursively computes similarity between two patterns at multiple layers with different similarity weights (*weight*$_k$). MLRM attempts to find a pair of the most similar tokens from *pattern*$_1$ and *pattern*$_2$. In each layer, the pair of tokens essentially divides each pattern into two sub-patterns. MLRM continues the process until one of the sub-patterns becomes empty. The weight at each layer is currently empirically assigned to reflect the relative importance of the token similarity in that layer. We tentatively set the weights w_1, w_2, w_3, and w_4 as 10, 9, 8 and 7 respectively.

Figure 5 illustrates MLRM with an example. At step 0, a pair of the most similar tokens is found (connected by a dark line) and divides each pattern into two sub-patterns. Each pair of sub-patterns is recursively processed by MLRM as indicated in step 1. In this step, two additional similarity pairs are found and the two patterns are further divided into four sub-patterns to be processed recursively at step 2. MLRM stops when no more nodes are left in the sub-patterns.

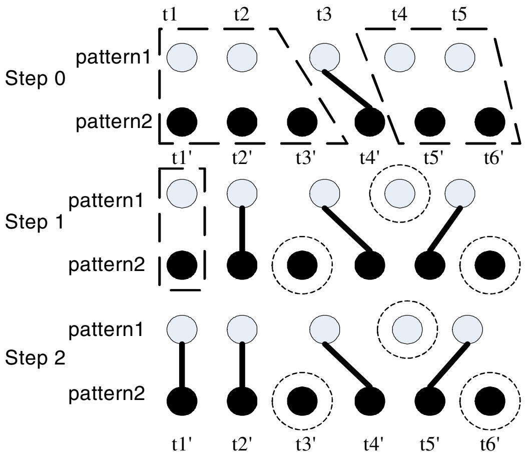


Fig. 5. An example of recursive matching in MLRM

3. Rule generalization

The task of rule generalization is to induce new rules from the pair of similar rules. The new rules should cover the pair of rules.

In rule generalization, we take a strategy of divide-and-conquer. For a pair of similar rules, our method is to generalize the three pairs of patterns (i.e. left patterns, body patterns, and right patterns), and then combine the generalized patterns into a new rule. For each pair of patterns, the generalization is performed by starting from constrains of all linguistic information on tokens to some relaxation. The algorithm of rule generalization is shown in figure 6.

In rule generalization, we first select the most similar rules by *getSimilarRules*(). The number of selected rules is not necessary two. Hence the algorithm calls function *getRealPairs*() to generate pairs of rules. Then, for each pair of rules, the algorithm uses *generalizePatterns*() to generalize left pattern, body pattern, and right pattern respectively. *generalizePatterns*() returns a collection of possible induced patterns for the two input patterns (note: for two patterns, there might induce multiple patterns with different linguistic information). Function *getAllPossibleRules*() returns all possible rules by combining the three collections of patterns. For each rule, algorithm evaluates it both on the initial rule set and on the original training corpus, and then adds it to the learned rule set if its score exceeds the minimal rule score (a threshold predefined). (For the detail of rule evaluation, please refer to the following section.)

```
RuleInduction(RuleSet ruleset)
{
   rssimilar=getSimilarRules(ruleset);
   collect_RP=getRealPairs(rssimilar);
   foreach rulepair in collect_RP
   {
      rule1 and rule2 in rulepair
      col_leftpat=generalizePatterns(rule1.leftpattern, rule2.leftpattern);
      col_bodypat=generalizePatterns(rule1.bodypattern, rule2.bodypattern);
      col_rightpat=generalizePatterns(rule1.rightpattern, rule2.rightypattern);
      col_rules=getAllPossibleRules(col_leftpat,col_bodypat,col_rightpat);
      foreach rule in col_rules
      {
         score=EvaluateRule(rule);
         if( score > minRuleScore)
            col_learnedrules.add(rule);
      }
   }
   return col_learnedrules;
}
```

(a) Generalization of rules

```
generalizePatterns ( Pattern pattern1, Pattern pattern2)
{
   if( pattern1==null or pattern2==null)
      return null;
   relaxlevel = 0;
   while( true )
   {
      collectpair = getMatchedTagPair(pattern1,pattern2, relaxlevel);
      newpattern = getPattern(pattern1, pattern2, collectpair );
      pattern_collection.add( newpattern );
      relaxlevel++;
      if( relaxlevel>maxrelax )
         break;
   }
   return pattern_collection;
}
```

(b) Generalization of patterns

Fig. 6. The algorithm of rules generalization

In pattern generalization, *generalizePatterns*() searches for the matchable pairs of tokens (by function *getMatchedTagPair*()). The search starts from full constraints (*relaxlevel*=0) to maximum relaxation on linguistic information (*relaxlevel*= *maxrelax*-1). In the case of full constraints, two tokens are considered to be matched only when all of their attributes are equal, including "indicator", "kind", "name", "orth", "pos", and "type" (see table 1 for details). With the increase of *relaxlevel*, the matcher relaxes the condition by the order of "name", "pos", "kind". By relaxing the condition, we mean ignoring the corresponding attributes in the matching process. Parameter *maxrelax* is the stop condition to control the relaxation progress. The discovered matchable token pairs are used as the initial points to generalize the pattern. The function *getPattern*() returns the generalized results of the two patterns based on their matchable token pairs. Figure 7 gives an example to illustrate the strategy used in *getPattern*().

In this example, t2-t2', t3-t4' and t5-t5' (linked by dark line) are three matchable token pairs returned by function *getMatchedTagPair*(). The three pairs of tokens divide the two patterns into four token groups: (t1-t1'), (t3'), (t4) and (t6'). For each token group, tokens in the two patterns are induced one by one from left to right for body pattern and right pattern or from right to left for left pattern. As for uneven length, such as (t3'), (t4) and (t6'), the algorithm adds wildcard "?" to the remainder tokens. Symbol C(t1,t1') means the induction of two tokens: t1 and t1'. "(t3')?" indicates that the token t3' can either occur or absent in the target cases.

RuleInduction() is run iteratively until the stop condition is met. The stop condition is that no more general rules can be induced. Finally, rule generalization outputs the learned rule set.

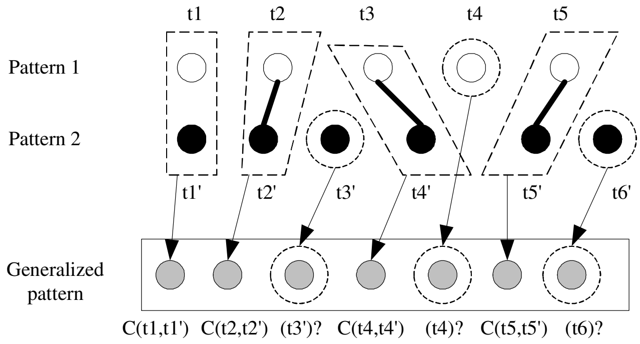


Fig. 7. An example to describe how two patterns are generalized in *getPattern*()

4. Rule evaluation

Each rule is scored by the function *Evaluation*() (shown in figure 6(a)). The score is the combination of two factors: score on the initial rule set and score on the original training corpus. Both scores are evaluated by F-measure. F-measure is defined as:

$$F = \frac{1+\beta}{\dfrac{\beta}{precision} + \dfrac{1}{recall}}$$

where *precision* is the percentage of correctly annotated instances in the annotated instances; *recall* is the percentage of correctly annotated instances in the correct instances. Parameter β indicates the degree of preference on precision/recall.

We call the score on the initial rule set as *rulescore*. We take the annotated instances of current entity as positive examples and instances of the other entity as negative ones. Score of each rule is evaluated by F-measure on the initial rule set. The metric *rulescore* should concern more precision than recall, because the generalized rule may only cover the two source rules after the first iteration, which leads to a very low recall. On the other hand, low precision indicates that the new rule covers negative examples. Therefore we set the weight β as 16.

We call the score on original training corpus as *realscore*. We apply the learned rule on the original training corpus (i.e. take training corpus set as test data), which can avoid over-learned rules to a certain extent. The over-learned rules may produce good result on the initial rule set, but they can also import noise. Such noise may lead to low precision. For calculating *realscore*, we set the parameter β as 1.

Finally, by multiplying *rulescore* by *realscore*, we obtain the final score of the new rule.

3.4 Rule Set Pruning

In rule set pruning, we aim to remove the redundant and unreliable rules in the learned rule set.

To remove redundant rules, it is necessary to define what a redundant rule is. We give the definition of redundant rule.

Redundant Rule: If a rule is covered by other rule(s) in the rule set, then this rule is redundant. In other words, if all instances that annotated by this rule can also be annotated by other rule(s), then this rule is a redundant rule.

In terms of the definition, we developed methods to judge whether a rule is redundant. Our methods include two steps: (1) judging whether a rule is covered by another rule; (2) judging whether a rule is covered by other rules. We exploit the annotated instances to judge whether a rule is covered by other rules.

Whether or not a rule is covered by another rule is based on the observation: if every pattern of a rule is more general than the corresponding pattern of another rule, we say that the latter rule is covered by the former rule.

Figure 8 gives an example. In the example, pattern 2 is covered by pattern 1 because token (<tag type="">) of pattern 1 is more general than token (<tag type="token">). <tag type=""> denotes a token that can be any type.

```
Pattern 1           Pattern 2
<pattern>           <pattern>
  <tag type="" >      <tag type="word" >
</pattern>          </pattern>
```

Fig. 8. An example of two patterns

We exploit the annotated instances to determine whether a rule is covered by other rules: if instances annotated by a rule can be also annotated by some other rules, we say that the rule is a redundant rule.

To remove unreliable rules, we make use of ontology knowledge. We use entity type to infer whether the rule is reliable or not. For example, in the seminar ontology the property *stime* is defined as:

```
<owl:DatatypeProperty rdf:ID="stime">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime" />
  <rdfs:domain rdf:resource="# seminar " />
</owl:DatatypeProperty>
```

Then we use the date type *dateTime* to judge whether the body pattern of a rule conforms to it or not. After pruning, SRL outputs the final learned rule set.

4 Annotator

Annotator takes the un-annotated documents as input, preprocesses them by NLP which is the same as that in SRL, applies the learned rules on them, and annotates the documents according to an ontology.

An example of output by the annotator is shown in figure 9. The format conforms to the ontology standard language OWL DL [15], which can facilitate further reasoning process.

Using iASA in the experiments, we have found that domain knowledge can greatly improve the accuracy of annotation. Domain knowledge can be used to evaluate annotations and prune wrong annotations, and it is also helpful for directing the search process. In semantic annotation, domain knowledge is usually represented by ontology. We then use the restrictions in the ontology for improving the annotation.

```
<rdf:RDF>
 <seminar rdf:ID="Soup_Substance_Lecture">
  <location rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Wherrett Room, Skibo</location>
  <stime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">12:00 PM</stime>
  <etime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">1:00 PM</etime>
  <hasSpeaker  rdf:resource="#Erik_Devereux" />
     <speaker rdf:ID="Erik_Devereux">
      <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Erik Devereux</name>
    </speaker >
  </hasSpeaker >
  <hasSpeaker >
    <speaker rdf:ID="Patrick_Stroh">
      <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Patrick Stroh</name>
    </speaker >
  </hasSpeaker >
  <hasSpeaker rdf:resource="#Richard_Smith" />
    <speaker rdf:ID="Richard_Smith">
      <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Richard Smith</name>
    </speaker >
  </hasSpeaker >
 </seminar >
</rdf:RDF>
```

Fig. 9. An example output of annotator

Domain Restriction: The constraints that are defined in the ontology include “date type”, “cardinality”, “minCardinality”, etc. Table 4 shows examples of domain restrictions currently used in iASA.

Table 4. Examples of domain restrictions that are exploited to improve the annotation

Constraint Types	Examples
date type	If the data type of annotated instance does not match the data type of entity *x* in ontology, then remove the instance.
Cardinality	If cardinality of entity *x* is 1, and annotator finds multiple candidate instances, then select the candidate annotated by the best rule (with highest score) as the instance of entity *x*.
	If cardinality of entity *x* is 1, and annotator doesn’t find any candidates, then relax the condition in the rule and search the document again (the relaxation strategy is same as that in rule induction).

5 Improving the Annotation by Machine Learning Methods

In annotation, we met two problems: correct instances selection and missing instances prediction. In this section, we try to make use of machine learning methods to solve the two problems.

5.1 Problem Statement

Now we give the definitions of the two problems.

(1) Correct instance selection. In automatic annotation, entity may be annotated with multiple instances even in one document. Some annotated instances are correct, but some may be wrong. The problem is how to identify the correct instances.

(2) Missing instance prediction. There may be some instances that are not annotated by automatic annotation, we call it missing instances. Since typically recall of semantic annotation is significantly lower than its precision, it is indeed necessary to deal with the problem.

In existing methods for correct instances selection, Califf et al propose to select the instance(s) that annotated by the highest scored rule [7]. The method can solve some problems, but it is not sufficient, because: (a) even the same rule might annotate multiple instances including both correct and erroneous ones; (b) some correct instances can be annotated by the other rules instead of the highest scored rule.

In existing methods for missing instance prediction, Nahm et al propose to induce predictive rules which are then used to predict the missing instances [39]. The method mines the association rules from the data. For example, suppose following rule is discovered from data on entity *program language* and *topic area*: "SQL"$\in$*language*$\rightarrow$"Database"$\in$*area*. If the annotation system annotated only "SQL" for language, but failed to annotate "Database" for *area*, then the method can assign the "Database" as the value of *area*.

In this paper, we formalize the correct instance selection problem as that of classification. When selecting the correct instances, we use a classification model to identify whether or not an instance is correct.

For missing instance prediction, it is difficult to accurately predict the missing instances that have random values, even by human. We confine ourselves to predict the instances that have enumerative values in missing instance prediction. It seems reasonable for predicting instances that have enumerative values, because we have observed that 17.4% of the entities have enumerative values in our experimental data.

We then formalize the missing instance prediction problem as that of multi-class classification. When predicting a missing instance, we use a classification model to predict which value has the highest probability as the missing instance. The classification model is trained using the training data in advance.

5.2 Classification Model

We make use of SVM (Support Vector Machines) as the classification model [47].

Let us first consider a two class classification problem. Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a training data set, in which x_i denotes an instance (a feature vector) and $y_i \in \{$-

1,+1} denotes a classification label. In learning, one attempts to find an optimal separating hyper-plane that maximally separates the two classes of training instances (more precisely, maximizes the margin between the two classes of instances). The hyper-plane corresponds to a classifier (linear SVM). It is theoretically guaranteed that the linear classifier obtained in this way has small generalization errors. Linear SVM can be further extended into non-linear SVMs by using kernel functions such as Gaussian and polynomial kernels.

We choose polynomial kernel, because our preliminary experimental results show that it works best for our current task. When there are more than two classes, we adopt the "one class versus all others" approach, i.e., take one class as positive and the other classes as negative.

Now, we have two kinds of "instances": annotated instance of an entity and instance in SVM model. To distinguish them from each other, we use sample to denote instance in SVM hereafter.

5.3 Correct Instance Selection

We divide the problem of instance selection into three categories:

(1) Multiple instances are annotated by the same rule. For example, "a MIS Manager" and "MIS Manager" might be annotated as "title" by a same rule. (The two examples are instances of entity "title" in the task of misc.job.offered. Correct one should be "MIS Manager".)
(2) Multiple instances are annotated by different rules. For example, "Marian D' Amico", "Charles E. Leiserson", and "Jeffrey V. Hill" may be annotated as "speaker" by different rules. (The three examples are instances of entity "speaker" in CMU seminar announcement. Correct one should be "Charles E. Leiserson".)
(3) Hybrid of the two situations. For example, "Leiserson", "Charles E. Leiserson", and "to Charles" may be annotated as "speaker", in which "Leiserson" and "Charles E. Leiserson" are annotated by a same rule, and "to Charles" is annotated by another rule. (The three examples are instances of entity "speaker" in CMU seminar announcement. Correct one should be "Charles E. Leiserson").

The method of selecting by highest scored rule can only deal with the problem of the second category, and will fail on the other two categories. Even on the second category, the method may fail when the correct instance is annotated by a lower scored rule.

We view the instance selection problem as that of classification. The correct instance selection consists of two stages: training and identification.

In identification, when problem of multiple instances occurs, we identify whether or not each instance is correct using SVM model. Then we rank the instances by scores output by SVM. We select the instance ranked top as the correct one.

In training, we construct the SVM model that can be used to identify the instance. In the SVM model, we view an instance as a sample in SVM. For each sample, we define a set of features and assign a label. The label represents whether the instance is correct or not. For each entity, we use instances of the entity in the annotated documents as positive samples for SVM and the others as negative samples. We use the labeled samples to train the SVM model in advance for each entity.

We view each instance as a ‘document’, and convert the ‘document’ into a bag of words. We apply stop-word filtering and word stemming on the bag of words. After that, we construct an attribute-value representation of the ‘document’. Each distinct word w_i corresponds to a feature with its value. For the feature value, we use $TF(w_i, x)*IDF(w_i)$, a typical method to estimate the word weight in that document. $TF(w_i, x)$ represents the frequency of word w_i occurs in the document x. $IDF(w_i)$ represents the inverse document frequency of a word. It is defined by:

$$IDF(w_i) = \log \frac{n}{DF(w_i)}$$

here n is the total number of instances. $DF(w_i)$ is the number of documents the word w_i occurs in.

Finally, for multiple instances, we obtain a ranked list, and then we do the selection according to the following rules:

(1) For entity whose cardinality is unique, the top ranked instance is selected as the correct one.
(2) For entity whose cardinality a is greater than 1, the a top ranked instances that do not overlap with each other are selected as correct ones. No overlap is very important. Considering the examples above, instances “a MIS Manager” and “MIS Manager” might both obtain a higher scores compared to other instances. No overlap rule can further remove “a MIS Manager”.
(3) For entity whose cardinality is multiple, instances that are classified as positive samples and do not overlap with each other are selected as correct ones.

5.4 Missing Instance Prediction

In missing instance prediction, we aim to predict the missing instances that have enumerative values (we judge whether an entity has enumerative values in terms of its definition in ontology). We view the problem of missing instance prediction as that of multi-class classification. Here, values of the missing instances correspond to the classes in the classification. It also consists of two stages: training and prediction.

In prediction, we predict the missing instances by using the extracted instances. Specifically, for an entity with enumerative value type, we view the annotated document as a sample in SVM, and then predict the value which the entity should have. We use the value that has the highest score output by SVM as the missing instance.

In training, we construct the SVM models that can be used to predict the value. In SVM model, for an enumerative valued entity, we view each annotated document as a sample. For each sample, we define a set of features and assign a label. The label represents all possible values of the entity. For each label of an entity, we use the annotated documents with this label as the positive samples for SVM and the others as negative samples. We use the labeled samples to train the SVM models in advance for each enumerative valued entity.

To represent the features for each sample, we view each annotated document as a sample. For an annotated document, we first extract all the annotated instances from the annotated document. Next, the instances are converted into a bag of words. After that we use the same method as that in the correct instance selection to prepare the attribute-value representation for each sample.

Finally, for a missing instance of the enumerative valued entity, we obtain scores from SVM for each possible value. We complete the missing instances by the following rules:

(1) For entity whose cardinality is unique, the value with the highest score is selected as the missing instance.
(2) For entity whose cardinality *a* is greater than 1, *a* values that have higher scores are selected as the missing instances.
(3) For entity whose cardinality is multiple, only the value with the highest score is selected as the missing instance.

6 Explanation

In iASA, we try to provide an environment for user to quickly annotate the web pages according to an ontology. Practically, the user needs to inspect the learned rules and the annotation results produced by the system, modify them and provide feedbacks to the system.

As an annotation system relies on complex algorithms, there is a requirement for the system to explain the nature of the generated rules to the user. This idea is derived from [16]. Explanations can greatly help user gain insights into the rule induction and annotation process. In this way, the user can easily focus on the correct rules and the annotation results.

Figure 10 is an example of the scenario. It gives a learned rule for entity *etime* by iASA.

This rule produces well accepted results on training corpus but low recall on the test corpus. The user is uncertain about how to improve this rule or whether or not the rule should be removed from the rule set. He wants iASA to explain how the rule is generalized and how this rule is evaluated.

```
<rule name="etime" no="12">
  <leftpattern>
    <tag indicator="word" kind="word" name="Time" orth="O: Capitalized" type="token" />
    <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
    <tag indicator="unknown" />
    <tag indicator="word" kind="punctuation" name="-" orth="O: OneHyphen" pos=":" type="token" />
  </leftpattern>
  <bodypattern>
    <tag indicator="nameentity" name="date" />
  </bodypattern>
  <rightpattern>
    <tag indicator="word" name="" type="spacetoken" />
    <tag indicator="word" kind="word" orth="O: Capitalized" type="token" />
    <tag indicator="word" kind="punctuation" name=":" orth="O: OtherPunct" pos=":" type="token" />
  </rightpattern>
</rule>
```

Fig. 10. An example rule

iASA induces rules from three initial rules and also shows how the tokens are generalized. For example, the third token in Figure 10 (<tag indicator="unknown"/> indicates a placeholder) in the left pattern of the rule is generalized from the tokens: a name entity (date: "Jan, 15th, 2004"), a word ("12") and a gazetteer entry (also called lookup) (time: "4:30"). This rule is tested on the initial rule set by score: 0.987 and on training corpus by score 0.87. The final score is 0.86, which is higher than the minimum rule threshold.

In the rest of this section, we will describe the data structure in explanation module and introduce what kinds of questions can be answered in the explanation.

6.1 Data Structure in Explanation

The key data structure underlying the explanation component of iASA is the dependency graph, which is constructed during the inducing process and the annotation process. The dependency graph records the flows of induction, data, and input and output of each system component. The nodes of the graph are: annotated documents, initial rules, similar rules, selected similar rule pairs, generalized pattern collections, all possible rules by the three generalized patterns, scores on the initial rule set and the training corpus.

Two nodes in the graph are connected by a directed edge if one of them is the successor of the other in the induction process. The label of the edge is the system process.

In explanation, we define an abstract node, which can be written as a tuple:

Abstract_Node=<*super_edges*, *node_type*, *node_data*, *sub_edges*>

The four elements in the tuple respectively represent a set of link-in edges, type of current node, data stored in the node, and a set of link-out edges. By link-in edge, we mean the directed edge that one predecessor of the current node links to the current node. By link-out edge, we mean the directed edge that the current node links to one successor. The type of current node can be one of rule, rule pair, pattern, tag, and annotated instance. The data corresponding to the *node_type* is stored in *node_data*.

We also define an abstract edge, which can be written as a tuple:

Abstract_Edge=<*subject_node*, *edge_type*, *edge_data*, *object_node*>

The four elements in the tuple respectively represent a subject node, type of the edge, data stored in the edge, and an object node. By subject node, we mean a node that the directed edge comes from. By object node, we mean a node that the edge directed to. The type of current edge can be one kind of processes in iASA, e.g. rule similarity computing, similar rule selection, rule induction, pattern generalization, rule scoring on initial rule sets, rule scoring on original training corpus, annotation, etc. The data corresponding to the type of the edge is stored in *edge_data*.

We make implementations for the nodes and the edges. In the implementation of each type of node, we extend the abstract node and define the corresponding data structure for storing information that is required in the explanation. In the implementation of each type of edge, we extend the abstract edge and define the corresponding data structure.

We take rule-node (implementation of node for rule) as the example to describe how we define the data structure for explanation. A rule has three patterns: left pattern, body pattern, and right pattern. The rule links to the three patterns by three link-out edges. A rule has three scores: score on initial rule set, score on original training corpus, and the final score. The three scores are defined as attributes in the rule-node. A rule has a semantic tag indicating which entity the rule is used to annotate. The semantic tag is also defined as an attribute. A rule has an attribute "number-of-covered-instance" indicating how many instances the rule can annotate in the training documents. Moreover, a rule has a global unique ID. Through link-in edges, a rule can have a set of predecessor. We can find one kind of predecessor by looking up the corresponding edge type in the set of edges. For example, if we want to find the pair of rules that are used to directly generalize into the current rule, we can lookup the link-in edges by "rule induction", the system returns a sub set of edges. We next query the subject nodes of the returned edges and can get the pair of rules. Traversing up in this way, we can get all rules that are used to generalize the current rule.

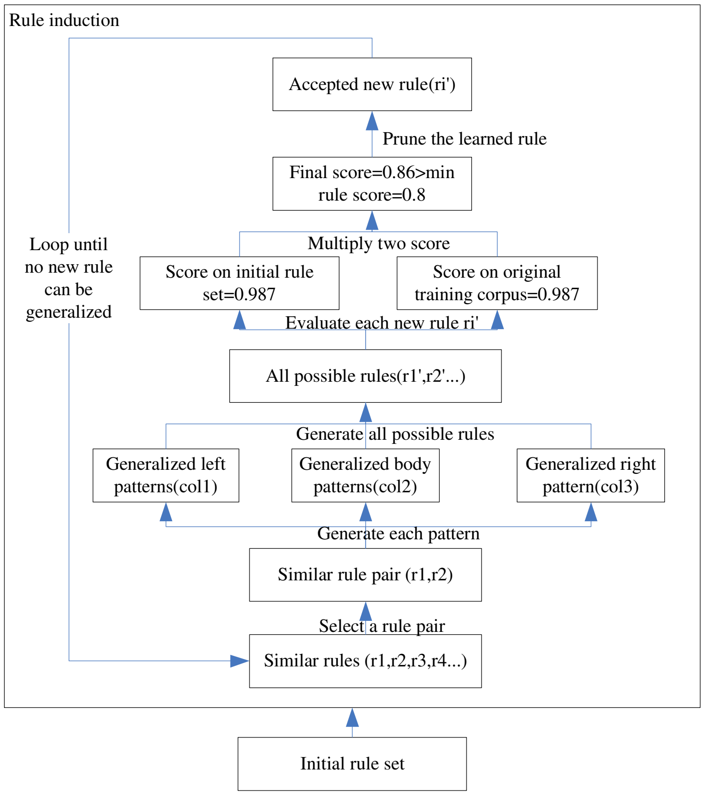

Rule induction
Accepted new rule(ri')
Prune the learned rule
Final score=0.86>min rule score=0.8
Multiply two score
Loop until no new rule can be generalized
Score on initial rule set=0.987
Score on original training corpus=0.987
Evaluate each new rule ri'
All possible rules(r1',r2'...)
Generate all possible rules
Generalized left patterns(col1)
Generalized body patterns(col2)
Generalized right pattern(col3)
Generate each pattern
Similar rule pair (r1,r2)
Select a rule pair
Similar rules (r1,r2,r3,r4...)
Initial rule set

Fig. 11. An example snippet of dependency graph recorded in iASA

Figure 11 shows a dependency graph fragment that records the generalization of a new rule. This procedure is the visualization of rule induction to give the user an insight look of the rule induction. SRL firstly uses MLRM algorithm to search for the most similar rules (r1, r2, r3, r4), and obtains a rule pair (r1,r2) for induction. Then, the rule generalization induces left patterns, body patterns and right patterns respectively. Each generalization returns a collection of possible patterns. After that, SRL combines the three pattern collections, and generates all possible rules. Next, SRL evaluates each rule on the initial rule set and the training corpus, and obtains the scores. For rule r1', the scores are 0.987 and 0.87 respectively. The final score 0.86 exceed the threshold. Therefore, SRL add the new rule (r1') into the learned rule set.

The dependency graph is constructed while the system is running. Each of the components contributes nodes and edges during the execution of the system. When the system generates rules or annotates new documents, the dependency graph is created at the same time.

6.2 Answer the Questions

In principle, a user may ask iASA many questions of following categories:

(1) Explain existing rule: "why is a rule induced as the final rule?" In essence, the user wants to know how it was induced and survived from the evaluation and pruning process.
(2) Explain absent rule: this is to answer the question: "Why is a certain rule not present in the output?"
(3) Explain rule score: "why is rule *x* scored higher than rule *y* in the output?". For such question, iASA gives the details of the two scores (i.e. *rulescore* and *realscore*), including their coverage, error count, missing count, precision, recall and F-measure. The score is the key point in deciding where the rule should be put in the output.
(4) Explain annotation: "which rule is an instance annotated by?" For a wrong annotation, the user wants to know which rule brings out the error. He wants to know the reason so as to modify the rule.

We now briefly describe how iASA generates explanations for the four kinds of predefined queries described above.

To answer the question "why is rule *x* present", iASA selects the slice of dependency graph that records the generation and processing of rule *x*.

To answer the question "why is rule *x* scored higher than rule *y*", iASA first searches for the two rules in the dependency graph. Then iASA compares the two slices of the dependency graph corresponding to *x* and *y*. When comparing the slices, it focuses on the places where the two rules are evaluated and scored. iASA outputs the details of the scores on the two rules. The details of the scores indicate to the user that the difference between the two rules. For example, the user can find from the scores in which step the rule *x* scored higher than the rule *y* so that it survives in the final rules.

To answer the question "why is rule *x* not present", iASA first examines the dependency graph to check whether rule *x* has been generated before. If it has, then iASA finds out where it has been eliminated, and searches for the places where the rule is scored. iASA outputs the scores of the rule on initial rule set, original training corpus, the final score, and the threshold. By comparing the scores with each other and with the threshold, the user can know why the rule is not present. For example, the reason might be the score of the rule is below the threshold.

If rule *x* has not been generated, then iASA checks whether or not the rule can be generalized from the initial rule set. We conduct the check by searching for whether there are rules that are covered by the rule. Success of the check indicates that the rule can be generalized (but did not). Then iASA checks why it was not generated. We conduct the check by tracing all rules that are covered by the rule. Those rules may be selected in different pair of similar rules for induction. iASA outputs all covered rules and presents the similar rule pairs that include the covered rules, and their similarity scores.

To answer the question "which rule is an instance annotated by?", iASA searches for the rule in the predecessors of annotated instance. At last, iASA outputs the rule.

By the explanation module, the user can take corresponding actions for improving the learned rules or the annotation results. For example, the user finds that a rule is not present in the learned rule set. He can first ask the explanation module the question, and the explanation returns the answer (e.g. the answer is: the rule is generated; however, it is pruned because its score is below the threshold). And then the user can choose to accept the rule.

7 Experiments and Discussion

7.1 Experimental Setup

In existing annotation systems, some are manual, some are based on GATE (its rules need to be predefined manually), and most of the semi-automatic semantic annotation systems exploit existing IE algorithms. Table 5 lists the relationships between IE algorithms and some semantic annotation systems.

For the semi-automatic semantic annotation system, its performance naturally depends on the IE algorithms that it exploits. Therefore, to evaluate iASA, we compare iASA with the IE algorithms used in the semantic annotation systems, e.g. LP^2 (LP^2 is an algorithm for adaptive Information Extraction from Web-related text that induces symbolic rules learning from a corpus tagged with SGML tags). We conducted the comparison between iASA and some other popular algorithms (e.g. Rapier, SRV, HMM, BWI, Whisk, etc).

Our experiments are performed on two standard tasks for adaptive IE: the CMU seminar announcements [20] and Austin job announcements [7][1].

CMU seminar task consists of 485 seminar announcements from Carnegie Mellon University. The announcements contain details of the upcoming seminars. Each seminar is annotated with unique *starting time*, *ending time*, *location* and possible multiple *speaker name*. We denote CMU as the data set of CMU seminar announcements.

The Austin job task consists of 300 newsgroup messages on job details in Austin area. The task has been required to annotate 17 elements (see table 7). We denote JOBS as the data set of Austin job announcements.

[1] http://www.isi.edu/info-agents/RISE/repository.html

Table 5. Relationships between semantic annotation systems and IE algorithms

SA Systems	IE algorithms
S-CREAM	LP^2
MnM	LP^2, Badger, Marmot, Crystal
SHOE	Manual
AeroDAML	AeroText, NLP
Annotea	Manual
KIM	GATE
SEAN	Syntactic and semantic structure learning
Protégé 2000	Manual
OntoMat-Annotizer	LP^2
Melita	LP^2
Artequakt	GATE
SemTag	TBD
SCORE	Name entity and relation learning

In experiments, we used a random 50:50 split of the two datasets and repeated 10 times. We used 50:50 split for facilitating the comparison because the results for BWI, RAPIER and LP^2, etc [33] use the same splits for each system.

All experiments use the strategy of dynamic window size and machine learning methods to conduct instance selection and missing instance prediction.

7.2 Evaluation Measures

A truly comprehensive comparison should compare each algorithm on the same dataset, using the same splits, and the same scoring system. Unfortunately, it is impossible to end up with a conclusive comparison of different algorithms using current published results. Different algorithms have been evaluated by slightly different methodologies [33].

In semantic annotation, two kinds of issues would be considered with respect to the evaluation: how to decide an instance is correct and how to count the correct/wrong instances.

For the first issue, we take a compromised approach of combining exact matches and partial matches. Exact match contributes a full score and partial match contributes a half score. For example, if the correct speaker is "Dr Jim Boshears, PhD" and iASA annotates "Dr Jim Boshears" as a speaker, this would be viewed as partial match and count as half a correct instance (0.5).

For the second issue, there exist two approaches: instance exact matching, value exact matching. The former one requires the system to annotate all possible instances. Thus if a document contains a *stime*'s instance which has two occurrences "1:00 PM" and "1 p.m", then the system is required to annotate them both. The second approach only compares the output annotations. In this case, it is sufficient to annotate *stime* either by "1:00 PM" or "1 p.m" as they refer to the same meaning. In this paper, we adopt the later approach to count the correct/wrong instances. As for the multiple instances referring to the same meaning, we only count one time.

In all the experiments, we conducted evaluations in terms of F-measure (β=1). The evaluation measure has been introduced in section 3.3.

7.3 Experimental Results

Table 6 shows the comparison between iASA and several algorithms on CMU. The columns respectively represent the Algorithm, F1-score on entity *starting time*, *ending time*, *location* and *speaker name*, and the average F1-score.

Table 6. Comparison of the seven methods on CMU (%)

Algorithm	stime	etime	speaker	location	Average
BWI	99.6	93.9	67.7	76.7	83.9
HMM	98.5	62.1	76.6	**78.6**	82.0
SRV	98.5	77.9	56.3	72.3	77.1
Rapier	93.4	**96.2**	53.0	72.7	77.3
Whisk	92.6	86.0	18.3	66.4	64.9
LP^2	99.0	95.5	**77.6**	75.0	**86.0**
iASA	**99.8**	95.2	75.7	76.5	85.8

As shown in table 6, we see that iASA outperforms most of the other algorithms (averagely +2.26% wrt BWI, +4.63% wrt HMM, +11.28% wrt SRV, +11.0% wrt Rapier, +32.2% wrt to Whisk), and is competitive with (LP)^2 (-0.2%).

In JOBS, It requires to annotate the seventeen kinds of information related to computer job (some are unique and some are not). We used half of the corpus to train, and the rest to test the learned rules. Table 7 shows the experimental results. The columns respectively represent the entity that is required to annotate, three algorithms (Rapier, BWI, and (LP)^2), and iASA.

Table 7. Comparison of the four methods on JOBS (%)

Entity	Rapier	BWI	(LP)2	iASA
Id	97.5	**100.0**	**100.0**	**100.0**
title	40.5	50.1	43.9	**89.1**
company	69.5	**78.2**	71.9	73.6
salary	67.4	-	62.8	**80.0**
recruiter	68.4	-	80.6	**91.3**
state	90.2	-	84.7	**91.5**
city	90.4	-	93.0	**95.6**
country	93.2	-	81.0	**96.6**
language	80.6	-	**91.0**	83.2
platform	72.5	-	80.5	**82.4**
application	69.3	-	**78.4**	73.8
area	42.4	-	**66.9**	55.3
req-years-e	67.1	-	68.8	**73.7**
des-years-e	**87.5**	-	60.4	66.7
req-degree	81.5	-	**84.7**	65.9
des-degree	72.2	-	65.1	**80.0**
post date	99.5	-	99.5	**100.0**
Average	75.1	-	84.1	**89.4**

We see averagely iASA significantly outperforms Rapier (by +19%) and $(LP)^2$ (by +6.3%) in terms of F1-measure. On the three entities that are available for BWI, iASA significantly outperforms it on *title* (+77.8%), and underperforms it on *company* (-5.9%).

7.4 Discussion

Intuitively, dynamic window size can optimize the learning scenario, instance selection can improve the precision by pruning the potential wrong annotations, and missing instance prediction can improve the recall of the annotation. We performed several special experiments for confirming the idea.

1. Dynamic window size vs. Fixed window size

In our experiments, some elements are not affected by the window size, while the others are sensitive to it. For the entities that are sensitive to window size, some prefer small window size while the others prefer large window size.

We give an experimental comparison of dynamic window size and fixed window size. We conducted the comparison on CMU. The result is shown in figure 12.

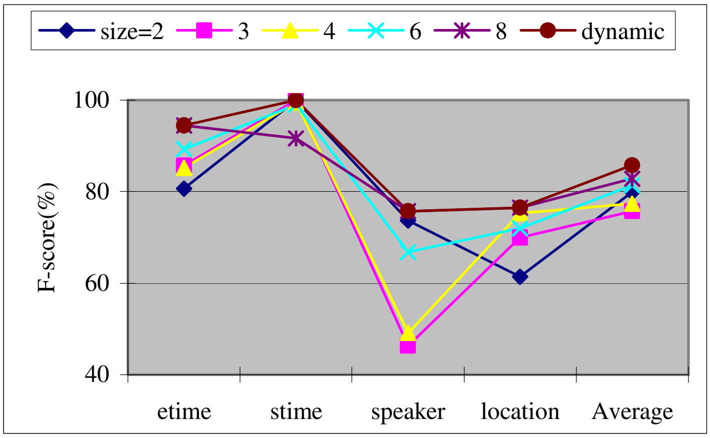


Fig. 12. Comparison between different fixed window size and dynamic window size on CMU

By fixed-window size, the best average result is 82.8% obtained by using fixed-window size 8, and the second is 81.4% when the window size is set to 6. The result of dynamic window size outperforms that of all the fixed window size (+3.6% than that of size=8, +5.4% than that of size=6).

We also see that for the entities that are sensitive to the window size, the improvements by dynamic strategy are significantly (e.g. for *speaker*, the improvements are respectively +54.2% and +63.5% compared to the results of size=4 and size=3).

We note that optimizing dynamic window size is a time consuming process, which limits its applications with large scale ontology.

2. Correct instance selection vs. High scored selection vs. Without selection

We conducted the experiment to test the performance of the proposed method for correct instance selection. We conducted the comparison of results by correct instance selection, high scored selection, and result without any selection. By high scored selection, we mean selecting the annotated instances by the highest scored rules. We conducted the comparison on JOBS. (On CMU, the two selection methods do not affect the results of the four entities.)

Table 8 shows the comparison. The columns respectively represent entity, F1-scores (F) without selection, with high scored selection and with correct instance selection. For short, we use ML to denote correct instance selection by machine learning; we use HS to denote high scored selection. In table 8, we also present the recall (R) and error rate (E) of the selection. Recall indicates how many correct instances are selected. Error rate indicates the accuracy of instance selection. The selection methods work on six entities in JOBS, and do not affect the other eleven entities. Therefore, in table 8, we list only the six entities and omit details of the other eleven entities.

Table 8. Correct instance selection vs. High scored (HS) selection in JOBS (%)

Entity	**Without selection**	**HS selection**			**ML selection**		
		R	**E**	**F**	**R**	**E**	**F**
title	77.1	33.3	0	82.1	74.2	0	89.1
platform	79.2	0	0	79.2	72.0	0	80.9
city	88.2	42.9	33.3	92.3	52.5	0	95.6
area	50.6	0	0	50.6	72.9	33.3	53.1
application	67.7	0	0	67.7	77.0	5.0	71.2
req-degree	57.2	0	0	57.2	50.0	12.5	65.9
Average	70.0	-		71.5	-		76.0

We see that ML based instance selection significantly outperforms high scored selection (+6.3% on average). The improvement over the result without selection is also significant (+8.6% on average). By high score selection, only two entities obtained improvements: title and city. By ML selection, all the six entities obtained improvements.

We also note that the recall of the ML based selection is still low (ranging from 50% to 77%) and errors are also induced by the wrong selection (e.g. *area*, *application*, and *req-degree*). This also means that we need further improve it.

3. ML based prediction vs. No prediction

We exploit machine learning methods to improve the recall of iASA. We conducted the comparison between results by machine learning based prediction and that without prediction. We conducted the comparison on JOBS. (On CMU, the prediction methods do not affect the results of the four entities.)

Table 9 shows the experiment results on JOBS. ML denotes the machine learning based method for prediction; E denotes the error rate of ML based prediction; F denotes the F1-score. The prediction methods work on four entities in JOBS, and do not affect the other thirteen entities. Therefore, in table 9, we list only the four entities and omit details of the other thirteen entities.

Table 9. ML based prediction vs. No prediction on JOBS (%)

Entity	No prediction	ML	
		E	F
language	75.7	0	83.2
platform	80.9	33.3	82.4
application	71.2	3.8	73.8
area	53.1	15	55.3
Average	70.1	-	73.7

By ML based prediction, we have observed improvement on the four entities (by +5.1% on average in terms of F1-measure). We have also observed that the prediction might slightly decrease the precision because of the wrong prediction.

On the other hand, we see that the prediction method only works on four entities in the two data sets. Because the instances of other entities (four entities in CMU and thirteen entities in JOBS) are not enumerative type and their instance values are too sparse. Moreover, the prediction introduces errors (ranging from 3.8% to 33.3%). Some errors (about 50%) imported from the fact that instance value of entity *platform* and *area* are similar. Such similarity confuses the prediction method.

4. More analysis

(1) From the experiments, we see that for the majority of the elements in the two tasks, iASA outperforms the other algorithms. On CMU, iASA averagely outperforms them (averagely +2.26% wrt BWI, +4.63% wrt HMM, +11.28% wrt SRV, +11.0% wrt Rapier, +32.2% wrt Whisk,), and is competitive with $(LP)^2$ (-0.2%). On JOBS, iASA averagely outperforms Rapier (by +19%) and $(LP)^2$ (by +6.3%).

(2) We conducted the analysis on each entity. On entities *stime*, *etime* in the CMU and *id*, *posting_date*, *platform* in JOBS, almost all algorithms perform well. These entities often have clear common linguistic information either in their contexts or in their body patterns. On the other hand, *location* and *speaker* are somewhat difficult for iASA. There may be two reasons. One is that these entities have little common linguistic information and their contexts are always inconsistent. The other lies in that several entities often appear in a document in a particular relationship which makes the situation suitable for learning them together. For example, the *stime* and *etime* in the CMU, and *title* with other elements (*area*, *country*, *state*, etc.) in the JOBS. This is the part of our future work for iASA.

(3) Considering the six elements that underperform LP^2 or Rapier, they can be classified into two groups: *des_years_e*, *req_degree* and *area*, *language*, *application*, *company*.

For the former two entities, we found that the two entities are difficult distinguished even by human. The context and body pattern for them are very similar. Analysis of syntactic structure and semantic structure can help to construct long distant context and semantic context (e.g. *subject predicate object*) and may correct the errors.

For the later four elements, the precisions are acceptable but the recalls are low. By experimental analysis, we found that many learned rules only are comprised of body patterns (i.e. left pattern and right pattern are null, e.g. “C++” for *language*), which means that such annotations heavily depend on whether the body patterns appear in the training samples. Two approaches may be useful to deal with such problems: increasing training samples and creating domain thesaurus. More training samples can induce the rules covering more body patterns. But more training samples also means more manual works. Creating domain thesaurus means to construct a word list for the entity (e.g. *language*) to assist the annotation.

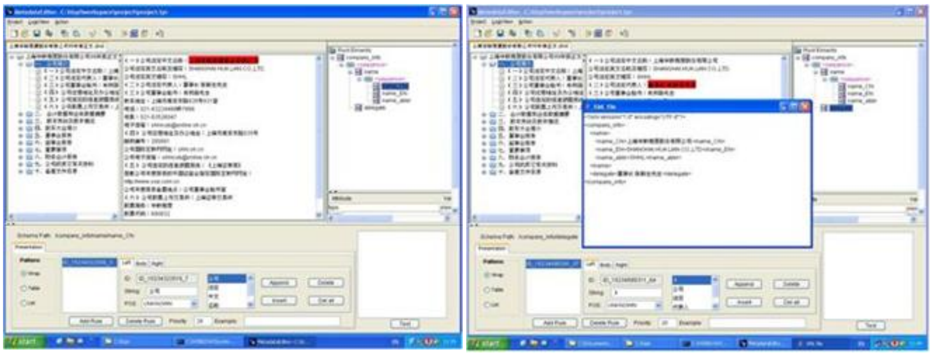
7.5 Applications

We have applied iASA to a practical application: TIPSI.

TIPSI is a project from Tsinghua-ITF Co-Lab. In TIPSI, we aim to extract and annotate the information in company annual reports for Stock Exchange.

The company annual report is a semi-structured document. In TIPSI, we first extract the logic structure from annual report. We conducted the logic structure extraction by a logic structure extractor. See [50] for details. Then, the document is organized into a tree structure. It is similar to the ‘Document Map’ in Office Word. Next, for each node in the logic structure, we applied iASA to annotate it according to the predefined ontology of company annual report.

For example, figure 13 (a) shows the user interface of TIPSI and figure 13 (b) shows the output of annotation. There are four main fields on the user interface. The top-left window shows the document logic structure. The mid window shows the text content of the selected node in logic structure. The right-top window shows the predefined ontology. The bottom window is a rule management tool.



(a) (b)

Fig. 13. Screenshots of TIPSI

In training stage, when user selects the text on the mid window, iASA automatically carries out tokenization and NLP processing, and then generates an initial rule. The rule can be incrementally added into the learned rule set and can be added to the initial rule set to retrain the whole rule set. If the rule is already covered by some other rules in the learned rule set, the system will prompt the user about that. In this way, the rule learning is a user interaction stage. User can also manually create a new rule or correct learned rules.

In annotation stage, iASA annotates documents by using the learned rules. It also records the occurring position of each instance. And the user can be navigated to instances by selecting the concept or property in the ontology. When the user selects a concept/property in the ontology, the system will highlight the annotated instance(s). Figure 13 (a) shows the scenario. In the mid window, the highlighted text is an instance of company Chinese name.

After annotation, the user can choose to output the annotation results. Figure 13 (b) shows the annotation results by a popup window. The results are presented in XML format according to the requirement of the project.

We are also trying to apply the tool into Contact Search on internet.

In Contact Search, we aim to annotate the contact information for a given person. The user inputs a person name. The system submits the person's name to Google. Then we use a text classifier to identify the web page that contains the contact information of the person. After that, we apply the iASA to annotate the contact information. The contact information includes: person name, email, telephone, fax, homepage, address, and job title. The project is still ongoing and the preliminary results show that the tool is promising.

8 Related Works

In this section, we introduce the related works from four aspects: Knowledge Acquisition Frameworks, Annotation Framework, (Semi-)Automated Annotation with Support from Information Extraction, and Information Extraction. There are a number of available systems that address these four aspects. A complete review of this subject is therefore outside the scope of this paper. We present some of them through their principles and availabilities.

8.1 Knowledge Acquisition Frameworks

Several systems are designed to allow for knowledge acquisition and to use knowledge markups in semantic web, for example: Protégé-2000 [18], WebKB [35], SHOE [26] and Artequakt [1]. These four systems all start from providing manual mark-up by editors.

Protégé-2000 is a tool which supports knowledge acquisition. But it doesn't support managing and annotating the web pages.

WebKB uses conceptual graphs to represent the semantic content of Web documents. It embeds conceptual graph statements into HTML pages. Essentially they offer a web template-based knowledge acquisition framework.

SHOE is one of the earliest systems for adding semantic annotations to web pages. SHOE Knowledge Annotator allows users to markup pages manually in SHOE guided by ontologies available locally or via a URL. These marked pages can be reasoned about by SHOE-aware tools such as SHOE Search. Such tools are described in [30, 48].

The Artequakt project links a knowledge extraction tool with ontology to achieve knowledge support and to guide information extraction. The extraction tool searches for online documents and extracts knowledge that matches the given classification structure. Knowledge extraction is further enhanced by using a lexicon-based term expansion mechanism that provides extended ontology terminology [1].

8.2 Annotation Frameworks

There are a number of systems designed particularly for annotation, for example: Annotea [27], Ontobroker [19], OntoMat-Annotizer, SEAN [37], etc.

Annotea is a Web-based shared annotation system based on a general-purpose open resource description framework (RDF) infrastructure. In Annotea, the annotations are modeled as a class of metadata. Annotations are viewed as statements made by an author about a Web document.

Ontobroker facilitates manual annotation of HTML documents with semantic markups.

SEAN automatically discovers and labels concept instances in template-based, content-rich HTML documents according to an ontology. It combines structural and semantic analysis for annotation. SEAN focuses on well-organized documents, for example documents generated from databases.

8.3 (Semi-)Automated Annotation with Support from Information Extraction

Recently, efforts have been put into automating the annotation task by using machine learning methods. The principal tool is "wrapper" (see [11, 29, 31]). The Semantic Annotation systems which use IE algorithms can be referred to Table 5.

For example, S-CREAM [24], MnM [49] and Melita [10] are three systems exploiting IE algorithm LP^2 to automate the procedure of annotation.

S-CREAM is a comprehensive framework for creating annotations, relational metadata in the semantic web, including tools for both manual and semi-automatic annotation of pages. It also comprises inference services, crawler, document management system, ontology guidance/fact browser, document editors/viewers, and a meta ontology.

MnM produces semantic markups with the support from IE algorithm. Besides LP^2, it also integrates other IE components (Marmot, Badger, Crystal) from the University of Massachusetts at Amherst (UMass). It allows the semi-automatic population of ontology with metadata.

Melita is a tool for defining and developing automatic ontology-based annotation services that provides different views over the task. It provides manual and semi-automatic annotation, as well as a rule editor for IE experts to edit the annotation rules.

AeroDAML [28] is a tool which takes ontology as metadata and automatically produces a semantic annotation using NLP techniques. It supports only DAML language.

The KIM platform provides semantic annotation, indexing, retrieval services and infrastructure. It performs information extraction based on ontology and a massive knowledge base [42]. The information extraction process in KIM is based on the GATE platform. GATE's pattern-matching grammars have been modified so as to handle entity class information and to allow generalization of the rules. But, GATE does not provide the feature for learning the annotation rules.

SCORE Enhancement Engine (SEE) supports heterogeneous contents, followed by an automatic classification with extraction of context relevant, domain-specific metadata. Extraction of semantic metadata includes not only the identification of the relevant entities, but also the relationships within the context of relevant ontology. It also presents an approach to automatic semantic annotation [22].

Li et al propose to combine natural language understanding with learning to automatically generate annotations for specific domains [34]. They aim to learn the syntactic structures from the sentences.

SemTag aims to annotate very large number of pages with terms from a standard ontology in an automated fashion based on disambiguation annotation [17]. SemTag operates as a centralized application with access to the entire database and associated metadata. SemTag manipulates the text linking in web page to its correct resource by disambiguation technology.

Esperonto has an annotation service that helps content providers bridge the gap between the current Web and the Semantic Web. It uses wrapper technology to upgrade content to Semantic Web content [6, 25].

So far, existing systems focus on different aspects that are concerned with semantic annotation. Comparing with the above methods, three features make iASA different: (1) similarity based rule induction; (2) using machine learning to refine the annotation; (2) explanation method by visualizing the main stages in rule induction and annotation procedure. Existing works improve the recall of annotation by combining data mining and IE techniques. They are similar to the missing instance prediction in iASA. In the experiments of [39], F1-measure could be improved about 3% by using soft matching mined rules when tested on 150 documents. It is difficult to conduct the comparison of the method and our method. The reason lies in that, their experiments were based on BWI algorithm (an IE algorithm), and their best F1-measure result was only 45% that is below the average in our experiments.

8.4 Information Extraction Technologies

In information extraction, given a sequence of instances, we identify and pull out a sub sequence of the input that represents the information we are interested in. Hidden Markov Model [21, 44], Maximum Entropy Model [3, 8], Maximum Entropy Markov Model [36], Support Vector Machines [13], Conditional Random Field [32], and Voted Perceptron [12] are widely used information extraction models.

Information extraction has been applied, for instance, to named entity recognition [51], table extraction [41], metadata extraction from research paper [23, 40].

9 Conclusions

In this paper, we have investigated the problem of semantic annotation. We have proposed a tool, called iASA, which learns to automatically annotate web documents according to an ontology. By using similarity based rule induction, we have been able to improve the rule learning procedure. We have tried to improve the annotation results by making use of machine learning methods. Finally, we have developed an explanation module to express the nature of the learner and annotator to users. Experimental results show that our approach can significantly outperform most of the existing wrapper methods. By the analysis of the experimental results, we observed that the proposed methods (including: correct instance selection and missing instance prediction) work well.

As the future work, we plan to make further improvement on the annotation accuracy. We also want to apply the annotation method to other annotation applications. Apart from that, several challenges for semantic annotation, also being our research interests, including: (1) Using active learning to make iASA more adaptive to new documents. By active learning, we can prepare training documents more efficiently and more effectively. (2) Making use of the annotation. The goal of semantic annotation is to improve the efficiency of obtaining information in web environment. Therefore, a friendly and flexible mechanism for using the annotation is necessary. (3) Combining ontology mapping with semantic annotation. In order to reach interoperability in the heterogeneous environment of semantic web, a system for integrating ontology mapping and semantic annotation is required. (4) Multilingual annotation. Multilingual annotation is also important given the fact that Chinese websites are increasing exponentially, which could be a future direction to go as well.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No. 60443002.

We offer thanks to anonymous reviewers for their valuable comments.

We express our great sadness and sorrow regarding Professor Hongjun Lu, who passed away while this paper was in preparation.

References

1. H. Alani, S. Kim, D. Millard, M. Weal, W. Hall, P. Lewis, and N. Shadbolt. Automatic Ontology-Based Knowledge Extraction from Web Documents. IEEE Intelligent Systems. 2003, 18(1):14-21.
2. R. Benjamins, J. Contreras. White Paper Six Challenges for the Semantic Web. Intelligent Software Components. Intelligent software for the networked economy (isoco). April, 2002.
3. A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra, A Maximum Entropy Approach to Natural Language Processing. In Computational Linguistics. 22, 1996:39-71
4. T. Berners-Lee, M. Fischetti, and M. L. Dertouzos. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web. 1999.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, vol. 284, May 2001:34-43

6. P. Buitelaar and T. Declerck. Linguistic Annotation for the Semantic Web. In Annotation for the Semantic Web, Frontiers in Artificial Intelligence and Applications Series, Vol. 96, IOS Press, 2003.
7. M. E. Califf. Relational Learning Techniques for Natural Language Information Extraction. Ph.D. thesis. University of Texas, Austin. 1998
8. H. L. Chieu and H. T. Ng. A Maximum Entropy Approach to Information Extraction from Semi-Structured and Free Text. In Eighteenth national conference on Artificial intelligence, 2002.
9. F. Ciravegna. $(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts. In Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining held in conjunction with 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, Usa, August 2001.
10. F. Ciravegna, A. Dingli, J. Iria, and Y. Wilks. Multi-strategy Definition of Annotation Services in Melita. In ISWC'03 Workshop on Human Language Technology for the Semantic Web and Web Services, 2003:97-107
11. W. Cohen, L. Jensen, A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents, In Proceedings of the Workshop on Adaptive Text Extraction and Mining (IJCAI'01), 2001.
12. M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In Proceedings of the Conference on Empirical Methods in NLP, 2002.
13. C. Cortes and V. Vapnik. Support-vector networks. Machine Learning 20, 1995:273-297
14. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics, 2002.
15. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation. Available at http://www.w3.org/TR/owl-ref/. 10 February 2004.
16. R. Dhamankar, Y. Lee, A.H. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. SIGMOD 2004 June 1318, 2004, Paris, France.
17. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. S. McCurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. A Case for Automated Large-scale Semantic Annotation. Journal of Web Semantics: Science, Services and Agents on the World Wide Web. Published by Elsevier B.V. July, 2003:115-132
18. H. Eriksson, R. Fergerson, Y. Shahar, and M. Musen. Automatic Generation of Ontology Editors. In Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff Alberta, Canada, 1999.
19. D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: Or how to enable intelligent access to the WWW. In Proceedings of 11th Banff Knowledge Acquisition for Knowledge-Based SystemsWorkshop, Banff, Canada, 1998.
20. D. Freitag and N. Kushmerick. Boosted Wrapper Induction. In Proceedings of 17th National Conference on Artificial Intelligence. 2000.
21. Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. Machine Learning, 1997, 29:245-273

22. B. Hammond, A. Sheth, and K. Kochut, Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content, in Real World Semantic Web Applications, V. Kashyap and L. Shklar, Eds., IOS Press. December 2002:29-49
23. H. Han, L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. Fox. Automatic Document Metadata Extraction Using Support Vector Machine. In Proceedings of Joint Conference on Digital Libraries (JCDL 2003). 2003:37-48
24. S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM—Semi-automatic Creation of Metadata, In Proceedings of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002), Siguenza, Spain, 2002:358-372.
25. S. Handschuh and S. Staab. Annotation for the Semantic Web. Volume 96 Frontiers in Artificial Intelligence and Applications. New IOS Publication. 2003
26. J. Heflin and J. Hendler. Searching the Web with SHOE. In Proceedings of AAAI-2000 Workshop on AI for Web Search, Austin, Texas, 2000.
27. J. Kahan and M. R. Koivunen. Annotea: an Open RDF Infrastructure for Shared Web Annotations. In Proceedings of World Wide Web, 2001:623-632.
28. P. Kogut and W. Holmes. AeroDAML: Applying Information Extraction to Generate DAML Annotations from Web Pages. 2001.
29. N. Kushmerick, D.S. Weld, and R.B. Doorenbos. Wrapper Induction for Information Extraction. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Nagoya, Japan. 1997: 729-737.
30. T. Leonard and H. Glaser. Large Scale Acquisition and Maintenance from the Web without Source Access. http://www.semannot2001.aifb.uni-karlsruhe.de/positionpapers/Leonard.pdf. 2001.
31. K. Lerman, C. Knoblock, S. Minton, Automatic data extraction from lists and tables in web sources, in: IJCAI-2001 Workshop on Adaptive Text Extraction and Mining, Seattle, WA, August 2001.
32. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In ICML 01, 2001.
33. A. Lavelli, M. Califf, F. Ciravegna, F. Freitag, D. Giuliano, C. Kushmerick, and N. Romano. A Critical Survey of the Methodology for IE Evaluation. In Proceedings of the 4th International Conference on Language Resources and Evaluation. 2004
34. J. Li and Y. Yu. Learning to Generate Semantic Annotation for Domain Specific Sentences. In Proceedings of the Knowledge Markup and Semantic Annotation Workshop in K-CAP 2001, Victoria, BC, 2001.
35. P. Martin and P. Eklund. Embedding Knowledge in Web Documents. In Proceedings of the 8th International World Wide Web Conf. (WWW'8), Toronto, Elsevier Science B.V. May 1999:1403-1419
36. A. McCallum, D. Freitag, and F. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation, In Proceedings of the ICML Coference, 2000.
37. S. Mukherjee, G. Yang, and I. V. Ramakrishnan. Automatic Annotation of Content-Rich HTML Documents: Structural and Semantic Analysis. In Second International Semantic Web Conference (ISWC), Sanibel Island, Florida, October 2003.
38. I. Muslea. Active Learning with Multiple Views. Ph.D. dissertation. (USC, 2002)
39. U. Y. Nahm and R. J. Mooney. Using Soft-Matching Mined Rules to Improve Information Extraction. In Proceedings of the AAAI-2004 Workshop on Adaptive Text Extraction and Mining (ATEM-2004), San Jose, CA, July, 2004:27-32.

40. F. Peng and A. McCallum. Accurate Information Extraction from Research Papers using Conditional Random Fields. In Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL), 2004.
41. D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table Extraction Using Conditional Random Fields. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval, 2003.
42. B. Popov, A. Kiryakov, D. Manov, A. Kirilov, D. Ognyanoff, and M. Goranov. Towards Semantic Web Information Extraction. In Proceedings of the ISWC'03 Workshop on Human Language Technology for the Semantic Web and Web Services, 2003.1-21
43. C. Schaffer. Selecting a Classification method by Cross-Validation. Machine Learning, 13(1), 1993:135-143
44. K. Seymore, A. McCallum, and R. Rosenfeld. Learning Hidden Markov Model Structure for Information Extraction. In Proceedings of AAAI'99 Workshop on Machine Learning for Information Extraction. 1999.
45. S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. Machine Learning. Jan, 1999:1-44
46. V. W. Soo, C. Y. Lee, C. –C. Li, S. L. Chen, and C. Chen. Automated Semantic Annotation and Retrieval Based on Sharable Ontology and Case-based Learning Techniques. In Proceedings of the 2003 Joint Conference on Digital Libraries. 2003 IEEE.
47. V. Vapnik. Statistical Learning Theroy. Springer Verlage, New York, 1998.
48. M. Vargas-Vera, E. Motta, J. Domingue, S. Buckingham Shum, and M. Lanzoni. Knowledge Extraction by Using an Ontology-based Annotation Tool. In Proceedings of K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation, Victoria, BC, Canada, October 2001.
49. M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semiautomatic and Automatic Support for Semantic Markup, In Proceedings of the 13th International Conference on Knowledge Engineering and Management (EKAW 2002), Siguenza, Spain, 2002.
50. K. Zhang, P. Xu, and J. Li. Optimal Hierarchical Clustering based Logic Structure Extraction. Journal of Tsinghua Science and Technology. 2005.
51. L. Zhang, Y. Pan, and T. Zhang. Recognising and using named entities: Focused named entity recognition using machine learning. In Proceedings of the SIGIR'04, 2004.

A Survey of Schema-Based Matching Approaches*

Pavel Shvaiko[1] and Jérôme Euzenat[2]

[1] University of Trento, Povo, Trento, Italy
pavel@dit.unitn.it
[2] INRIA, Rhône-Alpes, France
Jerome.Euzenat@inrialpes.fr

Abstract. Schema and ontology matching is a critical problem in many application domains, such as semantic web, schema/ontology integration, data warehouses, e-commerce, etc. Many different matching solutions have been proposed so far. In this paper we present a new classification of schema-based matching techniques that builds on the top of state of the art in both schema and ontology matching. Some innovations are in introducing new criteria which are based on (i) general properties of matching techniques, (ii) interpretation of input information, and (iii) the kind of input information. In particular, we distinguish between approximate and exact techniques at schema-level; and syntactic, semantic, and external techniques at element- and structure-level. Based on the classification proposed we overview some of the recent schema/ontology matching systems pointing which part of the solution space they cover. The proposed classification provides a common conceptual basis, and, hence, can be used for comparing different existing schema/ontology matching techniques and systems as well as for designing new ones, taking advantages of state of the art solutions.

1 Introduction

Matching is a critical operation in many application domains, such as semantic web, schema/ontology integration, data warehouses, e-commerce, query mediation, etc. It takes as input two schemas/ontologies, each consisting of a set of discrete entities (e.g., tables, XML elements, classes, properties, rules, predicates), and determines as output the relationships (e.g., equivalence, subsumption) holding between these entities.

Many diverse solutions to the matching problem have been proposed so far, e.g., [2, 5, 21, 23, 40, 46, 49, 50, 52, 57, 76]. Good surveys through the recent years are provided in [39, 62, 75]; while the major contributions of the last decades are presented in [3, 41, 42, 66]. The survey of [39] focuses on current state of the art in ontology matching. Authors review recent approaches, techniques and tools. The survey of [75] concentrates on approaches to ontology-based information integration and discusses general matching approaches that are used in information integration systems. However, none of the above mentioned surveys provide a comparative review of the existing ontology matching techniques and systems. On the contrary, the survey of [62] is devoted to a classification of database schema matching approaches and a comparative

* For more information on the topic (e.g., tutorials, relevant events), please visit the Ontology Matching web-site at www.OntologyMatching.org

review of matching systems. Notice that these three surveys address the matching problem from different perspectives (artificial intelligence, information systems, databases) and analyze disjoint sets of systems.

This paper aims at considering the above mentioned works together, taking in to account some novel schema/ontology matching approaches, and at providing a common conceptual basis for their analysis. Although, there is a difference between schema and ontology matching problems (see next section for details), we believe that techniques developed for each of them can be of a mutual benefit. Thus, we bring together and discuss systematically recent approaches and systems developed in schema and ontology matching domains. We present a new classification of schema/ontology matching techniques that builds on the work of [62] augmented in [29, 67] and [28]. Some innovations are in introducing new criteria which are based on (i) general properties of matching techniques, (ii) interpretation of input information, and (iii) the kind of input information. In particular, we distinguish between approximate and exact techniques at schema-level; and syntactic, external, and semantic techniques at element- and structure-level. Based on the classification proposed we provide a comparative review of the recent schema/ontology matching systems pointing which part of the solution space they cover. In this paper we focus only on schema-based solutions, i.e., matching systems exploiting only schema-level information, not instance data.

The rest of the paper is organized as follows. Section 2 provides, via an example, the basic motivations and definitions to the schema/ontology matching problem. Section 3 discusses possible matching dimensions. Section 4 introduces a classification of elementary automatic schema-based matching techniques and discusses in detail possible alternatives. Section 5 provides a vision on classifying matching systems. Section 6 overviews some of the recent schema/ontology matching solutions in light of the classification proposed pointing which part of the solution space they cover. Section 7 reports some conclusions and discusses the future work.

2 The Matching Problem

2.1 Motivating Example

To motivate the matching problem, let us use two simple XML schemas that are shown in Figure 1 and exemplify one of the possible situations which arise, for example, when resolving a schema integration task.

Suppose an e-commerce company needs to finalize a corporate acquisition of another company. To complete the acquisition we have to integrate databases of the two companies. The documents of both companies are stored according to XML schemas S and S' respectively. Numbers in boxes are the unique identifiers of the XML elements. A first step in integrating the schemas is to identify candidates to be merged or to have taxonomic relationships under an integrated schema. This step refers to a process of schema matching. For example, the elements with labels Office_Products in S and in S' are the candidates to be merged, while the element with label Digital_Cameras in S' should be subsumed by the element with label Photo_and_Cameras in S. Once the correspondences between two schemas have been determined, the next step has to generate

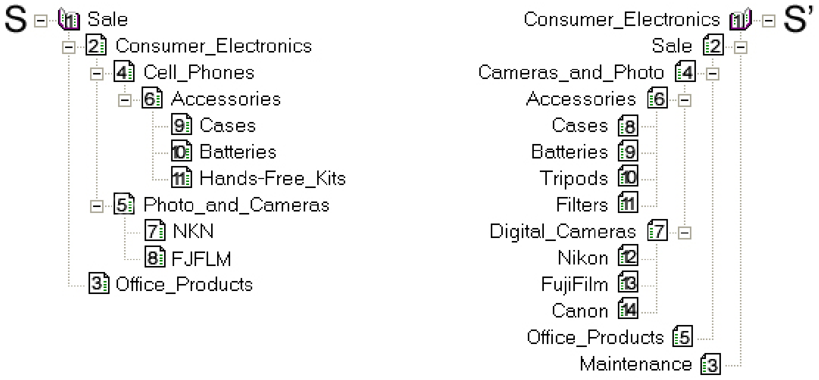


Fig. 1. Two XML schemas

query expressions that automatically translate data instances of these schemas under an integrated schema.

2.2 Schema Matching vs Ontology Matching

Many different kinds of structures can be considered as data/conceptual models: description logic terminologies, relational database schemas, XML-schemas, catalogs and directories, entity-relationship models, conceptual graphs, UML diagrams, etc. Most of the work on matching has been carried out among (i) database schemas in the world of information integration, (ii) XML-schemas and catalogs on the web, and (iii) ontologies in knowledge representation. Different parties, in general, have their own steadfast preferences for storing data. Therefore, when coordinating/integrating data among their information sources, it is often the case that we need to match between various data/conceptual models they are sticked to. In this respect, there are some important differences and commonalities between schema and ontology matching. The key points are:

– Database schemas often do not provide explicit semantics for their data. Semantics is usually specified explicitly at design-time, and frequently is not becoming a part of a database specification, therefore it is not available [56]. Ontologies are logical systems that themselves obey some formal semantics, e.g., we can interpret ontology definitions as a set of logical axioms.
– Ontologies and schemas are similar in the sense that (i) they both provide a vocabulary of terms that describes a domain of interest and (ii) they both constrain the meaning of terms used in the vocabulary [37, 70].
– Schemas and ontologies are found in such environments as the Semantic Web, and quite often in practice, it is the case that we need to match them.

On the one side, schema matching is usually performed with the help of techniques trying to guess the meaning encoded in the schemas. On the other side, ontology matching systems (primarily) try to exploit knowledge explicitly encoded in the ontologies.

In real-world applications, schemas/ontologies usually have both well defined and obscure labels (terms), and contexts they occur, therefore, solutions from both problems would be mutually beneficial.

2.3 Problem Statement

Following the work in [11, 27], we define a *mapping element* as a 5-uple: $\langle id, e, e', n, R\rangle$, where

- id is a unique identifier of the given mapping element;
- e and e' are the entities (e.g., tables, XML elements, properties, classes) of the first and the second schema/ontology respectively;
- n is a *confidence measure* in some mathematical structure (typically in the [0,1] range) holding for the correspondence between the entities e and e';
- R is a relation (e.g., *equivalence* (=); *more general* ($\sqsupseteq$); *disjointness* ($\perp$); *overlapping* ($\sqcap$)) holding between the entities e and e'.

An *alignment* is a set of mapping elements. The *matching* operation determines the alignment (A') for a pair of schemas/ontologies (o and o'). There are some other parameters which can extend the definition of the matching process, namely: (i) the use of an input alignment (A) which is to be completed by the process; (ii) the matching parameters, p (e.g., weights, thresholds); and (iii) external resources used by the matching process, r (e.g., thesauri); see Figure 2.

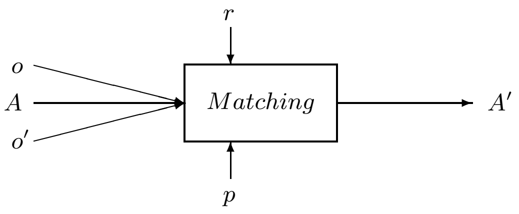


Fig. 2. The matching process

For example, in Figure 1, according to some matching algorithm based on linguistic and structure analysis, the confidence measure (for the fact that the equivalence relation holds) between entities with labels Photo_and_Cameras in S and Cameras_and_Photo in S' could be 0.67. Suppose that this matching algorithm uses a threshold of 0.55 for determining the resulting alignment, i.e., the algorithm considers all the pairs of entities with a confidence measure higher than 0.55 as correct mapping elements. Thus, our hypothetical matching algorithm should return to the user the following mapping element: $\langle id_{5,4}, Photo\_and\_Cameras, Cameras\_and\_Photo, 0.67, =\rangle$. However, the relation between the same pair of entities, according to another matching algorithm which is able to determine that both entities mean the same thing, could be exactly the equivalence relation (without computing the confidence measure). Thus, returning to the user $\langle id_{5,4}, Photo\_and\_Cameras, Cameras\_and\_Photo, n/a, =\rangle$.

2.4 Applications

Matching is an important operation in traditional applications, such as information integration, data warehousing, distributed query processing, etc. Typically, these applications are characterized by structural data/conceptual models, and are based on a design time matching operation, thereby determining alignment (e.g., manually or semi-automatically) as a prerequisite of running the system.

There is an emerging line of applications which can be characterized by their dynamics (e.g., agents, peer-to-peer systems, web services). Such applications, on the contrary to traditional ones, require a run time matching operation and take advantage of more "explicit" conceptual models.

Below, we first discuss an example of a traditional application, namely, catalog integration. Then, we focus on emergent applications, namely, peer-to-peer (P2P) databases, agent communication, and web services integration.

Catalog Integration. In B2B applications, trade partners store their products in electronic catalogs. Catalogs are tree-like structures, namely concept hierarchies with attributes. Typical examples of catalogs are product directories of www.amazon.com, www.ebay.com, etc. In order for a private company to participate in the marketplace (e.g., eBay), it is used to determine correspondences between entries of its catalogs and entries of a single catalog of a marketplace. This process of mapping entries among catalogs is referred to the catalog matching problem, see [12]. Having identified the correspondences between the entries of the catalogs, they are further analyzed in order to generate query expressions that automatically translate data instances between the catalogs, see, for example, [74]. Finally, having aligned the catalogs, users of a marketplace have a unified access to the products which are on sale.

P2P Databases. P2P networks are characterized by an extreme flexibility and dynamics. Peers may appear and disappear on the network, their databases are autonomous in their language, contents, how they can change their schemas, and so on. Since peers are autonomous, they might use different terminology, even if they refer to the same domain of interest. Thus, in order to establish (meaningful) information exchange between peers, one of the steps is to identify and characterize relationships between their schemas. Having identified the relationships between schemas, next step is to use these relationships for the purpose of query answering, for example, using techniques applied in data integration systems, namely Local-as-View (LAV), Global-as-View (GAV), or Global-Local-as-View (GLAV) [43]. However, P2P applications pose additional requirements on matching algorithms. In P2P settings an assumption that all the peers rely on one global schema, as in data integration, can not be made, because the global schema may need to be updated any time the system evolves, see [36]. Thus, if in the case of data integration schema matching operation can be performed at design time, in P2P applications peers need coordinating their databases on the fly, therefore requiring a run time schema matching operation.

Agent Communication. Agents are computer entities characterized by autonomy and capacity of interaction. They communicate through speech-act inspired languages which determine the ”envelope” of the messages and enable agents to position them within a particular interaction context. The actual content of messages is expressed in knowledge representation languages and often refer to some ontology. As a consequence, when two autonomous and independently designed agents meet, they have the possibility of exchanging messages, but little chance to understand each others if they do not share the same content language and ontology. Thus, it is necessary to provide the possibility for these agents to match their ontologies in order to either translate their messages or integrate bridge axioms in their own models, see [73]. One solution to this problem is to have an ontology alignment protocol that can be interleaved with any other agent interaction protocol and which could be triggered upon receiving a message expressed in an alien ontology. As a consequence, agents meeting each other for the first time and using different ontologies would be able to negotiate the matching of terms in their respective ontologies and to translate the content of the message they exchange with the help of the alignment.

Web Services Integration. Web services are processes that expose their interface to the web so that users can invoke them. Semantic web services provide a richer and more precise way to describe the services through the use of knowledge representation languages and ontologies. Web service discovery and integration is the process of finding a web service able to deliver a particular service and composing several services in order to achieve a particular goal, see [59]. However, semantic web services descriptions have no reasons to be expressed by reference to exactly the same ontologies. Henceforth, both for finding the adequate service and for interfacing services it will be necessary to establish the correspondences between the terms of the descriptions. This can be provided through matching the corresponding ontologies. For instance, if some service provides its output description in some ontology and another service uses a second ontology for describing its input, matching both ontologies will be used for (i) checking that what is delivered by the first service matches what is expected by the second one, (ii) verifying preconditions of the second service, and (iii) generating a mediator able to transform the output of the first service in order to be input to the second one.

In some of the above mentioned applications (e.g., two agents meeting or looking for the web services integration) there are no instances given beforehand. Thus, it is necessary to perform matching without them, based only on schema-level information.

3 The Matching Dimensions

There are many independent dimensions along which algorithms can be classified. As from Figure 2, we may classify them according to (i) input of the algorithms, (ii) characteristics of the matching process, and (iii) output of the algorithms. Let us discuss them in turn.

Input Dimensions. These dimensions concern the kind of input on which algorithms operate. As a first dimension, algorithms can be classified depending on the data /

conceptual models in which ontologies or schemas are expressed. For example, the Artemis [13] system supports the relational, OO, and ER models; Cupid [46] supports XML and relational models; QOM [26] supports RDF and OWL models. A second possible dimension depends on the kind of data that the algorithms exploit: different approaches exploit different information of the input data/conceptual models, some of them rely only on schema-level information (e.g., Cupid [46], COMA [21]), others rely only on instance data (e.g., GLUE [23]), or exploit both, schema- and instance-level information (e.g., QOM [26]). Even with the same data models, matching systems do not always use all available constructs, e.g., S-Match [34] when dealing with attributes discards information about datatypes (e.g., string, integer), and uses only the attributes names. In general, some algorithms focus on the labels assigned to the entities, some consider their internal structure and the type of their attributes, and some others consider their relations with other entities (see next section for details).

Process Dimensions. A classification of the matching process could be based on its general properties, as soon as we restrict ourselves to formal algorithms. In particular, it depends on the *approximate* or *exact* nature of its computation. Exact algorithms compute the absolute solution to a problem; approximate algorithms sacrifice exactness to performance (e.g., [26]). All the techniques discussed in the remainder of the paper can be either approximate or exact. Another dimension for analyzing the matching algorithms is based on the way they interpret the input data. We identify three large classes based on the intrinsic input, external resources, or some semantic theory of the considered entities. We call these three classes *syntactic*, *external*, and *semantic* respectively; and discuss them in detail in the next section.

Output Dimensions. Apart from the information that matching systems exploit and how they manipulate it, the other important class of dimensions concerns the form of the result they produce. The form of the alignment might be of importance: is it a one-to-one correspondence between the schema/ontology entities? Has it to be a final mapping element? Is any relation suitable?

Other significant distinctions in the output results have been indicated in [32]. One dimension concerns whether systems deliver a graded answer, e.g., that the correspondence holds with 98% confidence or 4/5 probability; or an all-or-nothing answer, e.g., that the correspondence definitely holds or not. In some approaches correspondences between schema/ontology entities are determined using distance measures. This is used for providing an alignment expressing equivalence between these entities in which the actual distance is the ground for generating a confidence measure in each correspondence, usually in [0,1] range, see, for example, [29, 46]. Another dimension concerns the kind of relations between entities a system can provide. Most of the systems focus on equivalence (=), while a few other are able to provide a more expressive result (e.g., equivalence, subsumption ($\sqsubseteq$), incompatibility ($\perp$), see for details [12, 33]).

There are many dimensions that can be taken into account when attempting at classifying matching methods. In the next section we present a classification of elementary techniques that draws simultaneously on several such criteria.

4 A Retained Classification of Elementary Schema-Based Matching Approaches

In this section we discuss only schema-based elementary matchers. We address issues of their combination in the next section. Therefore, only schema/ontology information is considered, not instance data[1]. The exact/approximate opposition has not been used because each of the methods described below can be implemented as exact or approximate algorithm, depending on the goals of the matching system. To ground and ensure a comprehensive coverage for our classification we have analyzed state of the art approaches used for schema-based matching. The references section reports a partial list of works which have been scrutinized pointing to (some of) the most important contributions. We have used the following guidelines for building our classification:

Exhaustivity. The extension of categories dividing a particular category must cover its extension (i.e., their aggregation should give the complete extension of the category);

Disjointness. In order to have a proper tree, the categories dividing one category should be pairwise disjoint by construction;

Homogeneity. In addition, the criterion used for further dividing one category should be of the same nature (i.e., should come from the same dimension). This usually helps guaranteeing disjointness;

Saturation. Classes of concrete matching techniques should be as specific and discriminative as possible in order to provide a fine grained distinction between possible alternatives. These classes have been identified following a *saturation* principle: they have been added/modified till the saturation was reached, namely taking into account new techniques did not require introducing new classes or modifying them.

Notice that disjointness and exhaustivity of the categories ensures stability of the classification, namely new techniques will not occur in between two categories. Classes of matching techniques represent the state of the art. Obviously, with appearance of new techniques, they might be extended and further detailed.

As indicated in introduction, we build on the previous work of classifying automated schema matching approaches of [62]. The classification of [62] distinguishes between *elementary* (individual) matchers and *combinations* of matchers. Elementary matchers comprise *instance-based* and *schema-based*, *element-* and *structure-level*, *linguistic-* and *constrained-based* matching techniques. Also *cardinality* and *auxiliary information* (e.g., thesauri, global schemas) can be taken into account.

For classifying elementary schema-based matching techniques, we introduce two synthetic classifications (see Figure 3), based on what we have found the most salient properties of the matching dimensions. These two classifications are presented as two trees sharing their leaves. The leaves represent classes of elementary matching techniques and their concrete examples. Two synthetic classifications are:

[1] Prominent solutions of instance-based schema/ontology matching as well as possible extensions of the instance-based part of the classification of [62] can be found in [23] and [40] respectively.

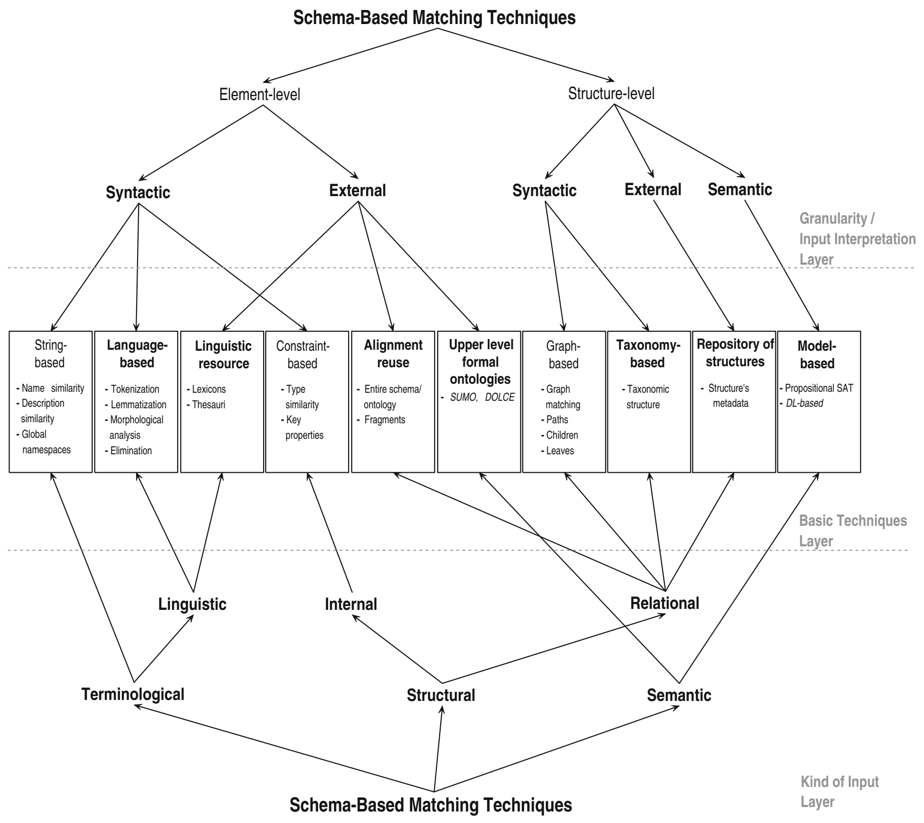


Fig. 3. A retained classification of elementary schema-based matching approaches

- *Granularity/Input Interpretation* classification is based on (i) granularity of match, i.e., element- or structure-level, and then (ii) on how the techniques generally interpret the input information;
- *Kind of Input* classification is based on the kind of input which is used by elementary matching techniques.

The overall classification of Figure 3 can be read both in descending (focusing on how the techniques interpret the input information) and ascending (focusing on the kind of manipulated objects) manner in order to reach the *Basic Techniques* layer. Let us discuss in turn *Granularity/Input Interpretation*, *Basic Techniques*, *Kind of Input* layers together with supporting arguments for the categories/classes introduced at each layer.

Elementary matchers are distinguished by the *Granularity/Input interpretation* layer according to the following classification criteria:

- *Element-level vs structure-level.* Element-level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with other entities. Structure-level techniques compute mapping elements by analyzing how entities appear together in a structure. This criterion is the same as first introduced in [62].

– *Syntactic vs external vs semantic.* The key characteristic of the syntactic techniques is that they interpret the input in function of its sole structure following some clearly stated algorithm. External are the techniques exploiting auxiliary (external) resources of a domain and common knowledge in order to interpret the input. These resources might be human input or some thesaurus expressing the relationships between terms. The key characteristic of the semantic techniques is that they use some formal semantics (e.g., model-theoretic semantics) to interpret the input and justify their results. In case of a semantic based matching system, exact algorithms are complete (i.e., they guarantee a discovery of all the possible mappings) while approximate algorithms tend to be incomplete.

To emphasize the differences with the initial classification of [62], the new categories/classes are marked in bold face. In particular, in the *Granularity/Input Interpretation* layer we detail further (with respect to [62]), the element- and structure-level of matching by introducing the *syntactic* vs *semantic* vs *external* criteria. The reasons of having these three categories are as follows. Our initial criterion was to distinguish between *internal* and *external* techniques. By *internal* we mean techniques exploiting information which comes only with the input schemas/ontologies. *External* techniques are as defined above. *Internal* techniques can be further detailed by distinguishing between *syntactic* and *semantic* interpretation of input, also as defined above. However, only limited, the same distinction can be introduced for the *external* techniques. In fact, we can qualify some oracles (e.g., WordNet [53], DOLCE [31]) as syntactic or semantic, but not a user's input. Thus, we do not detail *external* techniques any further and we omit in Figure 3 the theoretical category of *internal* techniques (as opposed to *external*). Notice, that we also omit in further discussions element-level semantic techniques, since semantics is usually given in a structure, and, hence, there are no element-level semantic techniques.

Distinctions between classes of elementary matching techniques in the *Basic Techniques* layer of our classification are motivated by the way a matching technique interprets the input information in each concrete case. In particular, a label can be interpreted as a string (a sequence of letters from an alphabet) or as a word or a phrase in some natural language, a hierarchy can be considered as a graph (a set of nodes related by edges) or a taxonomy (a set of concepts having a set-theoretic interpretation organized by a relation which preserves inclusion). Thus, we introduce the following classes of elementary schema/ontology matching techniques at the element-level: *string-based*, *language-based*, *based on linguistic resources*, *constraint-based*, *alignment reuse*, and *based on upper level ontologies*. At the structure-level we distinguish between: *graph-based*, *taxonomy-based*, *based on repositories of structures*, and *model-based techniques*.

The *Kind of Input* layer classification is concerned with the type of input considered by a particular technique:

– The first level is categorized depending on which kind of data the algorithms work on: strings (*terminological*), structure (*structural*) or models (*semantics*). The two first ones are found in the ontology descriptions, the last one requires some semantic interpretation of the ontology and usually uses some semantically compliant reasoner to deduce the correspondences.

– The second level of this classification decomposes further these categories if necessary: *terminological* methods can be *string-based* (considering the terms as sequences of characters) or based on the interpretation of these terms as linguistic objects (*linguistic*). The structural methods category is split into two types of methods: those which consider the *internal* structure of entities (e.g., attributes and their types) and those which consider the relation of entities with other entities (*relational*).

Notice that following the above mentioned guidelines for building a classification the *terminological* category should be divided into *linguistic* and *non-linguistic* techniques. However, since *non-linguistic* techniques are all *string-based*, this category has been discarded.

We discuss below the main classes of the *Basic Techniques* layer (also indicating in which matching systems they are exploited) according to the above classification in more detail. The order follows that of the *Granularity/Input Interpretation* classification and these techniques are divided in two sections concerning element-level techniques (§4.1) and structure-level techniques (§4.2). Finally, in Figure 3, techniques which are marked in italic (techniques based on upper level ontologies and DL-based techniques) have not been implemented in any matching system yet. However, we are arguing why their appearance seems reasonable in the near future.

4.1 Element-Level Techniques

String-based techniques are often used in order to match names and name descriptions of schema/ontology entities. These techniques consider strings as sequences of letters in an alphabet. They are typically based on the following intuition: the more similar the strings, the more likely they denote the same concepts. A comparison of different string matching techniques, from *distance* like functions to *token-based distance* functions can be found in [16]. Usually, distance functions map a pair of strings to a real number, where a smaller value of the real number indicates a greater similarity between the strings. Some examples of string-based techniques which are extensively used in matching systems are *prefix*, *suffix*, *edit distance*, and *n-gram*.

– *Prefix*. This test takes as input two strings and checks whether the first string starts with the second one. *Prefix* is efficient in matching cognate strings and similar acronyms (e.g., int and integer), see, for example [21, 34, 46, 50]. This test can be transformed in a smoother distance by measuring the relative size of the prefix and the ratio.
– *Suffix*. This test takes as input two strings and checks whether the first string ends with the second one (e.g., phone and telephone), see, for example [21, 34, 46, 50].
– *Edit distance*. This distance takes as input two strings and computes the edit distance between the strings. That is, the number of *insertions*, *deletions*, and *substitutions* of characters required to transform one string into another, normalized by the length of the longest string. For example, the edit distance between NKN and Nikon is 0.4. Some of matching systems exploiting the given technique are [21, 34, 57].
– *N-gram*. This test takes as input two strings and computes the number of common n-grams (i.e., sequences of n characters) between them. For example, $trigram(3)$

for the string nikon are nik, iko, kon. Thus, the distance between nkon and nikon would be $1/3$. Some of matching systems exploiting the given test are [21,34].

Language-based techniques consider names as words in some natural language (e.g., English). They are based on Natural Language Processing (NLP) techniques exploiting morphological properties of the input words.

– *Tokenization.* Names of entities are parsed into sequences of *tokens* by a tokenizer which recognizes punctuation, cases, blank characters, digits, etc. (e.g., Hands-Free_Kits $\rightarrow$ $\langle$hands, free, kits$\rangle$, see, for example [33]).
– *Lemmatization.* The strings underlying tokens are morphologically analyzed in order to find all their possible basic forms (e.g., Kits $\rightarrow$ Kit), see, for example [33].
– *Elimination.* The tokens that are articles, prepositions, conjunctions, and so on, are marked (by some matching algorithms, e.g., [46]) to be discarded.

Usually, the above mentioned techniques are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results. However, we consider these language-based techniques as a separate class of matching techniques, since they can be naturally extended, for example, in a distance computation (by comparing the resulting strings or sets of strings).

Constraint-based techniques are algorithms which deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys. We omit here a discussion of matching keys as these techniques appear in our classification without changes with respect to the original publication [62]. However, we provide a different perspective on matching datatypes and cardinalities.

– *Datatypes comparison* involves comparing the various attributes of a class with regard to the datatypes of their value. Contrary to objects that require interpretations, the datatypes can be considered objectively and it is possible to determine how a datatype is close to another (ideally this can be based on the interpretation of datatypes as sets of values and the set-theoretic comparison of these datatypes, see [71,72]). For instance, the datatype day can be considered closer to the datatype workingday than the datatype integer. This technique is used in [30].
– *Multiplicity comparison* attribute values can be collected by a particular construction (set, list, multiset) on which cardinality constraints are applied. Again, it is possible to compare the so constructed datatypes by comparing (i) the datatypes on which they are constructed and (ii) the cardinality that are applied to them. For instance, a set of between 2 and 3 children is closer to a set of 3 people than a set of 10-12 flowers (if children are people). This technique is used in [30].

Linguistic resources such as common knowledge or domain specific thesauri are used in order to match words (in this case names of schema/ontology entities are considered as words of a natural language) based on linguistic relations between them (e.g., synonyms, hyponyms).

- *Common knowledge thesauri.* The approach is to use common knowledge thesauri to obtain meaning of terms used in schemas/ontologies. For example, WordNet [53] is an electronic lexical database for English (and other languages), where various *senses* (possible meanings of a word or expression) of words are put together into sets of synonyms. Relations between schema/ontology entities can be computed in terms of bindings between WordNet senses, see, for instance [12, 33]. For example, in Figure 1, a sense-based matcher may learn from WordNet (with a prior morphological preprocessing of labels performed) that Camera in S is a hypernym for Digital Camera in S', and, therefore conclude that entity Digital_Cameras in S' should be subsumed by the entity Photo_and_Cameras in S. Another type of matchers exploiting thesauri is based on their structural properties, e.g., WordNet hierarchies. In particular, hierarchy-based matchers measure the distance, for example, by counting the number of arcs traversed, between two concepts in a given hierarchy, see [35]. Several other distance measures for thesauri have been proposed in the literature, e.g., [61, 64].
- *Domain specific thesauri.* These thesauri usually store some specific domain knowledge, which is not available in the common knowledge thesauri, (e.g., proper names) as entries with synonym, hypernym and other relations. For example, in Figure 1, entities NKN in S and Nikon in S' are treated by a matcher as synonyms from a domain thesaurus look up: syn key - ”NKN:Nikon = syn”, see, for instance [46].

Alignment reuse techniques represent an alternative way of exploiting external resources, which contain in this case alignments of previously matched schemas/ontologies. For instance, when we need to match schema/ontology o' and o'', given the alignments between o and o', and between o and o'' from the external resource, storing previous match operations results. The alignment reuse is motivated by the intuition that many schemas/ontologies to be matched are similar to already matched schemas/ontologies, especially if they are describing the same application domain. These techniques are particularly promising when dealing with large schemas/ontologies consisting of hundreds and thousands of entities. In these cases, first, large match problems are decomposed into smaller sub-problems, thus generating a set of schema/ontology fragments matching problems. Then, reusing previous match results can be more effectively applied at the level of schema/ontology fragments compared to entire schemas/ontologies. The approach was first introduced in [62], and later was implemented as two matchers, i.e., (i) reuse alignments of entire schemas/ontologies, or (ii) their fragments, see, for details [2, 21, 63].

Upper level formal ontologies can be also used as external sources of common knowledge. Examples are the Suggested Upper Merged Ontology (SUMO) [55] and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [31]. The key characteristic of these ontologies is that they are logic-based systems, and therefore, matching techniques exploiting them can be based on the analysis of interpretations. Thus, these are semantic techniques. For the moment, we are not aware of any matching systems which use these kind of techniques. However, it is quite reasonable to assume that this will happen in the near future. In fact, for example, the DOLCE ontology aims at providing a formal specification (axiomatic theory) for the top level part of WordNet. Therefore, systems exploiting WordNet now in their matching process might also consider using DOLCE as a potential extension.

4.2 Structure-Level Techniques

Graph-based techniques are graph algorithms which consider the input as labeled graphs. The applications (e.g., database schemas, taxonomies, or ontologies) are viewed as graph-like structures containing terms and their inter-relationships. Usually, the similarity comparison between a pair of nodes from the two schemas/ontologies is based on the analysis of their positions within the graphs. The intuition behind is that, if two nodes from two schemas/ontologies are similar, their neighbors might also be somehow similar. Below, we present some particular matchers representing this intuition.

– *Graph matching*. There have been done a lot of work on graph (tree) matching in graph theory and also with respect to schema/ontology matching applications, see, for example, [65, 77]. Matching graphs is a combinatorial problem that can be computationally expensive. It is usually solved by approximate methods. In schema/ontology matching, the problem is encoded as an optimization problem (finding the graph matching minimizing some distance like the dissimilarity between matched objects) which is further resolved with the help of a graph matching algorithm. This optimization problem is solved through a fix-point algorithm (improving gradually an approximate solution until no improvement is made). Examples of such algorithms are [50] and [30]. Some other (particular) matchers handling DAGs and trees are *children*, *leaves*, and *relations*.
– *Children*. The (structural) similarity between inner nodes of the graphs is computed based on similarity of their children nodes, that is, two non-leaf schema elements are structurally similar if their immediate children sets are highly similar. A more complex version of this matcher is implemented in [21].
– *Leaves*. The (structural) similarity between inner nodes of the graphs is computed based on similarity of leaf nodes, that is, two non-leaf schema elements are structurally similar if their leaf sets are highly similar, even if their immediate children are not, see, for example [21, 46].
– *Relations*. The similarity computation between nodes can also be based on their relations. For example, in one of the possible ontology representations of schemas of Figure 1, if class Photo_and_Cameras relates to class NKN by relation hasBrand in one ontology, and if class Digital_Cameras relates to class Nikon by relation hasMarque in the other ontology, then knowing that classes Photo_and_Cameras and Digital_Cameras are similar, and also relations hasBrand and hasMarque are similar, we can infer that NKN and Nikon may be similar too, see [48].

Taxonomy-based techniques are also graph algorithms which consider only the specialization relation. The intuition behind taxonomic techniques is that *is-a* links connect terms that are already similar (being a subset or superset of each other), therefore their neighbors may be also somehow similar. This intuition can be exploited in several different ways:

– *Bounded path matching*. Bounded path matchers take two paths with links between classes defined by the hierarchical relations, compare terms and their positions along these paths, and identify similar terms, see, for instance [57]. For example, in Figure 1, given that element Digital_Cameras in S' should be subsumed by the element Photo_and_Cameras in S, a matcher would suggest FJFLM in S and FujiFilm in S' as an appropriate match.
– *Super(sub)-concepts rules*. These matchers are based on rules capturing the above stated intuition. For example, if super-concepts are the same, the actual concepts are similar to each other. If sub-concepts are the same, the compared concepts are also similar, see, for example [19, 26].

Repository of structures stores schemas/ontologies and their fragments together with pairwise similarities (e.g., coefficients in the [0 1] range) between them. Notice, that unlike the alignment reuse, repository of structures stores only similarities between schemas/ontologies, not alignments. In the following, to simplify the presentation, we call schemas/ontologies or their fragments as structures. When new structures are to be matched, they are first checked for similarity to the structures which are already available in the repository. The goal is to identify structures which are sufficiently similar to be worth matching in more detail, or reusing already existing alignments. Thus, avoiding the match operation over the dissimilar structures. Obviously, the determination of similarity between structures should be computationally cheaper than matching them in full detail. The approach of [63], in order to match two structures, proposes to use some metadata describing these structures, such as structure name, root name, number of nodes, maximal path length, etc. Then, these indicators are analyzed and are aggregated into a single coefficient, which estimates similarity between them. For example, schema $S1$ might be found as an appropriate match to schema $S2$ since they both have the same number of nodes.

Model-based algorithms handle the input based on its semantic interpretation (e.g., model-theoretic semantics). Thus, they are well grounded deductive methods. Examples are propositional satisfiability (SAT) and description logics (DL) reasoning techniques.

– *Propositional satisfiability (SAT)*. As from [12, 32, 33], the approach is to decompose the graph (tree) matching problem into the set of node matching problems. Then, each node matching problem, namely pairs of nodes with possible relations between them is translated into a propositional formula of form: $Axioms \rightarrow rel(context_1, context_2)$, and checked for *validity*. $Axioms$ encodes background knowledge (e.g., Digital_Cameras $\rightarrow$ Cameras codifies the fact that Digital_Cameras is less general than Cameras), which is used as premises to reason about relations *rel* (e.g., $=$, $\sqsubseteq$, $\sqsupseteq$, $\perp$) holding between the nodes $context_1$ and $context_2$ (e.g., node 7 in S and node 12 in S'). A propositional formula is valid iff its negation is unsatisfiable. The unsatisfiability is checked by using state of the art SAT solvers. Notice that SAT deciders are correct and complete decision procedures for propositional satisfiability, and therefore, they can be used for an exhaustive check of all the possible mappings.

– *DL-based techniques*. The SAT-based approach computes the satisfiability of theory merging both schemas/ontologies along an alignment. Propositional language used for codifying matching problems into propositional unsatisfiability problems is limited in its expressivity, namely it allows for handling only unary predicates. Thus, it can not handle, for example, binary predicates, such as properties or roles. However, the same procedure can be carried within description logics (expressing properties). In description logics, the relations (e.g., $=$, $\sqsubseteq$, $\sqsupseteq$, $\perp$) can be expressed in function of subsumption. In fact, first merging two ontologies (after renaming) and then testing each pair of concepts and roles for subsumption is enough for aligning terms with the same interpretation (or with a subset of the interpretations of the others). For instance, suppose that we have one ontology introducing classes company, employee and micro-company as a company with at most 5 employees, and another ontology introducing classes firm, associate and SME as a firm with at most 10 associates. If we know that all associates are employees and we already have established that firm is equivalent to company, then we can deduce that a micro-company is a SME. However, we are not aware of existence of any schema/ontology matching systems supporting DL-based techniques for the moment.

There are examples in the literature of DL-based techniques used in relevant to schema/ontology matching applications. For example, in spatio-temporal database integration scenario, as first motivated in [60] and later developed in [68] the inter-schema mapping elements are initially proposed by the integrated schema designer and are encoded together with input schemas in $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ language. Then, DL reasoning services are used to check the satisfiability of the two source schemas and the set of inter-schema mapping elements. If some objects are found unsatisfied, then the inter-schema mapping elements should be reconsidered.

Another example, is when DL-based techniques are used in query processing scenario [52]. The approach assumes that mapping elements between pre-existing domain ontologies are already specified in a declarative manner (e.g., manually). User queries are rewritten in terms of pre-existing ontologies and are expressed in Classic [10], and further evaluated against real-world repositories, which are also subscribed to the pre-existing ontologies. An earlier approach for query answering by terminological reasoning is described in [4].

Finally, a very similar problem to schema/ontology matching is addressed within the system developed for matchmaking in electronic marketplaces [18]. Demand D and supply S requests are translated from natural language sentences into Classic [10]. The approach assumes the existence of a pre-defined domain ontology T, which is also encoded in Classic. Matchmaking between a supply S and a demand D is performed with respect to the pre-defined domain ontology T. Reasoning is performed with the help of the NeoClassic reasoner in order to determine the *exact match* ($T \models (D \sqsubseteq S)$) and ($T \models (S \sqsubseteq D)$), *potential match* (if $D \sqcap S$ is satisfiable in T), and *nearly miss* (if $D \sqcap S$ is unsatisfiable in T). The system also provides a logically based matching results rank operation.

5 On Classifying Matching Systems

As the previous section indicates, elementary matchers rely on a particular kind of input information, therefore they have different applicability and value with respect to different schema/ontology matching tasks. State of the art matching systems are not made of a single elementary matcher. They usually combine them: elementary matchers can be used in sequence (called *hybrid* matchers in [62]), examples are [5, 46], or in parallel (also called *composite* matchers [62]) combining the results (e.g., taking the average, maximum) of independently executed matchers, see, for instance [21, 23, 26]. Finding a better classification here is rather difficult.

The distinction between the sequential and parallel composition is useful from an architectural perspective. However, it does not show how the systems can be distinguished in the matter of considering the alignment and the matching task, thus representing an user-centric perspective. Below, we provide a vision of a classification of matching systems with respect to this point:

- *Alignments as solutions.* This category covers purely algorithmic techniques that consider that an alignment is a solution to the matching problem. It could be characterized as a (continuous or discrete) optimization problem, see, for example [30,50].
- *Alignments as theorems.* Systems of this category rely on semantics and require the alignment to satisfy it. This category, strictly speaking, is a sub-category of *alignments as solutions* (the problem is expressed in semantic terms). However, it is sufficiently autonomous for being singled out, see, for example [32, 33].
- *Alignments as likeness clues.* This category refers to the algorithms which aim at producing reasonable indications for a user to select the alignment, although using the same techniques (e.g., string-based) as systems from the *alignments as solutions* category, see, for example [21, 46].

6 Review of State of the Art Matching Systems

We now look at some recent schema-based state of the art matching systems in light of the classification presented in Figure 3 and criteria highlighted in Section 5.

Similarity Flooding. The Similarity Flooding (SF) [50] approach utilizes a hybrid matching algorithm based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs; grounding on the OIM specification [15] the algorithm manipulates them in an iterative fix-point computation to produce an alignment between the nodes of the input graphs. The technique starts from string-based comparison (common prefix, suffix tests) of the vertices labels to obtain an initial alignment which is refined within the fix-point computation. The basic concept behind the SF algorithm is the similarity spreading from similar nodes to the adjacent neighbors through propagation coefficients. From iteration to iteration the spreading depth and a similarity measure are increasing till the fix-point is reached. The result of this step is a refined alignment which is further filtered to finalize the matching process. SF considers the alignment as a solution to a clearly stated optimization problem.

Artemis. Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) [13] was designed as a module of the MOMIS mediator system [5] for creating global views. It performs affinity-based analysis and hierarchical clustering of source schema elements. Affinity-based analysis represents the matching step: in a hybrid manner it calculates the name, structural and global affinity coefficients exploiting a common thesaurus. The common thesaurus is built with the help of ODB-Tools, WordNet or manual input. It represents a set of intensional and extensional relationships which depict intra- and inter-schema knowledge about classes and attributes of the input schemas. Based on global affinity coefficients, a hierarchical clustering technique categorizes classes into groups at different levels of affinity. For each cluster it creates a set of global attributes - global class. Logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table. Artemis falls into the alignments as likeness clues category.

Cupid. Cupid [46] implements a hybrid matching algorithm comprising linguistic and structural schema matching techniques, and computes similarity coefficients with the assistance of a domain specific thesauri. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases and operates only with tree-structures to which non-tree cases are reduced. The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalization, categorization, string-based techniques (common prefix, suffix tests) and a thesauri look-up. The second phase (structural matching) computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur. The third phase (mapping elements generation) computes weighted similarity coefficients and generates final alignment by choosing pairs of schema elements with weighted similarity coefficients which are higher than a threshold. Referring to [46], Cupid performs somewhat better overall, than the other hybrid matchers: Dike [58] and Artemis [13]. Cupid falls into the alignments as likeness clues category.

COMA. COMA (COmbination of MAtching algorithms) [21] is a composite schema matching tool. It provides an extensible library of matching algorithms; a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. Matching library is extensible, and as from [21] it contains 6 elementary matchers, 5 hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques (affix, n-gram, edit distance, etc.) as a background idea; others share techniques with Cupid (thesauri look-up, etc.); and reuse-oriented is a completely novel matcher, which tries to reuse previously obtained results for entire new schemas or for its fragments. Schemas are internally encoded as DAGs, where elements are the paths. This aims at capturing contexts in which the elements occur. Distinct features of the COMA tool in respect to Cupid, are a more flexible architecture and a possibility of performing iterations in the matching process. Based on the comparative evaluations conducted in [20], COMA dominates Autoplex [6] and Automatch [7]; LSD [22] and GLUE [23]; SF [50], and SemInt [44] matching tools. COMA falls into the alignments as likeness clues category.

NOM. NOM (Naive Ontology Mapping) [26] adopts the idea of composite matching from COMA [21]. Some other innovations with respect to COMA, are in the set of elementary matchers based on rules, exploiting explicitly codified knowledge in ontologies, such as information about super- and sub-concepts, super- and sub-properties, etc. At present the system supports 17 rules. For example, rule (R5) states that if super-concepts are the same, the actual concepts are similar to each other. NOM also exploits a set of instance-based techniques, this topic is beyond scope of the paper. The system falls into the alignments as likeness clues category.

QOM. QOM (Quick Ontology Mapping) [25] is a successor of the NOM system [26]. The approach is based on the idea that the loss of quality in matching algorithms is marginal (to a standard baseline), however improvement in efficiency can be tremendous. This fact allows QOM to produce mapping elements fast, even for large-size ontologies. QOM is grounded on matching rules of NOM. However, for the purpose of efficiency the use of some rules have been restricted, e.g., R5. QOM avoids the complete pair-wise comparison of trees in favor of a (n incomplete) top-down strategy. Experimental study has shown that QOM is on a par with other state of the art algorithms concerning the quality of proposed alignment, while outperforming them with respect to efficiency. Also, QOM shows better quality results than approaches within the same complexity class. The system falls into the alignments as likeness clues category.

OLA. OLA (OWL Lite Aligner) [30] is designed with the idea of balancing the contribution of each component that compose an ontology (classes, properties, names, constraints, taxonomy, and even instances). As such it takes advantage of all the elementary matching techniques that have been considered in the previous sections, but the semantic ones. OLA is a family of distance based algorithms which converts definitions of distances based on all the input structures into a set of equations. These distances are almost linearly aggregated (they are linearly aggregated modulo local matches of entities). The algorithm then looks for the matching between the ontologies that minimizes the overall distance between them. For that purpose it starts with base distance measures computed from labels and concrete datatypes. Then, it iterates a fix-point algorithm until no improvement is produced. From that solution, an alignment is generated which satisfies some additional criterion (on the alignment obtained and the distance between aligned entities). As a system, OLA considers the alignment as a solution to a clearly stated optimization problem.

Anchor-PROMPT. Anchor-PROMPT [57] (an extension of PROMPT, also formerly known as SMART) is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms. The anchor-PROMPT is a hybrid alignment algorithm which takes as input two ontologies, (internally represented as graphs) and a set of anchors-pairs of related terms, which are identified with the help of string-based techniques (edit-distance test), or defined by a user, or another matcher computing linguistic similarity, for example [49]. Then the algorithm refines them by analyzing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar positions on similar paths. Finally, based on the frequencies and a user feedback, the algorithm determines matching candidates. Anchor-PROMPT falls into the alignments as solutions and alignments as likeness clues categories.

S-Match. S-Match [32–34] is a schema-based matching system. It takes two graph-like structures (e.g., XML schemas or ontologies) and returns semantic relations (e.g., equivalence, subsumption) between the nodes of the graphs that correspond semantically to each other. The relations are determined by analyzing the *meaning* (concepts, not labels) which is codified in the elements and the structures of schemas/ontologies. In particular, labels at nodes, written in natural language, are translated into propositional formulas which explicitly codify the label's intended meaning. This allows for a translation of the matching problem into a propositional unsatisfiability problem, which can then be efficiently resolved using (sound and complete) state of the art propositional satisfiability deciders. S-Match was designed and developed as a platform for semantic matching, namely a highly modular system with the core of computing semantic relations where single components can be plugged, unplugged or suitably customized. It is a hybrid system with a composition at the element level. At present, S-Match libraries contain 13 element-level matchers, see [35], and 3 structure-level matchers (e.g., SAT4J [9]). S-Match falls into the alignments as theorems category.

Table 1. Characteristics of state of the art matching approaches

	Element-level		**Structure-level**	
	Syntactic	**External**	**Syntactic**	**Semantic**
SF [50]	string-based (2); data types; key properties	-	iterative fix-point computation	-
Artemis [13]	domain compatibility; language-based (1)	common thesaurus (CT): synonyms, broader terms, related terms	matching of neighbors via CT	-
Cupid [46]	string-based (2); language-based (2); data types; key properties	auxiliary thesauri: synonyms, hypernyms, abbreviations	tree matching weighted by leaves	-
COMA [21]	string-based (4); language-based (1); data types	auxiliary thesauri: synonyms, hypernyms, abbreviations; alignment reuse (2)	DAG (tree) matching with a bias towards leaf or children structures (2); paths	-
NOM [26] **FOAM/QOM** [25]	string-based (1); domains and ranges	application-specific vocabulary	matching of neighbors (2); taxonomic structure (4)	-
Anchor-PROMPT [57]	string-based (1); domains and ranges	-	bounded paths matching (arbitrary links); bounded paths matching (processing *is-a* links separately)	-
OLA [30]	string-based (3); language-based (1); data types	WordNet(1)	iterative fix-point computation; matching of neighbors; taxonomic structure	-
S-Match [33, 34]	string-based (5); language-based (3);	WordNet: sense-based (2), gloss-based (6)	-	propositional SAT (2)

Table 1 summarizes how the matching systems cover the solution space in terms of the proposed classification. Numbers in brackets specify how many matchers of a particular type a system supports. For example, S-Match supports 5 string-based element-level syntactic matchers (prefix, suffix, edit distance, n-gram, and text corpus, see [34]), OLA has one element-level external matcher based on WordNet. Table 1 also testifies that schema/ontology matching research was mainly focused on syntactic and external techniques so far. Semantic techniques have been exploited only by S-Match [33].

Having considered some of the recent schema-based matching systems, it is important to notice that the matching operation typically constitutes only one of the steps towards the ultimate goal of, e.g., schema/ontology integration, web services integration or meta data management. To this end, we would like to mention some existing infrastructures, which use matching as one of its integral components. Some examples are Chimaera [49], OntoMerge [24], Rondo [51], MAFRA [47], Protoplasm [8]. The goal of such infrastructures is to enable a user with a possibility of performing such high-level tasks, e.g., given a product request expressed in terms of the catalog $C1$, return the products satisfying the request from the marketplaces $MP1$ and $MP2$. Moreover, use matching component $M5$, and translate instances by using component $T3$.

7 Conclusions

This paper presents a new classification of schema-based matching approaches, which improves the previous work on the topic. We have introduced new criteria which are based on (i) general properties of matching techniques, i.e., we distinguish between approximate and exact techniques; (ii) interpretation of input information, i.e., we distinguish between syntactic, external, and semantic techniques at element- and structure-level; and (iii) the kind of input information, i.e., we distinguish between terminological, structural, and semantic techniques. We have reviewed some of the recent schema/ontology matching systems in light of the classification proposed pointing which part of the solution space they cover. Analysis of state of the art systems discussed has shown, that most of them exploit only syntactic and external techniques, following the input interpretation classification; or terminological and structural techniques, following the kind of input classification; and only one uses semantic techniques, following both classifications. However, the category of semantic techniques was identified only recently as a part of the solution space; its methods provide sound and complete results, and, hence it represents a promising area for the future investigations.

The proposed classification provides a common conceptual basis, and, hence, can be used for comparing (analytically) different existing schema/ontology matching systems as well as for designing a new one, taking advantages of state of the art solutions. As the paper shows, the solution space is quite large and there exists a variety of matching techniques. In some cases it is difficult to draw conclusions from the classifications of systems. A complementary approach is to compare matching systems experimentally, with the help of benchmarks which measure the quality of the alignment (e.g., computing precision, recall, overall indicators) and the performance of systems (e.g., measuring execution time, main memory indicators). We have started working on such an approach and we have found useful our classifications for designing systematic benchmarks, e.g., by discarding features (one by one) from schemas/ontologies with respect to the classifications we had (namely, what class of basic techniques deals with what feature), see for preliminary results the I3CON initiative[2], Ontology Alignment Contest[3] [69], and Ontology Alignment Evaluation Initiative[4].

Future work proceeds in at least three directions. The first direction aims at taking into account some novel matching approaches which exploit schema-level information, e.g., [1, 14, 38, 45, 54]. As it has already been mentioned, in some applications (e.g., agents communication, web services integration) there are no instances given beforehand, and therefore, schema-based matching is an only solution for such cases. However, in the other applications (e.g., schema/ontology integration), instances are given beforehand, and therefore, instance-based approaches should be considered as a part of the solution space. Thus, the second direction of the future work aims at extending our classification by taking into account instance-based approaches, e.g., [17, 23, 40]. Finally, the third direction aims at conducting an extensive evaluation of matching systems by systematic benchmarks and by case studies on the industrial size problems.

Acknowledgements. This work has been partly supported by the Knowledge Web European network of excellence (IST-2004-507482).

References

1. Z. Aleksovski, W. ten Kate, and F. van Harmelen. Semantic coordination: a new approximation method and its application in the music domain. In *Proceedings of the Meaning Coordination and Negotiation workshop at the International Semantic Web Conference (ISWC)*, 2004.
2. D. Aumüller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *Proceedings of the International Conference on Management of Data (SIGMOD), Software Demonstration*, 2005.
3. C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
4. H. W. Beck, S. K. Gala, and S. B. Navathe. Classification as a query processing technique in the CANDIDE semantic data model. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 572–581, 1989.
5. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, (28(1)):54–59, 1999.
6. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 108–122, 2001.
7. J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 452–466, 2002.
8. P. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, (33(4)):38–43, 2004.
9. D. Le Berre. A satisfiability library for Java. http://www.sat4j.org/.
10. A. Borgida, R. Brachman, D. McGuinness, and L. Resnick. CLASSIC: A structural data model for objects. *SIGMOD Record*, 18(2):58–67, 1989.

[2] http://www.atl.external.lmco.com/projects/ontology/i3con.html
[3] http://oaei.inrialpes.fr/2004/Contest/
[4] http://oaei.inrialpes.fr/2005/

11. P. Bouquet, J. Euzenat, E. Franconi, L. Serafini, G. Stamou, and S. Tessaris. D2.2.1: Specification of a common framework for characterizing alignment. Technical report, NoE Knowledge Web project delivable, 2004. http://knowledgeweb.semanticweb.org/.
12. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 130–145, 2003.
13. S. Castano, V. De Antonellis, and S. De Capitani di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, (13(2)):277–297, 2001.
14. S. Castano, A. Ferrara, S. Montanelli, and G. Racca. Semantic information interoperability in open networked systems. In *Proceedings of the International Conference on Semantics of a Networked World (ICSNW), in cooperation with ACM SIGMOD*, pages 215–230, 2004.
15. Meta Data Coalition. Open information model, version 1.0. http://mdcinfo/oim/oim10.html, August 1999.
16. W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Proceedings of the workshop on Data Cleaning and Object Consolidation at the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.
17. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 383–394, 2004.
18. T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. A system for principled matchmaking in an electronic marketplace. In *Proceedings of the World Wide Web Conference (WWW)*, pages 321–330, 2003.
19. R. Dieng and S. Hug. Comparison of ”personal ontologies” represented through conceptual graphs. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 341–345, 1998.
20. H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the workshop on Web and Databases*, 2002.
21. H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the Very Large Data Bases Conference (VLDB)*, pages 610–621, 2001.
22. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 509–520, 2001.
23. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the semantic web. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 662–673, 2003.
24. D. Dou, D. McDermott, and P. Qi. Ontology translation on the Semantic Web. *Journal on Data Semantics (JoDS)*, II:35–57, 2005.
25. M. Ehrig and S. Staab. QOM: Quick ontology mapping. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 683–697, 2004.
26. M. Ehrig and Y. Sure. Ontology mapping - an integrated approach. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 76–91, 2004.
27. J. Euzenat. An API for ontology alignment. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 698–712, 2004.
28. J. Euzenat, J. Barrasa, P. Bouquet, R. Dieng, M. Ehrig, M. Hauswirth, M. Jarrar, R. Lara, D. Maynard, A. Napoli, G. Stamou, H. Stuckenschmidt, P. Shvaiko, S. Tessaris, S. van Acker, I. Zaihrayeu, and T. L. Bach. D2.2.3: State of the art on ontology alignment. Technical report, NoE Knowledge Web project delivable, 2004. http://knowledgeweb.semanticweb.org/.

29. J. Euzenat and P. Valtchev. An integrative proximity measure for ontology alignment. In *Proceedings of the Semantic Integration workshop at the International Semantic Web Conference (ISWC)*, 2003.
30. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 333–337, 2004.
31. A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI Magazine*, (24(3)):13–24, 2003.
32. F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review Journal (KER)*, (18(3)):265–280, 2003.
33. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75, 2004.
34. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. Technical Report DIT-05-014, University of Trento, 2005.
35. F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Proceedings of Meaning Coordination and Negotiation workshop at the International Semantic Web Conference (ISWC)*, 2004.
36. F. Giunchiglia and I. Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *Proceedings of the International workshop on Cooperative Information Agents (CIA)*, pages 18–35, 2002.
37. N. Guarino. The role of ontologies for the Semantic Web (and beyond). Technical report, Laboratory for Applied Ontology, Institute for Cognitive Sciences and Technology (ISTC-CNR), 2004.
38. B. He and K. C.-C. Chang. A holistic paradigm for large scale schema matching. *SIGMOD Record*, 33(4):20–25, 2004.
39. Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review Journal (KER)*, (18(1)):1–31, 2003.
40. J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 205–216, 2003.
41. V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *The International Journal on Very Large Data Bases (VLDB)*, 5(4):276–304, 1996.
42. J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attributed equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989.
43. M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.
44. W. S. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proceedings of the Very Large Data Bases Conference (VLDB)*, pages 1–12, 1994.
45. J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 57–68, 2005.
46. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the Very Large Data Bases Conference (VLDB)*, pages 49–58, 2001.
47. A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 235–250, 2002.
48. A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 251–263, 2002.

49. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 483–493, 2000.
50. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.
51. S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 193–204, 2003.
52. E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An approach for query processing in global information systems based on interoperability between pre-existing ontologies. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 14–25, 1996.
53. A. G. Miller. WordNet: A lexical database for English. *Communications of the ACM*, (38(11)):39–41, 1995.
54. P. Mitra, N. Noy, and A. Jaiswal. OMEN: A probabilistic ontology mapping tool. In *Proceedings of the Meaning Coordination and Negotiation workshop at the International Semantic Web Conference (ISWC)*, 2004.
55. I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*, pages 2–9, 2001.
56. N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 2002.
57. N. Noy and M. Musen. Anchor-PROMPT: using non-local context for semantic matching. In *Proceedings of the workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 63–70, 2001.
58. L. Palopoli, G. Terracina, and D. Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *ADBIS-DASFAA, Matfyzpress*, pages 108–117, 2000.
59. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 333–347, 2002.
60. C. Parent and S. Spaccapietra. Database integration: the key to data interoperability. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*. The MIT Press, 2000.
61. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, (19(1)):17–30, 1989.
62. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The International Journal on Very Large Data Bases (VLDB)*, (10(4)):334–350, 2001.
63. E. Rahm, H. H. Do, and S. Maßmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.
64. P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 448–453, 1995.
65. D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 39–52, 2002.
66. A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
67. P. Shvaiko. A classification of schema-based matching approaches. In *Proceedings of the Meaning Coordination and Negotiation workshop at the International Semantic Web Conference (ISWC)*, 2004.

68. A. Sotnykova, C. Vangenot, N. Cullot, N. Bennacer, and M.-A. Aufaure. Semantic mappings in description logics for spatio-temporal database schema integration. *Journal on Data Semantics (JoDS), Special Issue on Semantic-based Geographical Information Systems*, III, 2005.
69. Y. Sure, O. Corcho, J. Euzenat, and T. Hughes. *Evaluation of Ontology-based Tools*. Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON), 2004. http://CEUR-WS.org/Vol-128/.
70. M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Record*, 33(4):58–64, 2004.
71. P. Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.
72. P. Valtchev and J. Euzenat. Dissimilarity measure for collections of objects and values. *Lecture Notes in Computer Science*, 1280:259–272, 1997.
73. R. van Eijk, F. de Boer, W. van de Hoek, and J. J. Meyer. On dynamically generated ontology translators in agent communication. *International Journal of Intelligent System*, 16:587–607, 2001.
74. Y. Velegrakis, R. J. Miller, and J. Mylopoulos. Representing and querying data transformations. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 81–92, 2005.
75. H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huebner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 108–117, 2001.
76. L. Xu and D. W. Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In *Proceedings of the Semantic Integration workshop at the International Semantic Web Conference (ISWC)*, 2003.
77. K. Zhang and D. Shasha. Approximate tree pattern matching. In A. Apostolico and Z. Galil, editors, *Pattern matching in strings, trees, and arrays*, pages 341–371. Oxford University, 1997.

An Overview and Classification of Adaptive Approaches to Information Extraction

Christian Siefkes[1,2] and Peter Siniakov[1]

[1] Database and Information Systems Group, Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
siefkes@inf.fu-berlin.de, siniakov@inf.fu-berlin.de
[2] Berlin-Brandenburg Graduate School in Distributed Information Systems*

Abstract. Most of the information stored in digital form is hidden in natural language texts. Extracting and storing it in a formal representation (e.g. in form of relations in databases) allows efficient querying, easy administration and further automatic processing of the extracted data. The area of information extraction (IE) comprises techniques, algorithms and methods performing two important tasks: finding (identifying) the desired, relevant data and storing it in appropriate form for future use.

The rapidly increasing number and diversity of IE systems are the evidence of continuous activity and growing attention to this field. At the same time it is becoming more and more difficult to overview the scope of IE, to see advantages of certain approaches and differences to others. In this paper we identify and describe promising approaches to IE. Our focus is adaptive systems that can be customized for new domains through training or the use of external knowledge sources. Based on the observed origins and requirements of the examined IE techniques a classification of different types of adaptive IE systems is established.

1 Introduction

1.1 Information Extraction

There are things unique to humans that astonish and fascinate at the same time. One of them is the human language, admirable for its richness, complexity and ability to adapt to different cultural and social environments. But as valuable from cultural and aesthetic point of view human language is as challenging it is to grasp it scientifically, to formalize and make it manifest for computer processing. *Information extraction* (IE) builds the bridge between the evolutionary aspects of language development and the algorithmic approach to language. IE is one of the most promising efforts to exploit computational capabilities, accurateness and correctness of machines for accomplishing elaborate, often tedious task of searching for, analyzing and identifying desired information.

* The work of this author is supported by the German Research Society (DFG grant no. GRK 316).

Most of the information stored in digital form is hidden in natural language texts. Extracting and storing it in a formal representation (e.g. in form of relations in databases) allows efficient querying and easy administration of the extracted data. Moreover, information stored and queried in a canonical way can be processed and interpreted by computers without human interaction; it can serve for establishing ontologies, creation of knowledge bases and data analysis.

The area of IE comprises techniques, algorithms and methods performing two important tasks: finding (identifying) the desired, relevant data and storing it in appropriate form for future use. The notion of *fact extraction* is often used interchangeably with the notion of IE. The goals of fact extraction, however, are typically more specific and according to them fact extraction can be defined as the transformation of facts expressed in natural language to a given, formal, properly defined target structure. Fact extraction can therefore be regarded as a subset of IE extraction focusing on more rigidly structured representation forms.

1.2 Related Areas

A precursor of information extraction was the field of *text understanding* (or message understanding) which had the more ambitious aim of completely representing the contents of texts. To stimulate research in this area was the original goal of the *Message Understanding Conferences* (MUC) held from 1987 through 1998 under the auspices of the US government (ARPA/DARPA).

The term *text mining* (TM) is sometimes used almost synonymously to IE. It also denotes the application of data mining techniques to text with the goal of generating new knowledge by finding unknown patterns. TM in this second meaning aims farther than IE, which does not try to generate new knowledge, but only to represent facts explicitly expressed in a text in a more formal structure. But IE can be used as a first step in text mining, by extracting facts from the unstructured text to a database or other structured representation. In a second step, usual data mining techniques can be applied to the resulting database structure to discover interesting relationships in the data. This approach is utilized by [39].

IE and the better established field of *information retrieval* (IR) which locates relevant texts can be combined in various ways. IR can be used to select relevant documents for further analysis by IE. On the other hand, the structure filled by IE can also be utilized for more flexible IR (using a structured query language like SQL). Thus IE might be useful as a preparatory step for information retrieval as well as for postprocessing.

1.3 Goals and Evaluation Criteria

The history of IE as an independent field of research began in the 1980s as a number of academic and industrial research institutions were working on extracting information from naval messages in projects sponsored by the U.S. navy. After establishing of the Message Understanding Conference for comparing the performance of IE systems, information extraction experienced rapid growth extending its applications to new domains and employing diverse new techniques. Originally consisting of handcrafted rule-based systems, the spectrum of IE methods has been continuously enriched by systems applying statistical and learning methods. Now it ranges from classical pattern-oriented systems over numerous combinations of different AI and statistical methods to rather new approaches such as wrapper induction.

The rapidly increasing number and diversity of IE systems are the evidence of continuous activity and growing attention to this field. At the same time it is becoming more and more difficult to overview the scope of IE, to see advantages of certain approaches and differences to others. Furthermore it is hard to estimate what the development perspective of IE is, what ideas are promising and where the focus of IE will be in the future.

In this paper we identify and describe promising approaches to IE. Our focus is adaptive systems that can be customized for new domains by training or the use of external knowledge sources. Handcrafted systems that can only be adapted by elaborate rewriting are not considered. According to the observed origins and requirements of the examined IE techniques, a classification of different types of adaptive IE systems is established. The classification is significantly based on the essential methods and resources used for extraction such as learning techniques and models and central features. Therefore the approaches that belong to different classes are not necessarily completely orthogonal to each other since some techniques and features are not exclusive to an approach (e.g. rule-based approaches may use some statistical techniques for solving some subtasks in the extraction algorithm).

Since the number of existing systems is considerably large it will not be possible to provide a detailed description of each system. Instead we select systems that distinctly represent directions of research without focusing on details of the systems. However, features of systems are identified that are common for the approach they pursue. Table 1 lists the regarded systems and the approaches they represent.

We distinguish three main classes: rule learning, knowledge-based and statistical approaches. In Sec. 3–5 the approaches are presented according to the classification so as related subclasses are discussed in the common context. To make the analysis of different approaches more systematic and establish a common base for their comparison and correlation we consider several qualitative criteria.

Used methods and algorithms: We focus on how relevant content is identified in texts and what techniques are used to match it to the target structure. Learning capabilities, learning models, the amount and role of human interaction are analyzed to infer advantages and weaknesses of the approach. These aspects form the basis of our classification and are mainly discussed in Sec. 3–5 where the different types of approaches are presented.

Input and output features: Input characteristics involve the prerequisites that the processed texts should fulfill and requirements on used resources. These characteristics affect the domains where approaches can be employed

Table 1. Overview of the Selected Approaches and Systems

Approach	**System(s)**	**Section**
	Rule Learners	
Pattern & Template Creation	[40] [11] [44]	3.1
Covering Algorithms	Crystal [56, 52] Whisk [54] $(LP)^2$ [9, 10]	3.2
Relational	Rapier [4, 5] SRV [21]	3.3
Case-based	[6]	3.4
Wrapper Induction	Stalker [37, 38] BWI [22]	3.5
Hybrid (Decision Trees)	IE^2 [1]	3.6
	Knowledge-based Approaches	
Horn Clauses	TANKA/MaLTe [13]	4.1
Ontology-based	[15]	4.2
Thesaurus-based	TIMES [2, 7]	4.3
	Statistical Approaches	
Probabilistic Parsing	SIFT [35, 36]	5.1
Hidden Markov Models	Active HMMs [48, 49] Stoch. Optimization [23, 24] (C)HHMMs [51]	5.2
Conditional Markov Models & Random Fields	MEMMs [33] CRF [28, 34]	5.3
Token Classification	MaxEnt [8] MBL [59] TIE [50] ELIE [18, 19]	5.4
Fragment Classification & Bayesian Networks	SNoW-IE [46, 47] BIEN [41]	5.5

(application range) and how easily they can be adapted to new domains and resources (adaptability). It is examined how much preparatory work and linguistic preprocessing is necessary, whether morphological and syntactic analysis is presupposed etc. Another important factor is whether the approaches rely on external resources such as semantic resources (e.g. thesauri or ontologies).

Output features define the accomplished tasks—which IE tasks have been solved completely or partially. We consider whether single attributes of target structure can be identified in text (single slot extraction) or complex facts consisting of several attributes (template unification) can be found. A résumé over these characteristics is given in Sec. 6.

The description of a single approach features its analysis with respect to the proposed criteria, which is summarized at the end of the description. Sometimes criteria are omitted if they are not applicable. The universally applicable criteria concerning input requirements and considered features, learning characteristics and accomplished tasks are used for comparison of different approaches in Sec. 6 and allow conclusions about important differences between the identified classes of approaches in IE.

Considering the quantitative metrics *precision*, *recall* and *F-measure* isolated from the testing environment may be misleading since they depend a lot on the complexity of the target structure and training texts. Quantitative parameters are meaningful only if the systems are tested in comparable environments, with the same text corpus and target structure. Section 6.5 discusses quantitative comparisons and evaluation results on two standard corpora.

2 Architecture of a Typical IE System

A typical trainable IE system follows a pipeline architecture that comprises linguistic preprocessing, learning and application stage and, during the application phase, semantic postprocessing as the three main blocks (Fig. 1). Each of them handles a subset of steps that are particularly relevant for a pursued approach.

A text corpus including texts of the application domain and a target structure defining what the relevant information is constitute the minimum input for an IE system. Besides, it can be supported by additional semantic resources provided by a human.

Preprocessing of Input Texts: Text corpora often consist of unstructured, “raw” natural language texts. A big part of the relevant information can be distinguished by some regularity found in the linguistic properties of texts. Thus linguistic analysis can give helpful hints and determine important features for identifying relevant content. Following linguistic components proved to be useful for information extraction:

Tokenization: Starting with a sequence of characters the goal is to identify the elementary parts of natural language: words, punctuation marks and separators. The resulting sequence of meaningful tokens is a base for further linguistic and any text processing.

Sentence Splitting: Sentences are one of the most important elements of the natural language for structured representation of the written content. Binding interrelated information they are the smallest units for expression of completed thoughts or events. The correct recognition of the sentence borders is therefore crucial for many IE approaches. The task would be trivial if the punctuation marks were not ambiguously used. Correct representation of a text as a sequence of sentences is utilized for syntactic parsing.

Morphological Analysis: Certain facts are typically expressed by certain parts of speech (e.g. names). Determining parts of speech of tokens is known as POS tagging. Statistical systems can use POS tags as classification features, rule-based systems as elements of extraction rules.

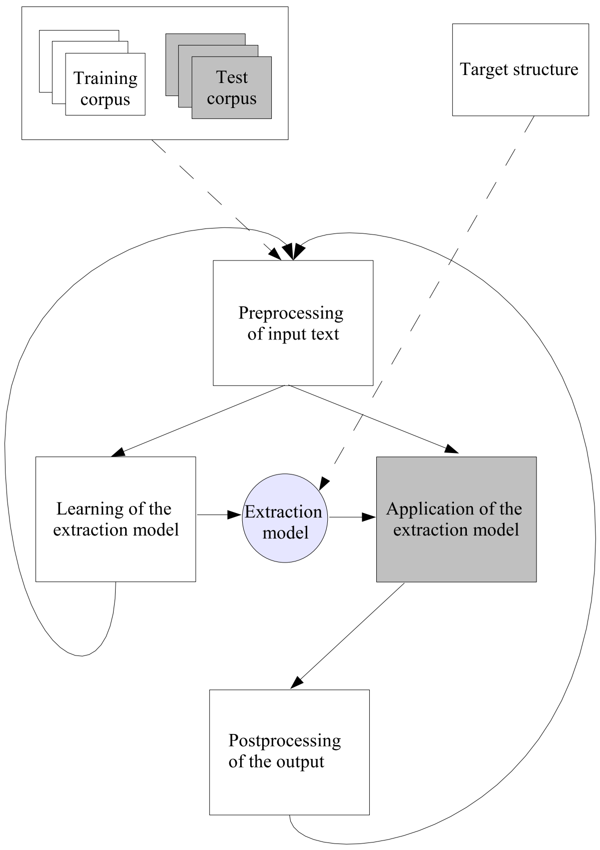

Training corpus
Test corpus
Target structure
Preprocessing of input text
Learning of the extraction model
Extraction model
Application of the extraction model
Postprocessing of the output

Fig. 1. Architecture of a Typical IE System

Segmentation of compounds, recognition of flection forms and consecutive normalization disclose further important morphological features.

(Chunk) Parsing: While full sentence parsing is preferred by knowledge-based systems, some statistical approaches rely on chunk parsing—shallow syntactic analysis of the sentence fragments performed on phrasal level. It is justified by the fact that the extracted information is often completely included in a noun, verb or prepositional phrase that build the most relevant context for its recognition.

Named Entity Recognition, Coreference Resolution: Named entities are one of the most often extracted types of tokens. Some approaches use a simple lookup in predefined lists (e.g. of geographic locations, company names), some utilize trainable Hidden Markov Models to identify named entities and their type. Coreference resolution finds multiple references to the same object in a text. This is especially important because relevant content may be expressed by pronouns and designators ("she held a seminar", "The company announced"). Both tasks require deeper semantic analysis and are not as reliable as other linguistic components.

While for knowledge-based and some rule-based systems linguistic preprocessing is an element of the core system, for statistical and other rule-based approaches it is optional but can have a serious impact on the quality of extraction.

Learning and Application of the Extraction Model: The application range of today's IE systems is intended to be as wide as possible. The features of a concrete domain cannot be hardwired in a system since the adaptation effort to other domains is too high. Modern systems use a learning component to reduce the dependence on specific domains and to decrease the amount of resources provided by human. An extraction model is defined according to the pursued approach and its parameters are "learned" (optimized) by a learning procedure. Statistical approaches learn, for example, relevant classification features, probabilities, state sequences, rule-based approaches learn a set of extraction rules and knowledge-based approaches acquire structures to augment and interpret their knowledge for extraction. The challenge is to find an extraction model that allows learning all relevant domain parameters using the same extraction framework for each application domain.

Considering the problems and complexity of IE, supervised learning appears to be the most appropriate and is the most widely used learning technique. The majority of approaches prefer annotated training corpora albeit some rely on human supervision during the learning stage. To assess the quality of an approach the training text corpus is created by annotating text fragments that contain relevant content and divided into two parts. One part, the training set, is used for training (learning the parameters of the extraction model) and another, the test set, is used to test the ability of the model to correctly extract new information it was not trained on. The test results can also be used to improve the extraction model to perform better on new domain texts when applied to real domain texts.

Some approaches allow further refinement of an extraction model based on the human feedback about extractions during the application. The new evaluated extractions can be incorporated as new training instances and the model can be retrained.

The learning component is crucial for an IE system, because it comprises the algorithms for identification of relevant text parts and transferring them according to the target structure.

Postprocessing of Output: The main motivation for IE is the structured representation of information that enables formal queries and automatic processing. One of the possibilities to structure the extracted data is to model the target structure as a database relation. After the relevant information has been found by application of the extraction model the identified text fragments are assigned to the corresponding attributes of the target structure. They can normalized according to the expected format (e.g. representation of dates and numbers). Some identified facts may appear in text more than once or already exist in the database. In this case, different instances could be merged (instance unification). Finally, the identified, normalized and unified information is stored at the appropriate relation in the database. Most current trainable systems do not yet perform much preprocessing, leaving such tasks as future work.

3 Rule-Learning Approaches

3.1 Automatic Pattern and Template Creation

To overcome some serious limitations of classical rule-based approaches, alternative techniques have been developed that reduce the manual effort and the amount of human knowledge used for the creation of extraction rules. In the optimal case the rules are determined automatically after the information about data to be extracted has been provided. Automatic acquisition of linguistic patterns and templates partially performs this task constructing the left-hand side of the rule and the target structure respectively. It is noteworthy that this approach does not presuppose a fix given target structure, in fact, the target structure is determined dynamically using provided semantic information. Therefore methods described below do not belong to the scope of fact extraction, but certainly comprise one substantial direction of IE. As representatives for this approach Nobata and Sekine's system for pattern acquisition [40], a method for template creation proposed by Collier [11] and a successor of *AutoSlog-TS* that accomplishes both tasks [44] are considered.

To compensate the lack of human interaction, syntactic and lexical resources should be provided that sufficiently cover word semantics and disclose necessary domain information. Therefore preclassified text corpora (Riloff) or reliable IR technology (Collier), a thesaurus or keyword list, part-of-speech (POS) tagger (Nobata) and named entity recognizer are required. Moreover shallow parsing is needed for syntactic analysis. Riloff's system is supported by a list of categories with five seed words in each for the creation of semantic lexicon and by a set of template slots (roles) mapped to corresponding categories. In all systems human inspection of intermediary or final results serves for quality assurance and learning purposes.

Pattern acquisition: Methods for automatic pattern acquisition have in common that the acquired patterns have a simple syntactic structure and the final set is selected from a large amount of initial candidate patterns iteratively using heuristics and statistical methods. The accent of Nobata's system lies on finding patterns describing certain events—an overview of the process is given in Fig 2. Actual information extraction is a direct mapping of ordered lexical items matching acquired pattern to fix template slots. Patterns are acquired by consecutive selection of text fragments and final merging of ordered lexical items in similar sentences. Articles are retrieved by scenario relevant keywords from a large untagged corpus, which are filtered by subject line. Selected articles are POS-tagged and named entities (NE) are recognized. Every sentence in the

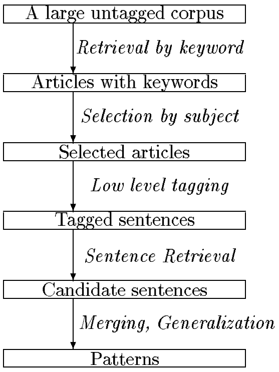


Fig. 2. Overview of Nobata's algorithm (from [40])

selected articles is regarded as initial pattern. Final patterns are acquired by merging lexical items and named entities of two similar sentences. Sentences are considered similar if they have the smallest amount of different items. Merging is facilitated by either ignoring extra items of one of the patterns or creating clusters of different items of both patterns. The merging process is repeated iteratively until the biggest possible generalization is achieved. In a later work [57] a tree representation for patterns is proposed to better account for dependency structures of syntactic patterns and to cope with the problem of free word order.

Riloff's system uses *AutoSlog-TS* for the generation of patterns [43]. It is guided by an assumption that the items to be extracted are comprised by noun phrases, therefore it uses heuristics to create linguistic patterns that represent relevant context for extracting of a given noun phrase (NP). These should be general enough to extract other relevant NPs as well. Typical patterns would be *<subject> exploded; exploded in <noun-phrase>*. They are activated by a keyword and information to be extracted is contained in a syntactic constituent of the pattern clause. In the first stage patterns are generated that collectively extract every noun phrase from the training text. In the second stage their relevance in the examined domain is estimated and a ranking of patterns is produced. The pattern score depends on the number of extracted NPs also found in semantic lexicon for examined domain and on their percentage among all extracted NPs. After human review the best patterns are selected to extract relevant information.

Template generation: The idea of generating target structure automatically may appear somewhat surprising since humans are primarily interested in and determine what should be extracted. However, it is motivated by the fact that the number of templates can be considerably large and therefore difficult to manage. Generation of templates is guided by semantic constraints provided to the system in form of categorization of domain entities in a semantic lexicon or thesaurus.

Riloff's system [44] constructs the semantic lexicon taking only a list of categories with five seed entries as input. Context of the seed words is regarded for each category and the words in context are scored. The score is basically the conditional probability that the word appears in category context. The top five are added to the category list and the process is continued iteratively. After several iterations a user corrects the list and approves the lexicon. After acquisition of patterns described above and lexicon creation semantic profiles of patterns are established. The correlation between a pattern and a domain category (semantic preference) is expressed based on extracted items. Therefore patterns are applied to relevant domain texts and assigned to categories depending on extracted NPs. These assignments serve later as constraints for the assignment of extracted items to template roles.

Another important source of semantic information provided by the user is the functional dependency between semantic categories and domain roles (template slots). It is based on a reasonable assumption that members of a category can fulfill only one role but one role can be fulfilled by entities of several categories. During extraction the role is assigned according to the semantic preferences (prevalent semantic categories of extracted items) determined earlier. A pattern can extract words of different categories and thus different roles. The presented patterns have a serious limitation extracting only one syntactic constituent and filling therefore only one template slot. To consolidate scattered information patterns that share the same trigger word and compatible syntactic constraints are merged into single pattern. Such a generalized pattern is able to extract items of several roles and creates a multi-slot template as the result of extraction. Hence templates are not fix but can comprise any subset of the set of roles. Giving the system the set of roles only sketches the scope of possible target structure while its actual creation is based on described algorithms.

Collier's approach proposes template creation on a pretty general level using syntactic information and statistical techniques. It identifies three types of information contained in texts, which are relevant for template creation: objects, their interactions and features. Named entities are considered as important objects and the assumption is made that fundamental objects can be found in every relevant domain text. Their identification can be facilitated by existing NE recognizers. Coreferences can be another source for object recognition. Interactions are expressed by verb/subject/object relationships on sentence level. Considering the categories of verbs (obtained from a thesaurus) classes of relationships can be established. Such a class would correspond to a template while features would be mapped to template slots. Relevant features can be found by looking on entities occurring not in every document, consulting thesauri, analyzing n-grams and collocations and using other statistical methods.

The main advantage of automatic creation of patterns and templates is obvious just by looking at the name of this approach. Provided with initial domain and semantic information, the described algorithms solve the problem of IE by generating patterns and templates and partially also extraction mechanisms. The algorithms include no or occasional human interaction and rather small human support involving basically review of obtained results. There is not much pre-processing and additional resources required for this approach, no deep syntactic analysis is necessary. Many subtasks (e.g. creating patterns and semantic lexicon) are solved using robust statistical methods. Generally, presented approach achieves successful results using quite simple comprehensive techniques.

Due to automatic processing researchers restricted the syntactic structure of patterns to be very simple, which is a serious limitation since many relevant facts are expressed in a complex linguistic context with complicated syntactic structure. Facts expressed over multiple sentences remain uncovered. Therefore such patterns cannot be applied to every domain. Since human influence during the runtime is very restricted, the quality of final results depends very much on the quality of semantic information provided at the beginning. Especially the demands on domain specification by categories and roles have to be very detailed and precise. Because of functional dependency between a semantic category and a role wrong slots may be generated, which leads to an adulterated target structure. Another big problem is unknown words (not occurring in the training texts) since heuristics are necessary to decide which role to assign. Additionally, Nobata's system suffers from the choice of sentences based on keywords that may fail because of noise or polysemous keywords. Besides, generalization of patterns is quite limited since, while matching a pattern, clusters of items have to be searched for a matching element, which corresponds to matching against many patterns. Collier's assumption that relevant objects occur in all documents is also very arguable.

Generally, this approach can be applied to domains and languages where desired information is expressed by facts with simple syntactic structure. Although the described techniques are primarily designed for information extraction other areas involved in creation of lexical resources or translation may use them to solve related tasks. This approach is promising because of its main advantage, many weaknesses are not inherent and can be overcome in the future.

3.2 Covering Algorithms

A number of IE systems are based on *covering (separate-and-conquer) algorithms* [25], a special type of *inductive learning*. These systems require a predefined target structure, they do not create it. Except for *Whisk*, which employs active learning, they also require a set of fully tagged training texts where all text fragments that fill a certain slot in the target structure are marked. Based on this input, the systems learn rules that extract the tagged slot fillers. After learning rules that cover a part of the training instances, they remove (separate) these instances from the training set and continue to learn rules that cover (conquer) some of the remaining instances, looping until all or most of the training instances are covered. What is regarded as an instance and which features are considered depends on the system.

Crystal: Crystal [56, 52] builds on a chunk parser that identifies syntactic constituents (subject, verb phrase, direct and indirect object, prepositional phrases) and a domain-specific dictionary that specifies semantic classes for all words. Crystal looks for constituents that fit predefined conceptual types (e.g. *diagno-*

sis, symptom) and subtypes (a diagnosis is either *confirmed, ruled-out, suspected, pre-existing,* or *past*).

The definitions learned to extract subtypes identify a constituent to extract if certain constraints are fulfilled by the surrounding constituents. Constraints may test for word sequences contained in a phrase or for semantic classes of the head noun or a modifier of a phrase. For example, an "absent symptom" is extracted from the direct object if the head of the direct object is of the class *[Sign or Symptom]*, the verb is "denies" in the active voice, and the subject includes the word "patient" and has the head class *[Patient or Disabled Group]*. Negative constraints are not supported.

In a later work [55], the problem of negation is solved by learning different kinds of semantic relations (classes) in a predefined order. Rules for "absent ... " are learned (and applied) first. The examples covered by these rules are removed prior to learning rules for "present ... ". Thus specialized rules like *verb group includes "not observed"* can be learned for the absent case, and general rules like *verb group includes "observed"* for the present case (without a predefined order this is not possible, because the general rule covers both cases).

Crystal learns suitable definitions by generalization, i.e. bottom-up: each training instance is used as a highly constrained initial definition. Crystal tries to unify "similar" definitions by relaxing constraints. The similarity metric is based on the number of constraint changes necessary to unify definitions.

Two definitions are unified by finding the most restrictive constraints that cover both. For semantic class constraints the most specific common ancestor in the semantic hierarchy is used; for word constraints the subset of words contained in both constraints is kept. If there is no common ancestor or the subset is empty, the constraint is dropped. The new definition replaces the original ones if the number of false positives it extracts is below a defined error threshold—increasing this threshold results in higher recall at the cost of precision (and vice versa).

For multi-slot extraction Crystal treats each subset of slot combinations as a concept to be learned—this can result in data sparseness. Crystal does not extract exact phrases, it only identifies a constituent to extract from. These are the major limitations of the system.

Whisk: The *Whisk* system developed later by the same author [54] is aimed at handling a larger range of texts, from free texts as found in newspapers and books to semi-structured texts (often ungrammatical or in "telegram style") that are common on the World Wide Web or in advertisements.

Whisk is targeted at multi-slot extraction at the sentence level. The learned rules are a kind of regular expressions. Expression pattern can contain verbatim text, character classes (e.g. digit, number), and wildcards like "*" which lazily skips any characters until the next part of the pattern can match. In addition to the hard-wired character classes, semantic classes of equivalent terms can be defined by the user, e.g. a class *Bdrm* that contains different forms and abbreviations of the term "bedroom." Parentheses indicate a phrase to be extracted. The *Output* part of a rule specifies where to store the extracted phrases. "Pattern:: *

(*Digits*) *Bdrm* * '$' (*Number*). Output:: Rental {Bedrooms $1} {Price $2}." is a rule to extract the number of bedrooms and the price from a rental ad.

When processing free (grammatical) text, each sentence is split into the fields returned by the chunk parser (subject, verb etc.); these fields can be specified in the regular expressions to constrain matching (but still the whole expression must match left to right, so the ordering of fields matters). Additional semantic classes are defined that match the output of a named entity recognizer (*person*, *company* etc.).

Rules are derived top-down (starting with the most general rule) by a covering algorithm. For judging the quality of rules, Whisk uses the Laplacian expected error: $Laplacian = \frac{e+1}{n+2}$, where n is the number of extractions made and e is the number of errors among these. In case of a tie, the more general rule is used. The found rules might not be optimal due to the limitations of hill climbing—each specialization is evaluated in isolation, so if two specializations (adding two terms) must be applied together to yield a better rule (according to the Laplacian), they will not be found.

Whisk incorporates *active learning*, so only a small part of the training corpus needs to be tagged in advance. The system proceeds by selecting three kinds of untagged instances for hand-tagging by the user: instances covered by a rule (which will either increase the support of the rule or force further refinement), "near misses" (to check and adapt the boundaries of rules), and a random sample of instances not covered by any rule (to check whether there are still rules to discover).

A disadvantage is that semantic classes must be predefined by the user, they are not learned by the system. Another drawback is the strict ordering constraints of each rule—different rules must be learned for each possible arrangement of slots.

(LP)^2: (LP)^2 [9] learns rules to add SGML/XML tags to a text. (LP)^2 is based on *tagging rules* that insert a single (starting or ending) SGML tag into the text. This means that the task of each rule is to recognize the start or the end of a supposed slot filler in the text, not to extract/tag a whole slot filler (or several slot fillers) at once, as in most other systems.

The tagging rules are learned from the hand-tagged training corpus. Rules are learned bottom-up, taking an instance as an initial rule whose constraints are subsequently relaxed (e.g. requiring only a lexical class instead of a specific word) or completely dropped. The k best generalizations of each initial rule found by a beam search are stored in a "best rules pool." As (LP)^2 is a covering algorithm, the training instances covered by a rule in this pool are removed from the training set.

(LP)^2 proceeds in four steps:

1. The *tagging rules* from the "best rules pool" are applied.
2. *Contextual rules* are applied to resulting text. These are tagging rules whose overall reliability was not high enough for the best rules pool but that perform better when restrained to the vicinity of tags inserted in the first step

(for example, a rule that inserts an end tag is applied provided that a corresponding start tag occurred some words before).
3. *Correction rules* do not add or delete tags, they only change the position of a tag, moving it some words forward or backward.
4. Finally, invalid markup (unclosed tags etc.) is deleted in a *validation* step.

In the *Amilcare* system, $(\mathrm{LP})^2$ is employed in a "LazyNLP" setting where the amount of utilized linguistic information can be dynamically adjusted [10]. The learner initially induces rules without any linguistic knowledge; then it iterates adding linguistic information (provided by third-party components), stopping when the effectiveness of the generated rules no longer increases. The adequate amount of linguistic input is learned for each type of slot separately (e.g. recognizing a person name might require more NLP input than recognizing a date or time).

$(\mathrm{LP})^2$ is targeted at slot filling and does not perform any template unification. In Amilcare a shallow discourse representation module is added for this purpose [26, Sec. 5]. Slot fillers are unified in templates or subtemplates with the nearest preceding slot fillers of a suitable type. E.g. when describing hotels, *address* data and *room types* (single room, double room) will be attached to the last mentioned *hotel*; *price* information might in turn be attached to the last mentioned *room type*.

3.3 Relational Rule Learners

The basic approach of the systems presented in this section is similar to those of the previous section—indeed, they are based on covering algorithms too. The main difference is that the systems presented here explicitly take relations—especially positional relations—between a (potentially unlimited) number of features into account, while those in the previous section are limited to predefined (finite) combinations of features.

Rapier: The *Rapier* [4, 5] system uses syntactic (POS tags) and semantic (WordNet classes) information to induce rules for slot fillers. Each rule consists in three parts, a pre-filler pattern, a pattern for the actual slot filler and a post-filler pattern. Each pattern contains an ordered list (whose length might be zero for pre/post-fillers) of constraints that restrict the POS tag, the semantic class, and/or the word itself (disjunctions are allowed). Instances are most specific rules with all their constraints set. The pre- and post-filler patterns of instances contain every word from the start resp. to the end of the document, there is no "context window" of limited length.

New rules are created by randomly selecting two rules and creating the least general generalization for the filler pattern. Actually, there are several reasonable generalizations (different values of a constraint can be disjuncted or the constraint can be simply dropped), so each of these generalizations is re-specialized by adding generalized pieces of the pre- and post-filler patterns of the original rules. A list of n best candidates is kept until the best generalization is found, based on the evaluation metric

$$ruleVal = -\log_2\left(\frac{p+1}{p+n+2}\right) + \frac{ruleSize}{p},$$

where p is the number of correct extractions and n the number of erroneous extractions; *ruleSize* is calculated depending on the number of pattern items, lists, and disjuncts in a rule.

Semantic classes are generalized by finding the nearest common ancestor in the WordNet hypernym hierarchy (dropping the constraint if no common ancestor exists). Instances covered by the found best generalization are subsequently ignored and further rules are learned based on the other instances.

Rapier has been extended to use *active learning* [58]. The system is initially trained from a small number of annotated examples. Then it tries to annotate a large number of untagged examples, selecting those examples whose annotation is least certain (certainty-based selective sampling). After the user has annotated the selected examples, the system is incrementally retrained and the process continued. Rapier does not provide probabilities, so the certainty of a rule is estimated based on its coverage: $pos - 5 \times neg$, where *pos* is the number of correct extractions on the training data and *neg* the number of incorrect ones. The active-learning version requires approximately half the examples to reach the performance level of Rapier without active learning.

SRV: SRV [21] considers any combination of simple features (mapping a token to a value, e.g. *word length: 5*, *character type: alpha*, *orthography: capitalized*, *POS tag: noun*, *semantic class: geographical-place*) and relational features (mapping a token to another token, e.g. *next-token*, *subject-verb*). Feature values can be sets, e.g. all synonyms and hypernyms (superordinate concepts) listed by WordNet are combined in a set for each token. SRV performs only a two-class classification, i.e. different rule sets are learned for classifying each text fragment as an instance or non-instance of a single slot filler—there is no component for template unification or other postprocessing.

The learning algorithm is similar to the relational rule learner FOIL [42]. SRV learns top-down, greedily adding predicates of some predefined types: the number of tokens in the fragment (*length*), whether a condition is matched by one or several (*some*) or by all (*every*) tokens in the fragment; *position* specifies the position of a token in a *some* predicate, *relpos* constrains the ordering and distance between two tokens. The *some* predicate can be constrained by relational features, for example, *some(?A [prev_tok prev_tok] numeric true)* means: there is some token in the fragment preceded by a numeric token two tokens back.

Rules are validated and their accuracy estimated by three-fold cross validation. The three resulting rule sets are then merged. The accuracy estimations are available for each prediction.

An advantage of relational learners is their being able to acquire powerful relational rules that cover a larger and more flexible context than most other rule-learning and statistical approaches. The downside is that the large space of possible rules can lead to high training times and there is no guarantee of finding optimal rules (local maxima problem).

3.4 Case-Based Approaches to IE and Knowledge Acquisition

To identify semantic and syntactic word features, knowledge-based approaches rely on manually prepared world knowledge while statistical approaches are only guided by regularities and patterns found in the training corpus. Case-based approaches combine the advantages of both by accumulating knowledge only from the training corpus and using previous experience to handle new words. The case-based method suggested by Cardie [6] needs therefore no explicit disambiguation heuristics, but domain knowledge provided at the initial stage. It includes knowledge about plausible parts of speech, word senses and contexts in a given domain. The training phase is supervised by a user. The system solves three independent tasks: POS assignment, word sense disambiguation and determination of the word concept (category of the word). Processing of training texts results in a case base of cases comprising these three types of information for each non-functional word in the training corpus. The case base is used to perform the tasks described above in new domain texts with unknown words.

Case base is constructed by acquiring a case for each occurrence of any non-functional word in different context while parsing sentences of the training set. A case summarizes the actual word features and features of its context. Word features describe part of speech, general and specific word senses and concept, context features include word features of two preceding and following words and the parser state before processing of the current word. To define features of the current word the human user is consulted. For the specification of the context features of two preceding words case base is queried, the features of two following words are added after the parser reaches them.

When a new text is processed, sentences are parsed and the context features are identified in the same way. To identify the current word, the features of the most similar cases are retrieved from the case base. Feature values that occur in retrieved cases most frequently are selected to be the features of the new word. Similarity of cases is measured by k-nearest-neighbors metric. Subsets of case features that are relevant for determination of the value of each word feature are identified using decision trees. For each word feature the relevant subset of the context features of the currently processed word is compared with the same subset of context features of cases in the case base. The more equal feature values are found the more similar are the cases. Since similarity is influenced only by context features, the assumption is proposed that the context is the only factor that defines all three types of knowledge for a word.

The approach benefits from the fact that there are no fix word-concept pairings and a much more realistic dependency of concept of the word on context is assumed. Syntactic and semantic knowledge is learned simultaneously and stored in one composite structure—the case. Source of domain information can be human or semantic lexical resources, which makes the training of the system easier.

From the point of view of IE, category of a word and its sense are a very sparse "target structure". Semantics are revealed not on a sentence, but on a word level. Besides, it is arguable whether the context can be captured adequately by regarding a static context window of 4 words. The assumption of exclusive role of context in determination of word features is simplistic.

An interesting perspective is the extension of this approach to work on the sentence level with cases summarizing sentence features, which would make solving more complex IE tasks such as fact extraction possible. Generally, similar problems in IE have similar solutions, therefore case-based reasoning methods are likely to be very promising for the future of IE.

3.5 Wrapper Induction

Stalker: The approach of *Wrapper Induction* (WI) is mainly targeted at structured and semi-structured documents that were generated automatically, e.g. Web pages offering products of listing information. The *Stalker* algorithm [37] covers documents that can be described in the *embedded catalog (EC)* formalism. This formalism represents a document as a tree whose leaves contain the relevant data (items of interest for the user). The inner nodes contain lists of k-tuples (e.g. of restaurant descriptions). Each item in a tuple is either a leaf or another (embedded) list.

Extraction is based of the EC description of a document and an extraction rule that extracts the contents of each node or tuple from the contents of its parent. List nodes require an additional *list iteration rule* that splits the list into tuples.

Extraction rules are based on groups of successive tokens called *landmarks. Start rules* locate the start of an item by find the first matching landmark from the begin of the parent; *end rules* locate the end of the item by finding the last matching landmark before the end of the parent. The text matched by a landmark itself can either be included (SkipUntil condition) or excluded (SkipTo condition) from the item text. Rules can combine several conditions, e.g. SkipTo(Name) SkipTo(<b>)' means that the item starts immediately after the first HTML tag <*b*> that follows the word *Name*. They can refer to specific tokens or to wildcards like Number, Punctuation, or HtmlTag.

Disjunctions (either ... or) are allowed to handle formatting variations. Disjunctions are ordered so the first successful match is used. Each node is extracted independently of the other nodes within its parent, so no fixed ordering is required.

Rules are learned by a covering algorithm that tries to generate rules until all instances of an item are covered (without false extractions, if possible) and returns a disjunction of the found rules. Rules with fewer false extractions (or more correction extractions, in case of a tie) are preferred when ordering the disjunction.

To support *active learning*, Stalker has been embedded in a *Co-Testing* approach [38]. Co-Testing combines a number of *views* that independently learn to recognize slot fillers. In the *Aggressive Co-Testing* approach, views can either be *strong* (they can learn how to reliably recognize slot fillers) or *weak* (they might learn either more general or more specific concepts, i.e. might either miss some instances or extract spurious instances).

Stalker is used as a *strong view.* Stalker's learning how to recognize the *begin* and *end* of a slot filler is complemented by a weak view that learns patterns to recognize the *content* of a slot filler (length range, contained token types etc.). The content recognizer is a *weak view* because it learns concepts that might be more general than the target slot filler, e.g. it cannot discern a phone number from a fax number. A second strong view is provided by running Stalker backwards, starting from the end of the document (BackTo instead of SkipTo). Predictions are combined by *majority voting:* When both strong views agree, the weak view is ignored; otherwise the prediction of the strong view that violates fewer constraints of the weak view is chosen.

Boosted Wrapper Induction: Typical WI algorithms such as Stalker are only suited for documents whose structure and layout are regular and consistent. They are inadequate for free text, where information is mainly expressed in natural language. The *BWI (Boosted Wrapper Induction)* system [22] aims at closing this gap and making WI techniques suitable for free text.

The rules learned by BWI are simple contextual patterns for finding the start and end of the field to extract. A pattern has two parts: a token sequence that immediately precede/follow the field to extract (outside) and a token sequence starting/ending it (inside). E.g. to identify the speaker's name in a seminar announcement, the pattern <[who :] [dr .]> would locate the start of all entries introduced by *Who:* and starting with the honorific *Dr.*. Patterns can also contain wildcards, e.g. <Alph>/<ANum>/<Punc> match tokens that contain only alphabetic/alphanumeric/punctuation characters, while <*> matches any token.

Such specialized simple patterns will often reach high precision but low recall because there are many other ways to express a fact, especially in natural language texts. To address this issue, a large number of simple patterns are learned and their results combined. For this purpose BWI applies the technique of *boosting,* i.e. repeatedly applying the learning algorithm to the training data, each time adjusting the weight of training examples to emphasize those examples where the algorithm failed before.

A wrapper learned by boosting consists in a set F of "fore" and a set A of "aft" detectors (patterns that detect the start and the end of a field) and a length function $H(k)$ that estimates the maximum-likelihood probability that the field has length k. A text fragment $< i, j >$ is extracted if $F(i) \times A(j) \times H(j-i) > \tau$. A trade-off between precision and recall is possible by varying the threshold τ. Generally, BWI is biased toward precision, so setting $\tau = 0$ results in a reasonable recall at still high precision.

While one of the goals of BWI is to make WI algorithms suitable for free (unstructured) text, BWI still performs significantly worse on free text than on highly or partially structured text [27]. Most detectors learned from free text merely memorize specific training examples [27, Sec. 4.2.3]. Also the algorithm is biased towards overfitting to the particularities of the training data—the final rounds of boosting actually lower the reliability of the results. Both precision and recall on free texts can be increased by incorporating the output of a shallow parser into the model, splitting the text into a number of noun, verb, and prepositional phrase segments [27, Sec. 8.1].

3.6 Hybrid Approaches

IE^2: The *IE*^{2} system [1] submitted by SRA International to the MUC-7 conference is built in a highly modular way. The output of a standard named entity recognizer is complemented by a custom component that recognizes domain-specific entities (e.g. different kinds of vehicles). Another component recognizes domain-specific types of noun phrases and relations between them (e.g. *employee_of*, *location_of*). Both these components are based on hand-written rules, no learning is involved.

However, the IE^2 system goes further than most approaches described in this paper in also handling template unification beyond the sentence level. For coreference resolution, different strategies are employed: one strategy uses simple hand-written rules, but another one learns decision trees (using the standard implementation *C50*) from a tagged corpus. Optionally these strategies are combined in a hybrid method where the decision tree algorithm works on a subset of possible candidates chosen by the hand-written rules.

IE^2 also handles *event merging*, i.e. deciding whether or not two descriptions refer to the same event and can be merged. Here hand-written rules are combined with external knowledge sources to check the consistency of locations (*Miami* is in *Florida*) and times (can *Wednesday* and *tomorrow* refer to the same day within the current text?).

While in most aspects IE^2 is a typical representative of the classical hand-written rules approach that was dominant in the MUC conferences, its hybrid nature has interesting traits. Template unification and event merging beyond the sentence level are complex challenges that so far have been largely out of reach for learning systems. Combining trainable modules with external knowledge sources and specialized hand-written code could be a viable approach to tackle problems where single-paradigm solutions fail.

4 Knowledge-Based Approaches

4.1 Translation of Texts into Horn Clauses

One of the main purposes of IE is to make efficient search and transactions on extracted information possible. Therefore one of the central requirements on the target structure is an efficient querying mechanism. Another important issue is how expressive a target structure is. A relational target structure can for instance express only facts described by predicates but no conjunctions or disjunctions. The approach suggested by Delisle et al. [13] envisions Horn clauses as the representation resource of the target structure. Such a target structure is more expressive and more powerful in comparison with a relational target structure. Horn clauses allow conjunctions and implicit disjunctions and can be used to infer new facts, draw inferences about extracted information or build a domain theory. Generally, any logical techniques can be applied to this representation.

Since the extracted information is presented in logical form the application of this approach can be regarded as knowledge extraction. Remarkably, the information is extracted exhaustively, that is, everything identified as information is extracted. Such an approach differs significantly from fact extraction paradigm since neither a fix target structure is available nor any focus on items of interest can be given. However, this approach still can be related to IE since relevant (in this case all) information is found in an unstructured text and transferred into formal structure.

The approach does not presuppose any external knowledge sources except for continuous human interaction. To start semantic analysis, shallow parsing of the text is necessary. Parsed sentences are matched against a growing set of (potentially learned) cases. A case is a semantic interpretation of certain syntactic patterns, more precisely, an interpretation of dependencies between the verb and its arguments within a clause produced by a parser. Examples of cases are *Agent, Accompaniment, Location to, Time at.* The best match and thus semantic interpretation is suggested by a case analyzer and either confirmed or corrected by the user. If a significantly different structure of a clause (primitive sentence) is detected (compared to those stored in case pattern list), a new case is stored. Case assignment relies on a hypothesis that "syntax gives a reliable indication of meaning" (at least in examined technical texts).

At the beginning no predefined case patterns are necessary, however, user interaction increases in this case. Bigger elements of a sentence (clauses) are then matched against a restricted number of predefined patterns (causation, prevention, disjunction...) to identify their semantic relationship. The conjunction between the clauses plays thereby a significant role. Every time a match is found it is suggested to the user for confirmation. Additional syntactic information is regarded to label dependencies between the clauses. Source of domain information is the user. Cases are not assigned to "stative clauses" (clauses of form of the verb to be). These are processed directly by creating corresponding predicates. Translation from found semantic dependencies to Horn clauses occurs after clause dependencies, cases in the clauses and semantics of nouns (determined with WordNet) have been completely analyzed. Identified cases are translated into predicates reflecting the pattern structure of the case. Clause dependencies suggest how to piece together the Horn clause. For example, the sentence "Jim is a resident of Canada because he is serving abroad in the armed forces" would be transformed into:

```
is_resident_of(jim, canada) :- serve_agt_lat_benf(jim, abroad, armed_forces).
```

Identified Horn clauses are passed to the EBL (explanation based learning) module for building of domain concepts and developing domain theory.

The biggest strength of this approach is the resulting expressive and powerful target structure, which allows plenty of possibilities for further processing. The idea to directly transform identified cases into predicates and compose the Horn clause according to found clause relationships is very beneficial for the approach.

Since cases and clause relationships already contain semantic dependencies, Horn clauses can be constructed in a very efficient and consistent way. However, the way semantic relationships are derived from syntactic patterns is quite critical because the underlying assumption does not always definitely hold; especially, it is hardly applicable to other languages. Besides, case matching may fail as many clauses would be semantically ambiguous because of interpretation by syntactic structure.

The considerable involvement of human user is advantageous on the one hand since the system gets reliable information, but on the other hand may cause much effort given a text with big variety of case patterns. However, the learning component successfully applies learned cases and case triggers so that the amount of human interaction reduces with time. Another learning unit based on EBL is used in the last stage of processing Horn clauses.

The algorithm is explicitly designed for knowledge acquisition task and can hardly be applied to any other NLP tasks without serious modifications. Giving up the basic assumption and extending case patterns to include lexical data would possibly make it applicable to other languages. Since the main source of domain information is the user and no assumptions are made about the domain properties (except for technical texts) this approach could easily be adapted to various environments.

4.2 Ontology-Based Extraction

The goodness of the initial semantic and syntactic domain description is crucial for any rule and knowledge-based approach to IE. Usually this domain information does not represent any abstract logical dependencies and relations in the domain, since it is trimmed to IE purpose excluding comprehensive elements. Embley et al. [15] chose the ontology as one of the most explicit and complete knowledge representation forms trying to employ its possibilities to express deeper semantic relations.

However, using the ontology imposes some restrictions on the kind of processed texts: they have to belong to a quite narrow domain and contain many constants, which can potentially be extracted (e.g. proper names, numbers). The extraction algorithm relies on the existence of a manually created ontology; elements of the ontology cannot be used directly. Consequently tools for parsing and transforming the ontology in suitable form are required.

The user does not provide an explicit target structure. An ontology parser creates a relational database schema that serves as the target structure for consecutive extraction. Since the same ontology is used for all texts in a certain domain and the parser output is deterministic, one can conclude an existence of predefined, fix target structure, even though it has to be derived anew from the ontology for each domain. The parser produces also a list of constants and keyword rules that describes properties of relations in the ontology and their attributes or their occurrence in text. Possible value range and text occurrence are specified as a regular expression.

In the next step regular expressions are applied to the text to identify relevant items. One expression matches potentially several times, therefore matching text fragments are temporarily kept in a list for "candidate extractions". The decision whether an item will be extracted and what item will be chosen if there are several candidates is guided by heuristics. Heuristics test candidate items for proximity of a relevant keyword, overlapping etc. The item that is the next to a keyword and in case of overlapping items the subsuming one is chosen. If an ontology allows many values for an attribute of a relation all found matches are inserted. Otherwise, if the criteria mentioned above are not applicable, the first one is extracted and the rest ignored. The output is a database with extracted items.

In this approach the ontology represents the only resource and knowledge bundle necessary to process every text of the described domain. The possibility to derive necessary rules or information for extraction from the ontology makes the approach flexible. An ontology contains also predicates describing entity relationships between entities and inference rules. This additional semantic information can be used for more reliable identification of desired facts in a text.

However, the structure of the ontology used in this approach goes far beyond the conventional notion including representational aspect and regular expressions. It is basically a collection of related resources. The extraction suffers from the fact that regular expressions cannot match non-trivial natural language expressions or whole sentences because of their complexity, so some items will not be extracted because they cannot be identified. Moreover, the used regular expressions are not changed or updated according to the results of extraction, learning mechanisms are not employed. Manual creation of ontologies is very tedious and hard to manage for bigger domains.

In the current form the ontology-based approach can handle listings, enumerations, generally preformatted text elements, but not complete sentences. It can be enhanced if creation of regular patterns is not static and manually specified, but can be dynamically influenced by the semantic level of the ontology. Currently the subject of extraction is mainly numbers and proper names. To cope with the variety of natural language the system should be able to extract other parts of speech. Changing elements of an ontology based on the extraction results would ease the adjustment of the ontology to domain texts.

4.3 Thesaurus-Based Extraction

The *TIMES* system developed by Bagga and Chai [2] requires a number of knowledge sources: the WordNet thesaurus, a general English dictionary, a domain-specific dictionary, and a gazetteer of location names. Texts are preprocessed with a tokenizer, a sentence splitter, an entity recognizer that identifies named and numeric entities, and a partial parser that recognizes noun, verb, and prepositional groups with their respective head words. The preprocessing components are based on finite-state rules.

Training is done by a user through a graphical interface. For each of the head words identified by the parser, the user selects the appropriate sense (concept) if WordNet defines several senses for this word. Then the user builds a *semantic* *network* to represent the content of each training text. Selected head words from the text are stored as nodes or relations within the network. For example, from the phrase *IBM Corp. seeks jobs candidates in Louisville*, the user might build a relation *seek* between two nodes *IBM Corp.* and *job candidate*.

The text-specific extraction rules created this way are then generalized according to the hypernym/hyponym (super-/subordinate terms) relations defined in WordNet. Generalization replaces a term by its hypernym n steps higher in the WordNet hierarchy. For named entities (NE), the category determined by the NE recognizer is generalized. E.g. *IBM Corp.* is identified as a company—generalizing this concept one step yields business, concern; three steps yields organization. A generalized rule matches any terms that are hyponyms of the generalized term. Increasing the generalization level results in higher recall at the cost of precision, because the generalized rules find instances missed by specialized rules but also produce more false positives.

In later versions of the system, the user only has to mark the target information to extract from a text. The system automatically builds relations between the marked information and generalizes extraction rules to the most suitable level [7].

The hypernyms of a word are sense-dependent. Rules for sense disambiguation of head words are learned from the user-provided word senses [7]. Rules for selecting a word sense contain a number of constraints for the phrases in the context of the word to classify. Each constraint (match function) determines the syntactic type (noun, verb or prepositional phrase) of a phrase and the token value, semantic type (named/numeric entity type) and word sense of its head word. More general rules contain fewer constraints. The system retains only rules whose precision (ratio of correctly identified word senses) on the training data exceeds a predefined threshold. When several rules fire, the rule with the highest precision wins; when none fires, the most frequent word sense is chosen.

5 Statistical Approaches

5.1 Probabilistic Parsing

The *SIFT* system [35, 36] submitted to MUC-7 is one of the earliest statistical approaches to IE. The system simultaneously handles part-of-speech (POS) tagging and parsing (syntactic annotations) as well as NE recognition and the finding of relationships (semantic annotations), so the results of each task can influence the others. Relationships link two entities of different types, e.g. in *GTE Corp. of Stamford* there is a location-of relation between the company and the city. The system was trained from the Penn Treebank corpus (1,000,000 words) for syntactic annotations and a domain-specific annotated corpus (500,000 words) for semantic annotations.

Tasks are primarily performed at the sentence level. In a final step, entity coreferences are resolved beyond the sentence level (trained from coreference annotations of the semantic corpus) and cross-sentence relationships are established.

The domain-specific corpus requires only semantic tagging (entities, coreferences, relationships), no syntactic annotations. After training the syntax model from the Penn Treebank, it is applied to the domain-specific corpus to produce parses that are consistent with the semantic annotations. The result is a single parse tree that contains both syntactic (e.g. S: sentence, VP: verb phrase) and semantic (e.g. per: person entity, emp-of: employee-of relationship) annotations. The sentence-level model is then retrained on the resulting joint annotations to produce an integrated model of syntax and semantics. Named entities are recognized by a Hidden Markov Model.

The statistical model predicts the categories and POS tags of constituents based on the data from the parse-tree context. The category of a head constituent depends on the category of the parent node; of a modifier on the category of the previous modifier and the parent node and its head constituent as well as on the head word itself. The POS tag of a modifier depends on the modifier itself and on the head word and its POS tag.

The probability of a whole augmented parse tree is the product of the probabilities of all components. The most likely augmented parse tree is found by a chart parser that proceeds bottom-up. Dynamic programming techniques and pruning are used to keep the search space feasible. Maximum likelihood estimates for all probabilities are obtained from the frequencies in the training corpus, using Witten-Bell smoothing to compensate data sparseness.

For determining whether a relation exists between two elements in different sentences, the cross-sentence model calculates the probabilities that a relation does or does not exist and chooses the more probable alternative. These probabilities are calculated on the assumption of feature independence. The considered features comprise *structural features* (the distance between the entities, whether one of the entities was referred to in the first sentence of an article) and *content features* (e.g. whether entities with similar names—probable coreferences—or with similar descriptors are related in other contexts).

The results of the SIFT system were close to those of the best (hand-written) systems in MUC-7.

5.2 Hidden Markov Models

Active Hidden Markov Models: The algorithm developed by Scheffer et al. [48, 49] learns Hidden Markov Models (HMMs) from sparsely (partially) labeled texts. Their HMM algorithm tags each token (word) in a document with one of a set of predefined tags, or the special tag none—the tags (to find) are the hidden states of the Markov Model, while the observed tokens are the visible output of the model. The state (tag) sequence minimizing the per-token error is found using the forward-backward algorithm. Thus the observation sequence *John Smith, extension 7343* should correspond to the state sequence *(firstname, name, none, phone)*. Attributes of each token store the output of preprocessing tools—e.g. the POS tag, word stem, or the surrounding HTML element.

For training the model, partially labeled documents where some of the tags are unspecified are sufficient. The remaining unknown tags are estimated using the Baum-Welch algorithm. Active learning is used to select the most "difficult" untagged tokens for hand-tagging by the user. The tokens with the lowest difference between the probabilities of the two most probable states are considered most difficult.

HMMs Learned by Stochastic Optimization: The state-transition structure of HMMs is often chosen manually. Freitag and McCallum [24] employ stochastic optimization for this purpose. The algorithm performs hill-climbing starting from a simple model and splitting states until a (locally) optimal state-transition structure has been found. The performance of each model is evaluated on a validation set.

The approach employs a separate HMM for each slot type (e.g. *seminar speaker*) in a document. Each model contains two types of states, *target states* that produce the tokens to extract and *non-target states*.

The Baum-Welch algorithm is used to estimate transition and emission probabilities of each tested model. To increase the reliability of estimated emission probabilities, the technique of *shrinkage* [23] is used. Parameter estimates from sparse states in a complex model are "shrinked" towards estimates from related states in a simpler model where more training data is available for each state (because the number of states is lower). All target states are considered as related, as are all non-target states.

A weighted average learned through Estimation-Maximization is used to combine the estimates of different models. The smoothed, shrinkage-based probability of state s emitting word w is $\lambda_1 P(w|s) + \lambda_2 P(w|a(s)) + \lambda_3 P(\frac{1}{K})$, where the last term represents the uniform distribution, $a(s)$ is the parent state of s (a state combining all target states if s is a target state, a state combining all non-target states otherwise), and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

For learning a suitable HMM structure, non-target states are further differentiated as either *prefix* or *suffix* (preceding resp. following a target phrase) or *background* states (anything else). The most simple HMM fitting this structure has four states (one of each kind) and considers exactly one prefix + suffix around each target state.[1]

This model is used as the starting point for hill climbing. Related models are generated by *lengthening* a prefix, suffix, or target string (adding a new state of the same kind that must be traversed before the model can proceed to the next kind of state), by *splitting* a prefix/suffix/target string (creating a duplicate where the first and last states of the duplicated prefix/suffix/target have the same connectivity as in the original), or *adding* a background state. Model variations are evaluated on a hold-out set or via 3-fold cross-validation.

Hierarchical Hidden Markov Models: Skounakis et al. [51] use *hierarchical HMMs* (HHMMs) [17] for IE. HHMMs combine several levels of states to describe a

[1] The background state is connected to itself and to the prefix state which is in turn connected to the target state, the target state is connected to itself and to the suffix state which is connected to the background state.

sequence at different granularity levels. A two-level HHMM is used—the top level models phrase segments (noun, verb, and prepositional phrases) provided by a shallow parser, the lower level models individual words (including their POS tags) within a phrase.

The Viterbi, Forward, and Backward algorithms are adapted to ensure that the embedded word model reaches the end state exactly at the end of a each phrase and to ensure the typing of the phrase model (each state has a type that corresponds to the type of the phrase segment it emits).

Context hierarchical HMMs (CHHMMs) are an extended variant that incorporate additional sentence structure information in each phrase. The word model is extended to consider the left and right neighbor of each word, generating a sequence of overlapping *trigrams*. To reduce the number of possible observations, individual features (words and tags) are combined under the assumption of conditional independence.

Evaluation shows superior results for hierarchical models, especially CHHMMs, compared with flat HMMs.

Generally, HMMs offer a simple yet powerful way to model text that has proved very successful in various areas of language processing. However, the generative nature of HMMs makes it hard to capture multiple interdependent sources of information. The approaches described in the following section address this problem by switching to sequential models that are conditional instead of generative.

5.3 Maximum Entropy Markov Models and Conditional Random Fields

The *Maximum Entropy Markov Models (MEMMs)* used by [33] are a conditional alternative to HMMs. MEMMs calculate the conditional probability of a state (tag) given an observation (token) and the previous state (tag). Thus the two parts of an HMM—calculating the probability of a state depending on the previous one (transition function) and calculating the probability of an observation depending on the current state (observation function)—are collapsed into a single function.

Observations can comprise many features which need not be independent. Features are binary, e.g. *the word "apple"*, *a lower-case word* etc. The actually used features are selected and weighted by maximum entropy (ME) modeling. Generalized Iterative Scaling (GIS) is used to train the parameters of the model. The most probable tagging sequence is found using a variation of Viterbi search adjusted for MEMMs.

A variation of the Baum-Welch algorithm can be used to estimate missing tags (states) during training, so the model can be trained from partially labeled or even unlabeled documents. No experimental results of doing this are reported though.

A disadvantage of associating observations with state transitions instead of states is the high number of parameters: $|S|^2 \times |O|$ instead of the $|S|^2 + |S| \times |O|$ of classical HMMs ($|S|$ is the number of states, $|O|$ of observations). This increases the risk of data sparseness.

Tested on a text segmentation task, MEMM performs significantly better than both classical HMMs and a stateless maximum entropy model.

A weakness of MEMMs is the *label bias problem:* the probability mass arriving at a state must be distributed among the successor states, thus outgoing transitions from a state compete only against each other, not against other transitions. This results in a bias in favor of states with fewer outgoing transitions. *Conditional Random Fields (CRFs)* [28, 34] address this problem by modeling the joint probability of an entire sequence of labels in a single exponential model instead of modeling the conditional probabilities of next states in per-state exponential models.

CRFs are undirected graphical models (a.k.a. *random fields* or *Markov networks*) that calculate the conditional of values on designated output variables depending on other designated input variables.

$$P(y|x) = \frac{1}{Z_x} \prod_{c \in C} \Phi_c(x_c, y_c)$$

is the conditional probability of output values y given input values x. $Z_x = \sum_{y'} \prod_{c \in C} \Phi_c(x_c, y_c)$ is the normalizer (partition function), C is the set of all cliques, $\Phi_c(\cdot)$ is the potential function for clique c, x_c and y_c are the sub-sets of the variables in x resp. y that participate in clique c.

CRFs have been employed for preprocessing tasks such as part-of-speech (POS) tagging [28] and for IE subtasks such as coreference resolution [32].

5.4 Token Classification

There are multiple approaches that employ standard classification algorithms, modeling information extraction as a token classification task. These systems split a text into a series of tokens and invoke a trainable classifier to decide for each token whether or not it is part of an slot filler of a certain type (e.g. *speaker* or *location* of a seminar).

Combination Strategies: To re-assemble the classified tokens into multi-token slot fillers, various *combination strategies* (or tagging strategies) can be used. The trivial (*Triv*) strategy would be to use a single class for each slot type and an additional “O” class for all other tokens. However, this causes problems if two entities of the same type immediately follow each other, e.g. if the names of two *speakers* are separated by a linebreak only. In such a case, both names would be collapsed into a single entity, since the trivial strategy lacks a way to mark the begin of the second entity.

For this reason (as well as for improved classification accuracy), various more complex strategies are employed that use distinct classes to mark the first and/or last token of an slot filler. The two variations of *IOB* tagging are probably most common: the variant usually called *IOB2* classifies each token as the begin of a slot filler of a certain type (B-*type*), as a continuation of the previously started slot filler, if any (I-*type*), or as not belonging to any slot filler (O). The *IOB1* strategy differs from *IOB2* in using B-*type* only when necessary to avoid ambiguity (i.e. if two same-type entities immediately follow each other); otherwise I-*type* is used even at the beginning of slot fillers. While the *Triv* strategy uses only $n + 1$ classes for n slot types, *IOB* tagging requires $2n + 1$ classes.

BIE tagging differs from *IOB* in using an additional class for the last token of each slot filler. One class is used for the first token of a slot filler (B-*type*), one for inner tokens (I-*type*) and another one for the last token (E-*type*). There are two variations that differ in the handling of slot fillers consisting in a single token (which is thus both begin and end): *BIE1* simply assigns the begin class (B-*type*), while *BIE2* uses a fourth class BE-*type* to mark them specially.[2] Thus $3n + 1$ classes are used by *BIE1*, $4n + 1$ by *BIE2*.

The strategies discussed so far require only a single classification decision for each token (through often multiple binary classifiers are used concurrently instead of a single multi-class classifier). Another option is to use two separate classifiers, one for finding the begin and another one for finding the end of slot fillers. *Begin/End* tagging requires $n + 1$ classes for each of the two classifiers (B-*type* + O for the first, E-*type* + O for the second). In this case, there is no distinction between inner and outer (other) tokens. Complete slot fillers are found by combining the most suitable begin/end pairs of the same type, e.g. by taking the length distribution of slots into account.

Classification Algorithms: There are various approaches that employ a classification algorithm with one of the combination strategies described above: [8] uses *Maximum Entropy (MaxEnt)* modeling with *BIE2* tagging; [59] uses *Memory-based Learning (MBL)* with the *IOB1* strategy; the *TIE* system [50] pairs the *Winnow* algorithm [31] with *IOB2*. Since Winnow is an online algorithm that can learn from a single pass over the training data, *TIE* support *incremental learning*, i.e. the extraction model can be updated on-the-fly without requiring a full retraining. However, better results are reported for batch training (multiple passes over the training data).

ELIE [18, 19] uses two Support Vector Machines (SVM) for *Begin/End* tagging. Highly improved results are reached by augmenting this setup with a second level *(L2)* of begin/end classifiers. The *L2* end classifier focuses on finding suitable end tags for matching left-over begin tags from the level-1 begin classifier (and vice versa). While the *L1* classifiers are trained on a very high number of tokens, almost all of which are negative instances (O), the *L2* classifiers only consider the near context of left-over *L1* begin/end tags which allows a more focused classification. In this way, the recall of the system can be increased without seriously affecting the precision.

While token-classifying approaches lack the genuinely sequential nature of HMMs and conditional models, they have proved very successful (cf. Sec. 6.5),

[2] Note that the actual names used to identify classes do not matter and can deviate from those used in the explanation; what matters is the chosen partitioning of tokens into classes.

due to their ability to combine rich feature representations of the tokens to classify with powerful classification algorithms.

5.5 Fragment Classification and Bayesian Networks

SNoW-IE: Roth and Yih [46] employ the *Winnow*-based *SNoW* classifier in a two-stage architecture. Among a small number of possible candidate fragments identified in the *filtering* stage, the (presumably) correct text fragment is determined and extracted in the *classifying* stage. The two-stage architecture allows using a rich feature representation in a second step for the small subset of promising candidates which would be infeasible (or very inefficient) to use for all possible fragments.

Rich context representations are created by encoding certain relational structures in propositional representations. In the first phase, only single word tokens and POS tags and collocations of two adjoint words/tags (bigrams) are used as features. For words and tags in the left and right context window, the relative position is encoded in the feature. In second phase, "sparse collocations" of words/tags from left and right window and target phrase are also considered. A sparse collocation of n elements generates an n-gram feature for each subsequence of elements $v_i \dots v_j$, $1 \leq i < j \leq n$.

In the version presented in [46], a different classifier is trained for each entity type in each phrase—dependencies between different types are not considered. When several classifiers choose identical fragments for extraction, the more confident classifier (higher activation value) wins.

But relations between entities can yield important hints for determining the exact entity type. Thus the approach has been modified to recognize entities and relations between them at the same time [47]. Borders of entities and existence of relations must be given, but their types are established in a joint step, by maximizing the joint likelihood of all type assignments in a *belief network* (Bayesian network), based on original estimates given by SNoW classifiers.

The mathematical model does not allow loops—different relations are assumed to be independent and entity types are assumed to be independent of relationship types. Another limitation of this approach is that entity borders must be known in advance and cannot be changed.

BIEN: BIEN (Bayesian Information Extraction Network) [41] is another approach utilizing Bayesian networks. For preprocessing, words are lemmatized (stemmed) and POS tagged and sentences are split into flat syntactic chunks (noun, verb, prepositional, and other phrases). Additional features are provided by capitalization, word length, and several gazetteers (location identifiers, popular names).

Dynamic Bayesian networks (DBNs) represent previous decisions to model the order of events ("flow of time"), generalizing Hidden Markov Models. The BIEN system is based on a DBNS that classifies each token as belonging to one of the target field types or to the background (hidden variable Tag). Another hidden variable (Last Target) stores the last recognized target field type, reflecting the order in which target information is expressed. The Viterbi algorithm is used for classification (determining the most likely sequence of Tag variables); the EM algorithm is used for training the model.

6 Comparison of Selected Approaches

In this section we compare the approaches according to the types of tasks and texts they can handle as well as the types of features they consider. We also compare tagging requirements and learning characteristics. The final subsection discusses quantitative evaluation and presents evaluation results on two standard corpora. Table 1 in Sec. 1.3 can be consulted to locate the detailed descriptions of approaches and systems.

6.1 Types of Tasks Handled

The main IE task is to fill a *template* that contains several *slots*, which is typically done in two steps:

- *Slot filling* or *single-slot extraction* to find suitable fillers for the defined slots.
- *Template unification* or *multi-slot extraction* to combine the found slot fillers into templates, resolving coreferences as required.

Most approaches described in this paper handle the first step only. Hence they are limited to corpora where each document contains a single template; otherwise additional pre- or postprocessing is necessary to split the input at template boundaries or to arrange the found slot fillers into adequate templates.

Some systems—*Crystal*[3], *Whisk* and *TIMES*—handle multi-slot extraction at the sentence level. Thus no special processing is necessary if each template is expressed within a single sentence in a input text. This might be sufficient for some domains but it is not a general solution to the template unification task.

Other approaches go further by unifying templates at a logical level, beyond sentence borders: the *Amilcare* extension of *(LP)*^{2}, *IE*^{2}, *SIFT*, and the extended version of *SNoW-IE*.

A significant difference can be observed in the requirements on target structures: the approaches presented in sections 3.1 and 4.1 generate templates dynamically; all other approaches require a predefined target structure. The dynamic construction of target structures reduces the human effort necessary to adapt a new domain; however, in many cases, automatically deduced structures will be less appropriate than hand-modeled ones.

6.2 Types of Texts Handled

Three types of texts are often distinguished (cf. [54, Sec. 1], [14, Sec. 2.5]):

[3] Crystal does not identify exact slot fillers but only sentence constituents containing slot fillers, thus it always requires postprocessing.

– *Free texts* are grammatical natural-language texts, e.g. newspaper articles or scientific papers.
– *Semi-structured texts* are not fully grammatical and sometimes telegraphic in style, e.g. newsgroups or email messages or classified ads.
– *Structured texts* contain textual information strictly following a predefined (but not necessarily known) format where items are arranged in a fixed order and separated by delimiter characters or strings. Examples are comma-separated values or web pages generated from a database.

Even though some systems are designed for certain types of texts, it cannot be assumed that some class of IE approaches is particularly suitable for a particular kind of text. Furthermore, all classes have in common that the performance on structured texts is better than on free texts.

Some approaches—*TANKA/MaLTe*, the original version of *Crystal*[4], *IE*^{2}, *TIMES* and *SIFT*—rely heavily on linguistic information and are thus suitable for free texts only. The system described by Embley et al. [15] is restricted to structured texts. Most other approaches are suitable for both free and semi-structured texts—they make use of linguistic information as far as it is available, but do not necessarily require it.

Most other systems make little or no use of linguistic knowledge, thus they are suited for semi-structured and structured texts. *Whisk*, *SRV* and *BWI* claim to be targeted at any text type, from free text to structured text. Approaches that allow variable input will play a major role in the future research, since in real world domains an IE system will be confronted with the large diversity of texts.

6.3 Considered Features

There is a wide variety in the types of features that are considered for learning by different approaches. All systems utilize the words (tokens) in a text as the main lexical features. Not only the presence or absence of a word but also the word order play an important role. Morphological information is used not quite as universally, but very frequently. Especially POS (part-of-speech) tags are used by a wide variety of systems. Some systems also utilize a stemmer or lemmatizer to determine the base forms of words.[5]

For linguistic information beyond the word level, several approaches[6] rely on simple chunkers that identify various types of clauses (noun, verb, prepositional clauses etc.) in a sentence. More refined chunk parsers that also assign grammatical roles for chunks (subject, direct or indirect object) are employed by the systems presented in Sec. 3.1 and by *Crystal* and *Whisk* (for free texts). Only two systems, *SRV* and *TANKA/MaLTe*, make use of a deep parser. Rule and knowledge-based systems tend to embed more syntactic information since

[4] [53] describes an extension to semi-structured text.
[5] *(LP)*^{2}, *TIE* and *BIEN*, optional for *Active HMMs*.
[6] Such as *TIMES*, *(C)HHMMs*, *TIE*, *BIEN*, and the extended version of *BWI*.

syntax is often used for rule construction. Statistical systems consider predominantly linguistic information related to single tokens due to their token-based processing of the text.

Semantic information is used less frequently than syntactic. Typically, it comprises simple gazetteers or word lists assigning semantic classes to words.[7] Some approaches[8] use a complete thesaurus, WordNet [16]. Knowledge-based systems use their own built-in knowledge-bases.

Some approaches[9] consider features derived from the shape of words/tokens, e.g. token type (lower-case, capitalized, all-caps, digits, etc.) or prefixes and suffixes. Most approaches work on plain text input without formatting, but a few can utilize structural information from HTML or XML documents: *Stalker* and *BWI* can handle HTML tags (treating them as normal tokens), *Active HMMs* optionally consider the HTML context of text tokens, *TIE* creates structured context representations based on the DOM tree of XML documents.

While usually the handled types of features are fixed in advance, the *Amilcare* system chooses an adaptive way to consider linguistic information ("LazyNLP"): the amount of linguistic information available for learning rules is gradually increased until the effectiveness of the generated rules stops improving.

The three main classes of IE approaches differ significantly in the amount of used features. Knowledge-based approaches utilize comparably few features restricting them on semantic and syntactic information. Some statistical systems try to exploit all available information about text elements generating relatively big amount of features. Rule-based systems tend to rely heavily on linguistic features for rule generation.

6.4 Tagging Requirements and Learning Characteristics

Most approaches require training texts to be fully tagged, i.e. all items to extract must be marked (either embedded within the texts or in external documents). Full tagging of a large number of documents is a serious burden. Some systems alleviate this requirement by using *active learning* on partially tagged texts (the extended version of *Rapier*, *Whisk*, *Stalker* in Co-Testing setting, *Active HMMs*). *TIE* is the only system that allows *incremental learning*, i.e. the extraction model can be updated on-the-fly without requiring a full retraining. The approaches described in Sections 3.1, 3.4 and 4 utilize human review and interaction instead of postulating pretagged texts.

The general trend should go towards relaxing the input requirements on the training texts by incorporating better learning models. Statistical systems partially succeed in processing not fully consistent text corpora, while rule-based and knowledge-based systems rely on traditional elaborately prepared text resources.

[7] Used by *Riloff's system*, *Crystal*, *(LP)^2*, *TIMES*, *TIE*, and *BIEN* for various word classes.

[8] *Rapier*, *SRV*, *TIMES*.

[9] *SRV*, *BWI*, *MEMM*, *TIE*.

6.5 Quantitative Comparison

Evaluation Metrics: The most commonly used metrics for quantitative evaluation of IE systems are *precision* and *recall*; the joint *F-measure* combines them both in a single figure. For each slot type, results are evaluated by counting *true positives tp* (correct slot filters), *false positives fp* (spurious slot filters), *false negatives fn* (missing slot filters) and calculating *precision* $P = \frac{tp}{tp+fp}$ and *recall* $R = \frac{tp}{tp+fn}$. The *F-measure* is the harmonic mean of precision and recall:

$$F = \frac{2 \times P \times R}{P + R}.$$

For a corpus containing multiple slot types, there are several ways to combine results of all types into a single measure. The *microaverage* is calculated by summing the respective *tp*, *fp* and *fn* counts for all types and then calculating P, R, and F over the summed counts. Thus slot types that occur more frequently have a higher impact on the joint measure than rare types. On the other hand, the *macroaverage* is calculated by computing the mean of all type-specific P and R values, so all types are considered of equal importance, no matter how often they occur.

A disadvantage of the *microaverage* is that is depends on knowing the raw counts, which are hardly ever published in research papers. This is addressed by a related metric, the *weighted average* proposed by [8]: here each slot type is weighted by the total number of slot fillers of this type in the corpus. These numbers can be determined by inspecting a corpus, allowing comparisons with other systems evaluated on the same corpus even if no raw counts have been published.

Evaluation Methodology: As discussed in [29, 30], there are several issues that need to be addressed to allow a fair comparison of different systems, some of which have often been neglected in previous IE evaluations. An important issue is the size of the split between training and testing set (e.g. 50/50 or 80/20 split) and the procedure used to determine partitions (n-fold cross-validation or n random splits).

Another issue is how to compare predicted answers (slot fillers) with the expected (true) answers. Typical options are to require that all occurrences of a slot in a document should be found ("one answer per occurrence") or to expect only a single answer per slot which is considered most likely to be correct ("one answer per slot"). The latter option is useful if multiple answers for the same slot are expected to be synonymous (e.g. "2pm" and "2:00 pm"), the former if each occurrence is assumed to contain relevant new information. A less frequently used option would be "one answer per different string" where multiple occurrences of the same string are collapsed into a single occurrence, i.e. different positions in the document are ignored.

Table 2. F-Measure Results on the Seminar Announcements Corpus

Approach	**Slots**	**BWI**	**ELIE/L2**	**HMM**	**(LP)^2**	**MaxEnt**	**MBL**	**SNoW-IE**	**TIE**
Reference		[22]	[18]	[23]	[9]	[8]	[59]	[46]	[50]
etime	*228*	93.9	96.4	59.5	95.5	94.2	96	96.3	**97.5**
location	*464*	76.7	86.5	83.9	75.0	82.6	**87**	75.2	80.6
speaker	*409*	67.7	**88.5**	71.1	77.6	72.6	71	73.8	85.2
stime	*485*	**99.6**	98.5	99.1	99.0	**99.6**	95	**99.6**	99.3
Weighted Avg		83.9	**92.1**	81.7	86.0	86.9	86.6	85.3	89.9
Macroaverage		84.5	**92.5**	78.4	86.8	87.3	87.3	86.2	90.7

Seminar Announcements Corpus: While there is no universal standard corpus that has been used to evaluate all (or most) IE approaches, several reference corpora have been used quite frequently.

The most frequently used IE corpus is probably the *CMU Seminar Announcements*[10] (SemAnn) corpus. The corpus contains 485 seminar announcements (plain text files) collected from university newsgroups; the task is to extract up to four slots from each document (if present): *speaker*, *location*, *start time (stime)* and *end time (etime)* of the talk.

Generally, training and test sets of equal size are used (50/50 split) and results are averaged over five or sometimes ten random splits. There are no predefined random splits for this corpus, so each system uses their own. The page model published with the corpus prescribes that the "one answer per slot" method should be used. Table 2 lists the best known results published on this corpus.[11] The last two rows contain the weighted average (based on the number of existing slot fillers given in column 2) and the macroaverage.

It is noteworthy that the systems reaching best results on this corpus (*ELIE* and *TIE*) are statistical classification-based systems (cf. Sec. 5.4). There are only two rule-learning systems, BWI and (LP)^2, among the eight best systems, and their performance is inferior to that of the best statistical systems.

Job Postings Corpus: Another frequently used corpus is the *Job Postings* collection of Mary E. Califf [3]. The corpus consists of 300 job offers posted to a Usenet newsgroup. The tasks defines 17 slots of information to extract about job offers (job *title*, *company*, *recruiter*, *salary* etc.) and postings (message *id*, *post date*). Sadly, evaluation methodologies used on this corpus vary wildly. Some authors use 10-fold cross-validation while others use a 50/50 training/test split averaged over 10 random splits. Another ([54]) uses only a subset of 100 randomly selected documents for his tests, while others ([12]) use an extended corpus that contains 600 documents. The original description of the corpus seems to suggest that "one answer per occurrence" is expected but is not quite clear about this.

[10] Accessible from [45], a corrected version with some minor annotation errors fixed is available at `http://nlp.shef.ac.uk/dot.kom/resources.html`.

[11] One other statistical approach, *BIEN* [41], is not directly comparable, since it uses an 80/20 split instead of 50/50. BIEN reaches an weighted average F-measure of 88.9%.

Table 3. F-Measure Results on the Job Postings Corpus

Approach	**Slots**	**ELIE/L2**	**RAPIER**	**(LP)^2**	**DeSitter**	**SNoW-IE**
id	*301*	99.7	97.5	**100.0**	97	99.7
title	*252*	**55.8**	40.5	43.9	36	52.7
company	*91*	**79.5**	70.0	71.9	57	75.4
salary	*107*	66.3	67.4	62.8	62	**72.9**
recruiter	*167*	82.0	68.4	80.6	53	**85.3**
state	*235*	**92.7**	90.2	84.7	86	91.7
city	*269*	**95.1**	90.4	93.0	89	89.0
country	*138*	**95.8**	93.2	81.0	95	95.5
language	*516*	**91.4**	80.6	91.0	26	82.5
platform	*469*	79.8	72.5	**80.5**	32	74.1
application	*367*	69.7	69.3	**78.4**	30	60.9
area	*658*	48.7	42.4	**66.9**	16	51.6
req-years-exp	*154*	80.0	67.1	68.8	62	**83.9**
des-years-exp	*44*	82.9	**87.5**	60.4	41	79.0
req-degree	*82*	79.0	81.5	**84.7**	35	83.5
des-degree	*21*	55.2	**72.2**	65.1	35	60.9
post date	*298*	97.5	**99.5**	**99.5**	91	99.2
Weighted Avg		78.6	72.9	**79.8**	49.9	76.4
Macroaverage		**79.5**	75.9	77.2	55.5	78.7

An overview of results reached on this corpus is given in Table 3 (based on [30] and [18]). However, due to the inconsistent evaluation methodologies and testing sets, they must be treated with caution. The statistical *ELIE* system and the rule-learning *(LP)*^{2} seem to be very close to each other. However, it is likely that *(LP)*^{2} was set up to evaluate only "one answer per slot" instead of the "one answer per occurrence" setup used by *ELIE*. This would explain the apparently better performance on fields such as *platform*, *application*, and *area*, which occur multiple times in many documents. Other fields such as message *id* and *post date* are highly regular (part of the message metadata), which explains the superior results of the rule-based system.

Other Corpora: There are various other corpora, many of which can be found in the *RISE Repository* [45]. The most popular of these is probably the *Corporate Acquisitions* corpus, another corpus which comes from the same source as the Seminar Announcements corpus, the PhD thesis of Dayne Freitag [20]. It contains 600 annotated articles about mergers and acquisitions from the *Reuters-21578* corpus. The task is to extract the full and abbreviated names of the parties to an acquisition, the location of the acquired company, the price paid and information about the status of negotiations. Another important corpus is a collection of *Apartment Rentals* created by Stephen Soderland [54]. However, none of these corpora has been used for evaluation of as many trainable systems as those detailed above.

7 Conclusion

Focus of adaptive methods is quite diverse and reaches from accomplishing subtasks of IE to complete IE systems. Adaptive approaches to information extraction comprise methods that apply experience acquired in training or knowledge gained from external resources (such as thesaurus or user) to establish some kind of a domain model that is capable of locating relevant contents in domain texts and matching them to the target structure of the domain. The structure of the model determines to a large degree the features of an approach and is therefore the main criterion for its classification. Rule-based approaches learn the model as a set of pattern-driven extracting rules, statistical approaches build formal mathematical representations, while knowledge-based approaches establish explicit logical models.

The principle of adaptive approaches originates from the endeavor to reduce the amount of hand-coded domain knowledge. However, the approaches still require human contribution in various forms depending on their class. Human knowledge is utilized as explicit knowledge sources (gazetteers, ontologies etc.), examples specifying what to extract by identifying relevant content in training text or in form of human supervision interacting directly with the system during correction of the extraction proposals. While statistical and rule-based approaches rely on the latter, knowledge-based approaches focus mainly (but not exclusively) on the former. Arguably, the annotation of texts is often less costly than the explicit formalization of knowledge, which makes statistical or rule-based approaches more attractive for adaptation to domains where no explicit knowledge sources are available. On the other hand, knowledge-based approaches allow easier re-use of existing formalizations.

Rule learning approaches try to exploit the regularity in expressions of certain information to find common linguistic patterns that match these expressions. The majority of approaches use rule learning techniques to acquire the patterns. During the learning process the existing incomplete, erroneous or insufficiently general patterns are improved using the feedback of a human supervisor or annotations of the training corpus. The interconnection between the learned patterns and the action transferring the relevant content in the target structure explicitly or implicitly constitutes the learned rule. A frequently found limitation is the rudimentary learning mechanisms that do not provide enough generalization capabilities. Often the shortcomings and weak results are camouflaged by restricting the complexity of the target structure. Developing more profound models will be an important research direction for rule learning methods. Another serious drawback of the actual extraction is that it typically happens within sentence boundaries or within the scope of predefined instances.

Statistical approaches basically reduce the IE task to the prediction problem. In the simple case every text token is classified as some attribute of the target structure or not relevant. Thereby they utilize training data very effectively being able to learn the correct prediction even from quite limited numbers of examples. Knowledge-based approaches are much stronger supported by external resources and consider additionally grammatical parse trees and lookup of semantic classes, either flat (dictionaries, gazetteers, entity recognizers) or arranged in hierarchies (thesauri, ontologies).

Due to their heavy reliance on linguistic information, knowledge-based approaches are suitable for grammatical texts ("free texts") only, while most statistical approaches can also handle "semi-structured texts" that are not fully grammatical and sometimes telegraphic in style, e.g. newsgroups or email messages or classified ads. Therefore they are more robust with respect to textual irregularities such as typographical errors or ungrammatical text. Statistical approaches typically process a single word at a time and combine the results, while knowledge-based approaches focus on whole sentences. A common trend of both types of approaches is that early systems (*TANKA/MaLTe*, *SIFT*) tend to be more ambitious, while later development narrows down on more pragmatic and realizable goals.

Even though remarkable progress has been achieved in the field of IE in recent time, especially making the systems more autonomous and universally applicable, many problems remain hardly or not yet tackled. The majority of approaches is not able to recognize and extract multiple occurrences of complex facts. The systems either cannot handle complex facts at all assuming the target structure to be a flat slot sequence or it is presupposed that every document contains at most one instance of the same complex fact. The major difficulty arises when fragments of information belonging together are scattered in different sometimes distant parts of a text and it has to be determined whether and which fragment belongs to what complex entity. Even the most advanced approaches can hardly handle unification of partial facts beyond the sentence boundaries. The research challenge will be to find a way of reassembling composite information without establishing a complete logical representation of the text content.

A similar problem with very different background occurs when the same instance of a fact repeatedly appears in different forms. Due to the richness of natural language we can refer to an entity in very many ways. However, only one occurrence containing the most complete information should be extracted. Current systems often do not recognize that text fragments refer to the same fact and extract them as a new found instance. The most straightforward solution not yet practiced would be to embed the mechanisms of coreference resolution in the extraction algorithm and develop strategies for selecting the most appropriate occurrence.

Information extraction suffers from uncertainty of the natural language. Often facts are expressed with a certain degree of tentativeness (e.g. indirect speech: "someone reported that...", "s.o. assumed that..."). In such cases even for humans it is difficult to decide whether the information is factual and hence relevant. An important task will be to determine the degree of reliability of information, which is possible deploying fuzzy methods.

Probably the hardest problem of IE is to identify implicitly expressed information. However, implicitness of information can be of different origin too. Saying "The furious battle between pirates and government troops ended in the crack of dawn as the pirates raised the white flag" we have to consult our world knowledge and retrieve the fact that white flag is a symbol for a defeat to conclude, who won the battle. Consider, on the other hand, “She lived in Berlin. James was born in the small homonymous town in Texas” where it is sufficient to know the semantics of the word “homonymous” to infer James’ place of birth. Surprisingly, some rule-based and statistical systems even now in simple cases can capture the implicit semantics in linguistic patterns resp. statistical context models as if it were an explicitly expressed fact. Two prerequisites must be fulfilled: the fact has to occur somewhere in the text explicitly (comp. “Berlin” in the last example) and there must be sufficiently many examples for this kind of implicit reference. More general solutions, however, would require different approaches which so far are not in sight.

In this context a very important and interesting question is whether the more profound embedding of semantic analysis will contribute to the advance in IE. Recent successes in single-slot information extraction reached by statistical systems that almost completely forgo semantic resources and analysis suggest that it may be dispensable. But does the good performance for the simplest of IE tasks open optimistic perspectives for much more difficult tasks or have the statistical approaches already reached their limit? One development perspective consists in the combination of statistical extraction systems with algorithms for normalization, coreference resolution and instance unification that rely more heavily on external knowledge bases or rules.

Future research will have to face these problems and questions to extend the practical usefulness of IE systems to a broader range of applications.

References

[1] C. Aone, L. Halverson, T. Hampton, and M. Ramos-Santacruz. SRA: Description of the IE2 system used for MUC. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[2] A. Bagga and J. Y. Chai. A trainable message understanding system. In *CoNLL*, pages 1–8. 1997.

[3] M. E. Califf. *Relational Learning Techniques for Natural Language Extraction.* PhD thesis, University of Texas at Austin, 1998.

[4] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998.

[5] M. E. Califf and R. J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[6] C. Cardie. A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 798–803. AAAI Press, 1993.

[7] J. Y. Chai and A. W. Biermann. The use of word sense disambiguation in an information extraction system. In *AAAI/IAAI*, 1999.

[8] H. L. Chieu and H. T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)*, pages 786–791, 2002.

[9] F. Ciravegna. $(LP)^2$, an adaptive algorithm for information extraction from Web-related texts. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, USA, 2001.

[10] F. Ciravegna and A. Lavelli. LearningPinocchio: Adaptive information extraction for real world applications. In *Proceedings of the 2nd Workshop on Robust Methods in Analysis of Natural Language Data (ROMAND 2002)*, Frascati, Italy, 2002.

[11] R. Collier. Automatic template creation for information extraction, an overview. Technical report, University of Sheffield, 1996.

[12] A. De Sitter and W. Daelemans. Information extraction via double classification. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-2003)*, 2003.

[13] S. Delisle, K. Barker, J.-F. Delannoy, S. Matwin, and S. Szpakowicz. From text to Horn clauses: Combining linguistic analysis and machine learning. In *10th Canadian AI Conf.*, 1994.

[14] L. Eikvil. Information extraction from World Wide Web – A survey. Technical Report 945, Norwegian Computing Center, 1999.

[15] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddl. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *Conference on Information and Knowledge Management (CIKM)*, pages 52–59, 1998.

[16] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[17] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[18] A. Finn and N. Kushmerick. Information extraction by convergent boundary classification. In *AAAI-2004 Workshop on Adaptive Text Extraction and Mining*, San Jose, USA, 2004.

[19] A. Finn and N. Kushmerick. Multi-level boundary classification for information extraction. In *ECML 2004*, pages 111–122, 2004.

[20] D. Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Carnegie Mellon University, 1998.

[21] D. Freitag. Toward general-purpose learning for information extraction. In C. Boitet and P. Whitelock, editors, *Proc. 36th Annual Meeting of the Association for Computational Linguistics*, pages 404–408, San Francisco, CA, 1998.

[22] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.

[23] D. Freitag and A. K. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[24] D. Freitag and A. K. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, pages 584–589, 2000.

[25] J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[26] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM—semi-automatic creation of metadata. In A. Gomez-Perez and V. R. Benjamins, editors, *Proc. 13th International Conference on Knowledge Engineering and Management*, 2002.

[27] D. Kauchak, J. Smarr, and C. Elkan. Sources of success for information extraction methods. Technical Report CS2002-0696, UC San Diego, 2002.

[28] J. Lafferty, A. K. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[29] A. Lavelli, M. Califf, F. Ciravegna, D. Freitag, C. Giuliano, N. Kushmerick, and L. Romano. A critical survey of the methodology for IE evaluation. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, 2004.

[30] A. Lavelli, M.-E. Califf, F. Ciravegna, D. Freitag, C. Giuliano, N. Kushmerick, and L. Romano. IE evaluation: Criticisms and recommendations. In *AAAI-2004 Workshop on Adaptive Text Extraction and Mining*, San Jose, USA, 2004.

[31] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[32] A. McCallum and B. Wellner. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[33] A. K. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, 2000.

[34] A. K. McCallum and D. Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *IJCAI'03 Workshop on Learning Statistical Models from Relational Data*, 2003.

[35] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, R. Weischedel, and the Annotation Group. Algorithms that learn to extract information—BBN: Description of the SIFT system as used for MUC. In *MUC-7*, 1998.

[36] S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. A novel use of statistical parsing to extract information from text. In *ANLP-NAACL*, pages 226–233, 2000.

[37] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[38] I. Muslea, S. Minton, and C. A. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.

[39] U. Y. Nahm and R. J. Mooney. Using information extraction to aid the discovery of prediction rules from text. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, Boston, MA, 2000.

[40] C. Nobata and S. Sekine. Towards automatic acquisition of patterns for information extraction. In *International Conference of Computer Processing of Oriental Languages*, 1999.

[41] L. Peshkin and A. Pfeffer. Bayesian information extraction network. In *IJCAI*, 2003.

[42] J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3,4):287–312, 1995.

[43] E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 1044–1049. The AAAI Press/MIT Press, 1999.

[44] E. Riloff and M. Schmelzenbach. An empirical approach to conceptual case frame acquisition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[45] RISE repository. `http://www.isi.edu/info-agents/RISE/`.

[46] D. Roth and W.-t. Yih. Relational learning via propositional algorithms: An information extraction case study. In *IJCAI*, 2001.

[47] D. Roth and W.-t. Yih. Probabilistic reasoning for entity & relation recognition. In *COLING'02*, 2002.

[48] T. Scheffer, C. Decomain, and S. Wrobel. Active hidden Markov models for information extraction. In *Proceedings of the International Symposium on Intelligent Data Analysis*, 2001.

[49] T. Scheffer, S. Wrobel, B. Popov, D. Ognianov, C. Decomain, and S. Hoche. Learning hidden Markov models for information extraction actively from partially labeled text. *Künstliche Intelligenz*, (2), 2002.

[50] C. Siefkes. Incremental information extraction using tree-based context representations. In A. Gelbukh, editor, *Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2005)*, Lecture Notes in Computer Science, pages 510–521. Springer, 2005.

[51] M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden Markov models for information extraction. In *IJCAI*, 2003.

[52] S. Soderland. *Learning Text Analysis Rules for Domain-specific Natural Language Processing*. PhD thesis, University of Massachusetts, Amherst, 1997.

[53] S. Soderland. Learning to extract text-based information from the World Wide Web. In *Proc. Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 251–254, 1997.

[54] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1–3):233–272, 1999.

[55] S. Soderland. Building a machine learning based text understanding system. In *Proc. IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[56] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: Inducing a conceptual dictionary. In C. Mellish, editor, *Proc. 14th International Joint Conference on Artificial Intelligence*, pages 1314–1319, San Francisco, 1995.

[57] K. Sudo, S. Sekine, and R. Grishman. Automatic pattern acquisition for Japanese information extraction. In *HLT2001*, 2001.

[58] C. A. Thompson, M. E. Califf, and R. J. Mooney. Active learning for natural language parsing and information extraction. In *Proc. 16th International Conf. on Machine Learning*, pages 406–414, 1999.

[59] J. Zavrel and W. Daelemans. Feature-rich memory-based classification for shallow NLP and information extraction. In J. Franke, G. Nakhaeizadeh, and I. Renz, editors, *Text Mining, Theoretical Aspects and Applications*, pages 33–54. Springer Physica, 2003.

View Integration and Cooperation in Databases, Data Warehouses and Web Information Systems*

Hui Ma[1], Klaus-Dieter Schewe[1], Bernhard Thalheim[2], and Jane Zhao[1]

[1] Massey University, Department of Information Systems & Information Science Research Centre,
Private Bag 11 222, Palmerston North, New Zealand
{h.ma, k.d.schewe, j.zhao}@massey.ac.nz
[2] Christian Albrechts University Kiel,
Department of Computer Science and Applied Mathematics,
Olshausenstr. 40, D-24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract. View integration aims at replacing a set of existing views by a single new one in such a way that with respect to information capacity the new view dominates or is equivalent to the old ones. Therefore, in this article we first investigate a theory of schema equivalence and dominance for the higher-order Entity-Relationship model (HERM) based on the notion of computable queries. We then develop formal transformation rules for schema integration that are embedded in a pragmatic method telling how they should be applied for integration.

We then apply the approach to views, which occur as the basic constituents for user interfaces as formalised by the notion of dialogue type. In two follow-on steps we apply the rule-based view integration technique to data warehouses and web information systems. In the case of data warehouses the fundamental idea is the separation of input from operational databases and output to on-line analytical processing (OLAP) systems. Both the extraction of data from the operational databases and the definition of the data-marts for OLAP can be formulated by views.

In the case of web information systems, views form the core of media types, which provide abstract means for describing content, functionality, context and adaptivity to user preferences and intentions, end-devices, and channel limitations. In this case the queries defining the views must be highly expressive, as they must involve the creation of abstract identifiers, complex values and links. We extend the transformation rules to cope with these requirements.

View cooperation provides an alternative to view integration in which the integrated view is only virtual. That is the constituting views are kept and exchange functions are designed to provide the same functionality as if the views were integrated.

Keywords: view integration, schema equivalence, data warehouses, web information systems, view cooperation

* The work in this paper was supported by FRST/NERF grant MAUX0025 "DIMO – Distributed Multi-Level Object Bases" and MU/ABRF grants 57413 "Foundations of Conceptual Modelling" and 57501 "Distributed Data Warehouses".

1 Introduction

Database *schema integration* is an old issue that has attracted a lot of research [12,13,14,16,30]. The starting point for schema integration is a set of schemata over some data models. Usually the focus is on two schemata over the same data model. If the underlying data models differ, then we may assume some preprocessing transforming both schemata – or one of them, if this is sufficient – into an almost equivalent schema over a data model with higher expressiveness. Then schema integration aims at replacing the given schemata by a single new one in such a way that the new schema dominates or is equivalent to the old ones.

A view on some database schema consists of another schema called the target schema, and a defining query, which maps instances of the source schema to instances of the target schema. If we integrate the target schemata of views we talk of *view integration* [3,29,30]. In this case we obtain embeddings for each target schema into the new schema. If these embeddings are coupled with the defining queries for the given views we obtain a new defining query, i.e. we obtain not only an integrated target schema, but an integrated view.

So the first thing we need is a theory of schema equivalence and dominance. Roughly, we may say that two schemata are equivalent, if they have the same *information capacity*, i.e. we may store the same information in both schemata. A schema *dominates* another one, if it has a larger information capacity. Various formal definitions for equivalence and dominance have been given [9,19,30] and compared in [15,16]. Here we extend this work and introduce novel definitions for equivalence and dominance based on *computable queries* [5,34], i.e. we base the transformation functions on the most general notion of query, but we discard total arbitrariness, as computable queries must respect isomorphisms. We compare the new notions with the ones defined in [9,19,30]. We base this comparison on the higher-order Entity-Relationship model (HERM) [30].

On this basis we then reconsider schema integration following the framework in [15,16]. In a nutshell, we first "clean" the given schemata by removing name conflicts, synonyms and homonyms, then we add inter-schema constraints, which leads to a single constrained schema, which is just the union of the given ones. To this schema we then apply formal equivalence transformation or augmentation rules, which will finally take us to an integrated schema that is either equivalent to the union of the given ones or dominates this.

One group of rules addresses the restructuring of the complex attributes, entity types, relationship types and clusters. Another group of rules considers the shifting of attributes over hierarchies and paths. A third group of rules deals with selected integrity constraints such as keys, functional, inclusion and join dependencies, cardinality constraints and path constraints. A fourth group of rules is devoted to aggregation, decomposition, specialisation and generalisation.

The transformation rules can be used as well for *view cooperation* [15,28,30], which provides an alternative to view integration in which the integrated view is only virtual. That is the constituting views are kept and exchange functions are designed to provide the same functionality as if the views were integrated.

Views are important for user interfaces. In [21] an integrated concept of *dialogue type* was developed, which combines views with operations defined on them. As conceptual interface specifications are a viable source during the systems development process, the problem of view integration comes up naturally. With respect to the transformation rules, we now need an extension that deals with the operations.

Data Warehouses are data-intensive systems that are used for analytical tasks in businesses such as analysing sales/profits statistics, cost/benefit relation statistics, customer preferences statistics, etc. The term used for these tasks is "on-line analytical processing" (OLAP) [33] in order to distinguish them from operational data-intensive systems, for which the term "on-line transaction processing" (OLTP) has become common. The idea of a data warehouse [11] is to extract data from operational databases and to store them separately. Thus, a first problem in data warehouse design is to integrate views from various source databases. This point of view of data warehouse design as a view integration problem has been strongly promoted in [12,32,36]. In fact, there is another related problem in here, the support of OLAP applications by materialised views, which in their totality reflect the data warehouse. In this sense data warehouses also involve a view design problem.

In [17] dialogue types have been applied to data warehouses and OLAP systems. The principle result states that OLAP functionality corresponds to operations on views over the data warehouse. So we may apply our extended transformation rules to this problem. In fact, the main idea of data warehouses implies a separation of input from operational databases and output to views that contain the data for particular OLAP tasks. This implies a three-layer architecture for data warehouses and OLAP systems [38], which is the basis of a formal design approach for such systems [26,37].

In the field of web information systems (WISs) the importance of views is commonly accepted [2,4,24,25]. In [7,6] the notion of *media type* has been introduced. A media type is an extended view over some underlying database. However, different to the approaches in [2,4] the defining query already constructs the navigation structure. Furthermore, media types are prepared for dynamic WISs [18] by adding operations, and for adaptivity by adding cohesion preorders. As a consequence, transformation rules for the purpose of media type integration have to extend the view integration rules in a way that cohesion is covered as well. In this article we will develop these extensions.

Outline. We start in Section 2 with a look at published work on the topic of this article. In Section 3 we describe the basics of HERM emphasising our approach to schema equivalence and dominance. Section 4 is devoted to the process of schema and view integration. We describe first the process in general followed by a presentation of the transformation rules for this process. We also show how these rules can be used for view cooperation instead of integration. Section 5 applies the integration framework to data warehouses and OLAP systems. We emphasise the particular case of view integration and indicate additional rules dealing with the adaptation of dialogue operations. In Section 6 we briefly present media types, i.e. other extended views that are used for web information systems. We particularly focus on the aspect of adaptivity for which we discuss cohesion. This is followed by an extension to the transformation rules dealing with the implications of view integration to cohesion. We conclude with a short summary and discussion of further research directions.

2 Related Work

The work on view integration in [13,14,29] is based on the Entity-Relationship model. Larson et al. [14] consider containment, equivalence and overlap relations between attributes and types that are defined by looking at "real world objects". Equivalence between types gives rise to their integration, containment defines a hierarchy with one supertype and one subtype, and overlapping gives rise to new common supertype. The work by Spaccapietra and Parent [29] considers also relationships, paths and disjointness relations between types. The work by Koh et al. [13] provides additional restructuring rules for the addition or removal of attributes, generalisation and specialisation, and the introduction of surrogate attributes for types.

The work by Biskup and Convent in [3] is based on the relational data model with functional, inclusion and exclusion dependencies. The method is based on the definition of integration conditions, which can be equality, containment, disjointness or selection conditions. Transformations are applied aiming at the elimination of disturbing integration conditions. In the same way as our work it is based on a solid theory. On the other hand, it has never been applied to large systems in practice. The approach by Sciore et al. in [28] investigates conversion functions on the basis of contexts added to values. These contexts provide properties to enable the semantic comparability of values.

The work by Lehmann and Schewe [15,16] assumes that the given schemata are defined on the basis of the higher-order Entity-Relationship model (HERM) [30] which is known to provide enough expressiveness such that schemata existing in practice can be easily represented in HERM. The work relies on the notions of equivalence and dominance as defined for HERM in [30].

In [15,16] these notions of equivalence and dominance are also compared with those defined by Hull [9] and Qian [19]. Basically, the four different notions of schema dominance introduced by Hull differ by the way the transformation functions are defined. In the simplest case (calculus dominance) they correspond to calculus queries, whereas in the most general case (absolute dominance) there are no restrictions at all. In fact, taking computable queries will remove the arbitrariness from absolute dominance that has been criticised in [19] while taking into account the most general form of queries [5,34].

The design of data warehouses and OLAP applications has been intensively studied. Widom [36] and Theodoratos [32] emphasise that data warehouse design is mainly a view integration problem, Whereas Lewerenz et al. [17], Thomson [33], and Zhao and Schewe [26,37,38] also take the OLAP functionality into account.

The design and development of WISs has attracted a lot of attention. The ARANEUS framework in [2] emphasises that conceptual modelling of web information systems should approach a problem triplet consisting of content, navigation and presentation. This leads to modelling databases, hypertext structures and page layout. OOHDM [27] is quite similar to ARANEUS, but its origins are not in the area of databases but in hypertext and it explicitly refers to an object oriented approach. OOHDM emphasises an object layer, hypermedia components, i.e. links, and an interface layer. The work in [8] also starts from hypertext design. The work introduces "authoring in the large", i.e., the conceptual modelling of information elements and navigation, and uses this to categorise different types of links. Another similar approach is WebML [4], which emphasises a multi-level architecture for the data-driven generation of WIS, thus takes the view aspect into account. Furthermore, it emphasises structures, derivation and composition, i.e. views, navigation and presentation, thus addresses the same problem triplet as the ARANEUS framework.

Our own work on WIS design combines the usage-oriented storyboarding methodology [22,23,31] and the content- and functionality-oriented theory of media types [6,20,24], which are embedded in an integrated co-design methodology based on an abstraction layer model [25].

3 Schemata, Equivalence and Dominance

As we will base our presentation on the higher-order Entity-Relationship model (HERM) [30], we start with a brief review of the model as far as it is important for our purposes here. In particular, we focus on algebraic and logical query languages for HERM. These will be needed to address the important issue of defining schema dominance and equivalence in a way that the expressiveness is sufficient for the integration of extended views as needed for data warehouses and web information systems. The basic case dealing only with plain views over HERM schemata, i.e. ignoring the extensions by operations, adaptivity, etc. was already handled in [15,16].

3.1 HERM

The major extensions of the HERM compared with the flat ER model concern nested attribute structures, higher-order relationship types and clusters, a sophisticated language of integrity constraints, operations, dialogues and their embedding in development methods. Here we only review some of the structural aspects.

In the following let $\mathcal{A}$ denote some set of *simple attributes*. Each simple attribute $A \in \mathcal{A}$ is associated with a base domain $dom(A)$, which is some fixed countable set of values. The values themselves are of no particular interest.

In HERM it is permitted to define nested attributes. For this take a set $\mathcal{L}$ of *labels* with the only restriction that labels must be different from simple attributes, i.e. $\mathcal{L} \cap \mathcal{A} = \emptyset$.

Definition 3.1. A *nested attribute* is either a simple attribute, the null attribute $\perp$, a tuple attribute $X(A_1, \dots, A_n)$ with pairwise different nested attributes A_i and a label $X \in \mathcal{L}$ or a set attribute $X\{A\}$ with a nested attribute A and a label $X \in \mathcal{L}$. Let $\mathcal{NA}$ denote the set of all nested attributes.

We extend *dom* to nested attributes in the standard way, i.e. a tuple attribute will be associated with a tuple type, a set attribute with a set type, and the null attribute with $dom(\perp) = \mathbb{1}$, where $\mathbb{1}$ is the trivial domain with only one value.

In principle we could also permit other constructors than tuple and set constructors, e.g. constructors $\langle \cdot \rangle$ and $[\cdot]$ for multisets and lists. This would, however, only complicate our presentation here without leading to additional insights. We therefore disregard these other constructors.

On nested attributes we have a partial order $\geq$ defined as follows.

Definition 3.2. $\geq$ is the smallest partial order on $\mathcal{NA}$ with

– $A \geq \perp$ for all $A \in \mathcal{NA}$,
– $X\{A\} \geq X\{A'\} \Leftrightarrow A \geq A'$ and
– $X(A_1, \dots, A_n) \geq X(A'_1, \dots, A'_m) \Leftrightarrow \bigwedge_{1 \leq i \leq m} A_i \geq A'_i$.

A *generalised subset* of a set $F \subseteq \mathcal{NA}$ of nested attributes is a set $G \subseteq \mathcal{NA}$ of nested attributes such that for each $A' \in G$ there is some $A \in F$ with $A \geq A'$.

It is easy to see that $X \geq X'$ gives rise to a canonical projection $\pi^X_{X'} : dom(X) \to dom(X')$.

Let us now define the entity and relationship types and clusters in HERM using the following compact definition.

Definition 3.3. A *level-k-type* R consists of a set $comp(R) = \{r_1 : R_1, \dots, r_n : R_n\}$ of labelled components with pairwise different labels r_i, a set $attr(R) = \{A_1, \dots, A_m\}$ of nested attributes and a key $key(R)$. Each component R_i is a type or cluster of a level at most $k - 1$, but at least one of the R_i must be level-$(k-1)$-type or -cluster.

For the key we have $key(R) = comp'(R) \cup attr'(R)$ with $comp'(R) \subseteq comp(R)$ and a generalised subset $attr'(R)$ of the set of attributes.

A *level-k-cluster* is $C = R_1 \oplus \cdots \oplus R_n$ with pairwise different components R_i, each of which is a type of a level at most k. At least one of the R_i must be level-k-type.

The labels r_i used in components are called *roles*. Roles can be omitted in case the components are pairwise different. A level-0-type E – here the definition implies $comp(E) = \emptyset$ – is usually called an *entity type*, a level-k-type R with $k > 0$ is called a *relationship type*.

A *HERM schema* is a finite set $\mathcal{S}$ of entity types, relationship types and clusters together with a set Σ of integrity constraints defined on $\mathcal{S}$. We write $(\mathcal{S}, \Sigma)$ for a schema, or simply $\mathcal{S}$, if $\Sigma = \emptyset$.

Note that the notion of level has only been introduced to exclude cycles in schemata. Besides this it has no other meaning. Conversely, if there are no cycles in a schema, then there is a straightforward way to assign levels to the types and clusters in the schema such that the conditions in Definition 3.3 are satisfied.

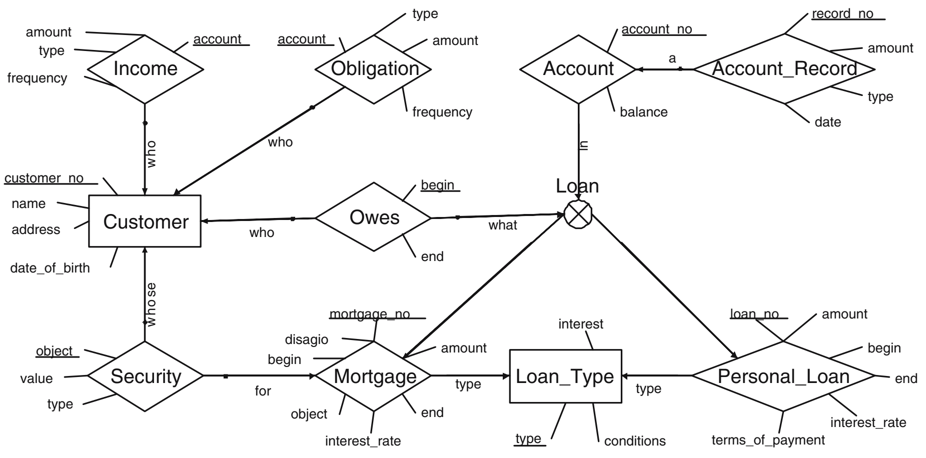


Fig. 1. HERM diagram for loan application

Example 3.1. The following type definitions define a HERM schema for a loan application as it might be used by some bank:

LOAN_TYPE = (∅, { type, conditions, interest }, { type })

CUSTOMER = (∅, { customer_no, name, address, date_of_birth },
{ customer_no })

PERSONAL_LOAN = ({ type : LOAN_TYPE }, { loan_no, amount,
interest_rate, begin, end, terms_of_payment }, { loan_no })

MORTGAGE = ({ type : LOAN_TYPE }, { mortgage_no, amount, disagio,
interest_rate, begin, end, object }, { mortgage_no })

LOAN = HOME_LOAN ⊕ MORTGAGE

ACCOUNT = ({ ln : LOAN }, { account_no, balance }, { account_no })

ACCOUNT_RECORD = ({ a : ACCOUNT }, { record_no, type, amount,
date }, { a : ACCOUNT, record_no })

OWES = ({ who : CUSTOMER, what : LOAN }, { begin, end },
{ who : CUSTOMER, what : LOAN, begin })

SECURITY = ({ whose : CUSTOMER, for : MORTGAGE }, { value, object,
type }, {whose : CUSTOMER, for : MORTGAGE, object })

INCOME = ({ who : CUSTOMER }, { type, amount, frequency, account },
{ who : CUSTOMER, account })

OBLIGATION = ({ who : CUSTOMER }, { type, amount, frequency,
account }, { who : CUSTOMER, account })

For this it is easy to see that the types CUSTOMER and LOAN_TYPE are on level 0, because they do not have components. Level-1-types are INCOME, Obligation, Mortgage and Personal_Loan, because all their components are on level 0. Consequently, the cluster Loan ist a level-1-cluster. The types Owes, Security and Account are then level-2-types, because all components are on level 1 or below, and finally, Account_Record is a level-3-type.

Figure 1 provides a graphical representation of the schema in Example 3.1. We call this a *HERM diagram.* According to the common convention in Entity-Relationship diagrams we represented types on level 0 by rectangles and types on higher levels by diamonds. Clusters are represented by $\bigoplus$. We use directed edges from a relationship type to each of its components, and from clusters to their components, and undirected edges between types and their attributes. Roles names are attached to the directed edges. Keys are marked by underlining attributes and marking the edges that correspond to components in the key by some dot.

As each HERM schema can be represented by such a HERM diagram, i.e. by a graph, we can apply all graph-theoretic notions. In particular, when we talk of *paths* in a HERM schema, we mean a path in the underlying undirected graph that results from ignoring the orientation of edges in the HERM diagram.

In order to define the semantics of HERM schemata we concentrate on identifier-semantics, also known as pointer-semantics. For this assume a countable set ID of identifiers with $ID \cap D = \emptyset$ for all domains D used for simple attributes. Furthermore, we associate with each type $R \in \mathcal{S}$ a *representing attribute*

$$X_R = R(X_{r_1}, \dots, X_{r_n}, A_1, \dots, A_n)$$

with new simple attributes X_{r_i} for each role r_i and $dom(X_{r_i}) = ID$, as well as a *key attribute*

$$K_R = R(X_{r_{i_1}}, \dots, X_{r_{i_\ell}}, A'_1, \dots, A'_m)$$

for key$(R) = \{r_{i_1} : R_{i_1}, \dots, r_{i_\ell} : R_{i_\ell}, A'_1, \dots, A'_m\}$. Obviously, we have $X_R \geq K_R$.

Definition 3.4. An *instance* of a HERM schema $(\mathcal{S}, \Sigma)$ is a family $\{db(R)\}_{R \in \mathcal{S}}$ of finite sets. For each type R the set $db(R)$ consists of pairs (i, v) with $i \in ID$ and $v \in dom(X_R)$ subject to the following conditions:

- Identifiers are globally unique, i.e. whenever $(i, v_1) \in db(R_1)$ and $(i, v_2) \in db(R_2)$ hold, we must have $R_1 = R_2$ and $v_1 = v_2$.
- Key values are locally unique, i.e. whenever $(i_1, v_1), (i_2, v_2) \in db(R)$ hold with $\pi^{X_R}_{K_R}(v_1) = \pi^{X_R}_{K_R}(v_2)$, then we must have $i_1 = i_2$.
- Roles are always defined, i.e. whenever $(i, v) \in db(R)$ and $\pi^{X_R} R(X_{r_{i_j}})(v) = i'$ for $r_{i_j} : R_{i_j} \in \text{comp}(R)$, then $(i', v') \in db(R_{i_j})$ for some $v' \in dom(X_{R_{i_j}})$.
- The integrity constraints in Σ are satisfied.

For each cluster $C = R_1 \oplus \cdots \oplus R_k$ the set $db(C)$ is the disjoint union of the sets $db(R_i)$ $(i = 1, \dots, k)$.

We write $inst(\mathcal{S}, \Sigma)$ for the *set of all instances* over $(\mathcal{S}, \Sigma)$.

3.2 Query Languages for HERM

As for the relational data model, basic queries against a HERM schema can be formulated both in an algebraic and a logical way. We will extend both the simple HERM algebra and the HERM calculus in a way that we can express more queries, but let us start with first-order queries.

Definition 3.5. The *HERM algebra* $\mathcal{H}$ provides the operations σ_φ (selection) with a selection formula φ, $\pi_{A_1,\dots,A_m}$ (projection) with a generalised subset $\{A_1,\dots,A_m\}$, ϱ_f (renaming) with a renaming function f, $\bowtie_G$ (join) with a common generalised subset G, $\cup$ (union), $-$ (difference), $\nu_{X:A_1,\dots,A_n}$ with attributes $A_1,\dots,A_n$ (nest), and μ_A (unnest) with a set attribute A.

As the details of these operations are not much different from the relational data model, we omit the details and refer to [30].

However, in order to make the HERM algebra operational for our purposes here, we need a little extension:

- We permit new type names R to be added to $\mathcal{S}$, and use assignments $R :=$ *exp* with a HERM algebra expression *exp*. Then applying such a query to an instance of $(\mathcal{S},\Sigma)$ results in an instance over $(\mathcal{S}\cup\{R\},\Sigma)$. The type or cluster definition for R is implicitly determined by the expression *exp*.
- In order to satisfy the global uniqueness of identifiers, such an assignments involve the non-deterministic creation of identifiers in *ID* for the pairs (i,v) in the added $db(R)$. Such identifier creation has been investigated thoroughly in [35].
- While sequences of assignments only extend the schema, we also allow to drop types or clusters.

In summary, a *HERM algebra program* P has the form $C_1;\dots;C_r \setminus \mathcal{S}'$, where each C_i is an assignment, say $R_i : +exp_i$ that extends the schema by R_i and the instance by $db(R_i)$, while $\mathcal{S}'$ is a subschema of $\mathcal{S}\cup\{R_1,\dots,R_r\}$. Thus, P defines a mapping $q(P)$ taking instances over $(\mathcal{S},\Sigma)$ to instances over $(\mathcal{S}',\Sigma')$, though the set Σ' of constraints is left implicit.

The algebra may be further extended with `WHILE`, in which case we talk of the *extended HERM algebra* $\mathcal{H}_{ext}$. In this case we add constructs of the form

$$\texttt{WHILE}\ \text{change}\ \texttt{DO}\ \ C_1;\dots;C_r\ \texttt{END}.$$

We obtain a logical perspective from the following simple observation. Each type $R \in \mathcal{S}$ – or more precisely, its representing attribute X_R – defines a variable in a higher-order logic. The order depends on the depth of nesting of the attributes in $\mathrm{attr}(R)$. For instance, if the set constructor is not used, we obtain a first-order variable. In fact, we obtain a *type* for each such variable, where types correspond to nested attributes. Thus, a schema defines a signature, and each instance defines a finite structure for this signature.

There are a few subtleties to be aware of. First, for clusters we have to allow particular union variables to cope with the requirement to have disjoint unions.

Second, we have to define first the logic and then permit only those structures that are in fact models for the theory defined by the restrictions in Definition 3.4.

Now use further variables and constants of any available type and define atoms as follows:

- A *predicative atom* has the form $X(t_1, \dots, t_n)$ with a higher-order variable X and *terms*, i.e. variables or constants, $t_1, \dots, t_n$ such that the types of the t_i and the one of X match properly.
- An *equational atom* has the form $t_1 = t_2$ with terms of the same type or of different types, where one is a cluster and the other one is one of its components.

Finally, use the usual connectives $\wedge$, $\vee$, $\rightarrow$, $\exists$ and $\forall$ to define *HERM logic*. Then the concept of free and bound variables is defined as usual. We write $fr(\varphi)$ for the set of free variables of a formula φ.

Definition 3.6. A *HERM calculus query* has the form $X(x_0, \dots, x_n) : \varphi$ with a formula φ of HERM logic such that $fr(\varphi) \subseteq \mathcal{S} \cup \{x_1, \dots, x_n\}$ and $\{x_1, \dots, x_n\} \subseteq fr(\varphi)$.

Obviously, we may interpret a formula φ provided we are given a value assignment $\sigma(x_i)$ for all the variables $x_1, \dots, x_n$ and an instance over $(\mathcal{S}, \Sigma)$. If according to this interpretation the formula φ is interpreted as true, we obtain a tuple $(\sigma(x_0), \dots, \sigma(x_n))$ with a new identifier $\sigma(x_0) \in ID$. The result is the set of all such tuples, and will be bound to variable X.

It can be shown that such HERM calculus queries express exactly the same as HERM algebra queries with non-deterministic identifier creation and assignments to new type variables. Using sequences and a fixed-point construction yields the same expressiveness as the extended HERM algebra. We use the term *extended HERM calculus* for this approach to query languages.

Let us finally extend the discussion on queries to *computable queries* in the spirit of [5]. Computable queries are defined on a semantic rather than a syntactic level. A query language that can express all computable queries is called *complete*, but this has to be verified. It is known that the calculus and algebra queries defined so far (even in their extended form) are not complete.

Roughly speaking, a computable query is a query that commutes with isomorphisms. So we have to define a concept of isomorphism for HERM instances. For this consider bijections $\psi_A : dom(A) \rightarrow dom(A)$ for each simple attribute $A \in \mathcal{A}$ and $\psi_{ID} : ID \rightarrow ID$. We call this family $\Psi = \{\psi_A\}_{A \in \mathcal{A} \cup \{ID\}}$ a *t-isomorphism*. Obviously, we may extend Ψ to all nested attributes defining

$$\psi_{X(A_1,\dots,A_n)}(v_1, \dots, v_n) = (\psi_{A_1}(v_1), \dots, \psi_{A_n}(v_n))$$

and

$$\psi_{X\{A\}}(\{v_1, \dots, v_n\}) = \{\psi_A(v_1), \dots, \psi_A(v_n)\}.$$

In particular, a t-isomorphism Ψ induces a mapping $\Psi_{\mathcal{S}} : inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}, \Sigma)$.

Definition 3.7. Two instances db_1 and db_2 over $(\mathcal{S}, \Sigma)$ are called *isomorphic* (notation: $db_1 \simeq db_2$) iff there exists a t-isomorphism Ψ with $\Psi_{\mathcal{S}}(db_1) = db_2$.

A *computable query* on a database schema $(\mathcal{S}, \Sigma)$ with output schema $(\mathcal{S}', \Sigma')$ is a partial recursive function $q : inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}', \Sigma')$ mapping isomorphic databases to isomorphic databases, i.e.,

$$db_1 \simeq db_2 \wedge q(db_1) \downarrow \Rightarrow \ q(db_2) \downarrow \wedge \ q(db_1) \simeq q(db_2),$$

where $q(db_1) \downarrow$ means that the partial recursive function q is defined on db_1.

3.3 Schema Dominance and Equivalence

As already stated schema and view integration requires precise notions for schema dominance and equivalence, which we will introduce now.

Definition 3.8. A HERM schema $(\mathcal{S}', \Sigma')$ *dominates* another HERM schema $(\mathcal{S}, \Sigma)$ by means of the language $\mathcal{L}$ (notation: $(\mathcal{S}, \Sigma) \sqsubseteq_{\mathcal{L}} (\mathcal{S}', \Sigma')$) iff there are mappings $f : inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}', \Sigma')$ and $g : inst(\mathcal{S}', \Sigma') \rightarrow inst(\mathcal{S}, \Sigma)$ both expressed in $\mathcal{L}$ such that the composition $g \circ f$ is the identity.

If we have $(\mathcal{S}, \Sigma) \sqsubseteq_{\mathcal{L}} (\mathcal{S}', \Sigma')$ as well as $(\mathcal{S}', \Sigma') \sqsubseteq_{\mathcal{L}} (\mathcal{S}, \Sigma)$, we say that the two schemata are *equivalent* with respect to $\mathcal{L}$ (notation: $(\mathcal{S}, \Sigma) \cong_{\mathcal{L}} (\mathcal{S}', \Sigma')$).

According to our discussion of HERM query languages in the previous subsection we obtain three different notions of dominance and equivalence. $\sqsubseteq_{\mathcal{H}}$ and $\cong_{\mathcal{H}}$ refer to the use of the HERM algebra or equivalently the HERM calculus as the language, in which the transformations f and g are to be expressed. Analogously, $\sqsubseteq_{\mathcal{H}_{ext}}$ and $\cong_{\mathcal{H}_{ext}}$ refer to the use of the extended HERM algebra or the extended HERM calculus. Finally, $\sqsubseteq_{comp}$ and $\cong_{comp}$ refer to the use of computable queries. According to these choices we talk of *HERM dominance*, *extended HERM dominance* and *computable dominance*, respectively.

For completeness, let us mention a fourth alternative. If we do not impose any restrictions on the language $\mathcal{L}$, i.e. f is any injective and g any surjective mapping, then we obtain *absolute dominance* $\sqsubseteq_{abs}$ and *absolute equivalence* $\cong_{abs}$. In the following sections we will always refer to $\sqsubseteq_{comp}$ and $\cong_{comp}$ and therefore drop the index and simply write $\sqsubseteq$ for dominance and $\cong$ for equivalence.

In the literature several other notions of schema dominance and equivalence have been introduced. Hull in [9] introduces four different notions on the basis of the relational data model. In the case of *calculus dominance* $\sqsubseteq_{calc}$ the mappings f and g must be defined by safe relational calculus or equivalently the relational algebra. In the case of *generic dominance* $\sqsubseteq_{gen}$ the mappings f and g must be generic in the sense that they commute with permutations of domain values fixing only a finite set Z of values. In the case of *internal dominance* $\sqsubseteq_{int}$ the mappings f and g may only introduce a finite set of new values. Finally, Hull also defines absolute dominance.

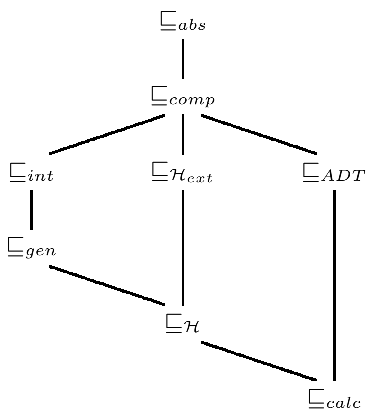


Fig. 2. Relationship between schema dominance notions

All these notions can be generalised to HERM schemata. For instance, the work in [10] uses the same absolute dominance relation as [9] without referring explicitly to the relational model. In [9] it has been shown that calculus dominance implies generic dominance, which itself implies internal dominance, which implies absolute dominance. All these implications are strict. In [15] it has been shown that calculus dominance implies HERM dominance $\sqsubseteq_{\mathcal{H}}$, which implies generic dominance. Of course, both in internal dominance and extended HERM dominance we have to deal with computable queries, so both imply computable dominance.

ADT dominance as defined by Qian [19] is based on order-sorted signatures and algebras. A schema transformation, i.e. our f, must be defined as a signature interpretation. It has been shown in [19] that calculus dominance implies ADT dominance, which implies absolute dominance. These implications are strict. Furthermore, ADT dominance is incomparable with the other notions of dominance as defined by Hull. In [15] it has been shown that ADT dominance and HERM dominance are also incomparable. However, the transformations defined by Qian define computable queries, so ADT dominance implies computable dominance.

Figure 2 illustrates the relationship between the various notions of schema dominance. We base our work on computable dominance, because absolute dominance is too general. It does not really preserve the semantics of the original schemata. On the other hand, extended HERM dominance, internal dominance and ADT dominance are pairwise incomparable, but all three notions make perfect sense in that the examples that can be expressed in them (but maybe not in the other formalisms) are reasonable. Therefore, computable dominance has been chosen, because it covers all reasonable notions of dominance without running into the problems arising from absolute dominance. However, computable dominance still leaves a lot of latitude for schema integration, and in most practical cases this latitude will not be exploited, i.e. a weaker notion of dominance would be sufficient.

4 Schema and View Integration and Cooperation in Databases

In this section we address first schema integration in databases following [16]. Without loss of generality we assume that we are given only two HERM schemata $(\mathcal{S}_1, \Sigma_1)$ and $(\mathcal{S}_2, \Sigma_2)$. Before starting the integration process we assume that these schemata have been "cleaned", i.e. we assume that all name clashes have been removed by renaming homonymous attributes. Ideally, we may assume that names used in both schemata are different.

4.1 Schema and View Integration Process

We first describe a pragmatic method for schema integration, and then explain how this method applies to view integration. The details are then filled by transformation rules described in the following subsections.

1. The first step is the homogenisation of the schemata. This includes the restructuring of the schemata turning attributes into entity types, entity types into relationship types and vice versa. Furthermore, we add attributes and shift attributes along hierarchies and paths. All these individual paces correspond to the application of transformation rules. The result of the homogenisation step are schemata $(\mathcal{S}'_1, \Sigma'_1)$ and $(\mathcal{S}'_2, \Sigma'_2)$.
2. The second step consists in adding inter-schema integrity constraints that describe the semantic relationships between the two schemata. Formally, we obtain another set of constraints Σ_0, and thus the result of this step is a single HERM schema $(\mathcal{S}'_1 \cup \mathcal{S}'_2, \Sigma'_1 \cup \Sigma'_2 \cup \Sigma_0)$.
3. The third step is only a preparation of the following steps. Due to the expected large size of the schemata, these are divided into modules, each of which describes a subschema. Corresponding modules are identified in order to approach the integration of modules first. If schemata are of moderate size, this step can be omitted.
4. Step four considers the integration of types on level 0, 1, etc., i.e. we start with entity types and level-0-clusters, then proceed with relationship types and clusters on level 1, then relationship types and clusters on level 2, etc. For each level we integrate corresponding types or clusters with respect to equality, containment, overlap and disjointness conditions. Note that this step is similar to the work done in [13,14,29].
5. The fifth step deals with the integration of paths using path inclusion dependencies.
6. Finally, we consider remaining integrity constraints such as (path) functional dependencies and join dependencies.

Let us now see how this method deals with view integration. First recall that a view is nothing but a stored query.

Definition 4.1. A *view* V on a HERM schema $(\mathcal{S}, \Sigma)$ consists of a schema $\mathcal{S}_V$ and a query q_V with a query mapping $inst(\mathcal{S}, \Sigma) \to inst(\mathcal{S}_V)$.

Example 4.1. Let $\mathcal{S}_1$ be the HERM schema from Example 3.1. Let us define a view $V_1 = \text{Customer\_Credibility}$ on this schema, in which the target schema $\mathcal{S}_{V_1}$ consists of a single entity-type

$$\begin{aligned}\text{Customer\_Cr} = (\emptyset, \{&\text{customer\_no}, \text{name}, \text{date\_of\_birth},\\ &\text{inc\_oblig}\{(\text{type}, \text{amount}, \text{frequency})\},\\ &\text{loans}\{(\text{type}, \text{amount}, \text{account\_balance})\}\},\\ \{&\text{customer\_no}\})\end{aligned}$$

What we want to obtain is the set of all customers with their customer number, name and date of birth, plus their set of incomes and obligations (each described by their type, amount and frequency of payment), plus their set of loans (each described by their type, amount and corresponding account balance).

The defining query for this view is a bit tricky, as it has to involve the construction of two set-valued attributes. Following [25] we could employ a generalised nest-operation or use an IQL-like logic program, which would look as follows:

$$C(i_c, c, n, b, O, L) \leftarrow \text{Customer}(i_c, (c, n, a, b));$$

$$\begin{aligned}\hat{L}(\text{“mortgage”}, a, c) \leftarrow &\text{Mortgage}(i_m, (i_{lt}, m, a, d, p, b, e, o)),\\ &\text{Loan}(i_l, (\text{mortgage} : i_m)),\\ &\text{Account}(i_a, (i_l, n, c),\\ &\text{Owes}(i_o, (i_c, i_l, b', e'),\\ &C(i_c, cc, nc, bc, O, L).\end{aligned}$$

$$\begin{aligned}\hat{L}(\text{“personal”}, a, c) \leftarrow &\text{Personal\_Loan}(i_{pl}, (i_{lt}, l, a, p, b, e, t)),\\ &\text{Loan}(i_l, (\text{personal} : i_{pl})),\\ &\text{Account}(i_a, (i_l, n, c),\\ &\text{Owes}(i_o, (i_c, i_l, b', e'),\\ &C(i_c, cc, nc, bc, O, L).\\ \hat{O}(\text{“income”}, a, f) \leftarrow &C(i_c, cc, nc, bc, O, L),\\ &\text{Income}(i_i, (i_c, c, a, t, f)).\\ \hat{O}(\text{“obligation”}, a, f) \leftarrow &C(i_c, cc, nc, bc, O, L),\\ &\text{Obligation}(i_o, (i_c, c, a, t, f));\end{aligned}$$

$$\text{Customer\_Cr}(i, (c, n, b, \hat{O}, \hat{L})) \leftarrow C(i_c, c, n, b, O, L).$$

In a nutshell – without going into formal details – this logic program is a sequence of clause sets, separated by a semicolon. These clause sets are evaluated sequentially. Each clause set itself is evaluated by computing an inflationary fixed point, i.e. for each clause we bind the variables on the right hand side using the given instance of the database, and add the corresponding value for the left hand side to the database. The interesting point is that the variables that only occur on the left hand side will be bound to new identifiers, and for each such variable X we obtain a new predicate $\hat{X}$ so that we can compute a set corresponding to each identifier.

In our special case here, we first construct a set C of customers with their identifier i_c, customer number c, name n, date of birth b and identifiers O and L representing the set of obligations and loans, respectively. In the second clause set we construct the corresponding sets $\hat{O}$ and $\hat{L}$ of obligations and loans, respectively, for each customer in C. In the final clause set we replace the identifiers O and L by the corresponding sets $\hat{O}$ and $\hat{L}$, respectively, and create a new identifier i for each element of the result.

So the view integration problem starts with two views V_1 and V_2 on the same HERM schema $(\mathcal{S}, \Sigma)$, and should result in a new integrated view V such that $\mathcal{S}_V$ results from integration of the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$, and for each instance db over $(\mathcal{S}, \Sigma)$ the two query results $q_{V_1}(db)$ and $q_{V_2}(db)$ together are equivalent to $q_V(db)$.

Now, if the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$ are "cleaned", we may combine the queries q_{V_1} and q_{V_2} into one yielding a query mapping $inst(\mathcal{S}, \Sigma) \rightarrow inst(\mathcal{S}_{V_1} \cup \mathcal{S}_{V_2})$ defined by the query $q_{V_1} \cup q_{V_2}$. If we simply integrate the schemata $\mathcal{S}_{V_1}$ and $\mathcal{S}_{V_2}$ into $\mathcal{S}_V$ according to the method described above, we obtain an induced mapping $f : inst(\mathcal{S}_{V_1} \cup \mathcal{S}_{V_2}) \rightarrow inst(\mathcal{S}_V)$. As we deal with computable queries, f is the query mapping of some computable query q_f. Taking $q_V = q_f \circ (q_{V_1} \cup q_{V_2})$, V becomes a view over $(\mathcal{S}, \Sigma)$ with schema $\mathcal{S}_V$ and defining query q_V.

This approach to view integration also works in the more general situation, where the given views V_1 and V_2 are defined over different schemata $(\mathcal{S}_1, \Sigma_2)$ and $(\mathcal{S}_2, \Sigma_2)$, respectively.

4.2 Transformation Rules

In the following subsections we describe transformation rules in detail. All rules will be presented in the same way, i.e. we assume a given HERM schema $(\mathcal{S}, \Sigma)$, but we only indicate parts of it. The resulting schema will be $(\mathcal{S}_{new}, \Sigma_{new})$. The new types in the new schema will be marked with a subscript $_{new}$. With these conventions the rules will be self-explanatory.

All the rules in this subsection subsume the rules that have been stated implicitly or explicitly in former work by others. In addition, the rules will establish computational equivalence in all cases, but formal proofs – which are not too hard – have been omitted. So what we can expect from our approach is soundness, but not completeness. The non-completeness results from the fact that computational dominance would allow us to define more general rules. It is very unlikely that there will exist a complete set of rules. On the other hand, having used the notion of computational dominance permits openness, i.e. the set of rules can be extended, if such a need arises.

Schema Restructuring. The first group of rules addresses the aspect of schema restructuring which will be used in the homogenisation step 1 of our method.

Rule 1. Replace a tuple attribute $X(A_1, \ldots, A_m)$ in an entity or relationship type R by the attributes $A_1, \ldots, A_m$. The resulting type R_{new} will replace R. For $X(A'_1, \ldots, A'_n) \in \mathit{key}(R)$ with $A_i \leq A'_i$ we obtain $A'_1, \ldots, A'_n \in \mathit{key}(R')$.

This rule includes the simple case, where R is an entity type, which could be treated as a separate rule.

Rule 2. Replace a component $r : R'$ in a relationship type R by lower level components and attributes. Let the new type be R_{new}. For $\text{comp}(R') = \{r_1 : R_1, \ldots, r_n : R_n\}$ we get $\text{comp}(R_{new}) = \text{comp}(R) - \{r : R'\} \cup \{r_1^{(r)} : R_1, \ldots, r_n^{(r)} : R_n\}$ with new role names $r_i^{(r)}$ composed from r_i and r and $\text{attr}(R_{new}) = \text{attr}(R) \cup \text{attr}(R')$. In the case $r : R' \in \text{key}(R)$ and $\text{key}(R') = \{r_{i_1} : R_{i_1}, \ldots, r_{i_k} : R_{i_k}, A_1, \ldots, A_m\}$ we obtain $\text{key}(R_{new}) = \text{key}(R) - \{r : R'\} \cup \{r_{i_1}^{(r)} : R_{i_1}, \ldots, r_{i_k}^{(r)} : R_{i_k}, A_1, \ldots, A_m\}$, otherwise we have $\text{key}(R_{new}) = \text{key}(R)$.

It is easy to see how to simplify this rule in the case, where R' is an entity type. Again, this could by formulated by two separate rules.

Rule 3. Replace a cluster $C = C_1 \oplus \cdots \oplus C_n$ with a cluster component $C_i = C_{i_1} \oplus \cdots \oplus C_{i_m}$ by a new cluster $C = C_1 \oplus \cdots \oplus C_{i-1} \oplus C_{i_1} \oplus \cdots \oplus C_{i_m} \oplus C_{i+1} \oplus \cdots \oplus C_n$.

Rule 4. Replace a relationship type R with a cluster component $r : C$ ($C = C_1 \oplus \cdots \oplus C_n$) by a new cluster $C_{new} = R_{1,new} \oplus \cdots \oplus R_{n,new}$ and new relationship types $R_{i,new}$ with $\mathit{comp}(R_{i,new}) = \mathit{comp}(R) - \{r : C\} \cup \{r_i : C_i\}$ and $\mathit{attr}(R_{i,new}) = \mathit{attr}(R)$. For $r : C \in \mathit{key}(R)$ we obtain $\mathit{key} = \mathit{key}(R) - \{r : C\} \cup \{r : C_i\}$, otherwise take $\mathit{key} = \mathit{key}(R)$.

In the case of the restructuring rules 1 – 4 we can always show that the original schema and the resulting schema are equivalent. The next rule only guarantees that the resulting new schema dominates the old one.

Rule 5. Replace a key-based inclusion dependency $R'[\mathit{key}(R_i)] \subseteq R[\mathit{key}(R)]$ by new relationship types R'_{new} with $\mathit{comp}(R'_{new}) = \{r' : R', r : R\} = \mathit{key}(R'_{new})$ and $\mathit{attr}(R'_{new}) = \emptyset$ together with participation cardinality constraints
$\mathit{card}(R'_{new}, R) = (0, 1)$ and $\mathit{card}(R'_{new}, R') = (1, 1)$.

The last two restructuring rules allow to switch between attributes and entity types and between entity and relationship types. These rules 6 and 7 guarantee schema equivalence.

Rule 6. Replace an entity type E with $A \in \mathit{attr}(E)$ by E_{new} such that $\mathit{attr}(E_{new}) = \mathit{attr}(E) - \{A\}$ holds. Furthermore, introduce an entity type E'_{new} with $\mathit{attr}(E'_{new}) = \{A\} = \mathit{key}(E'_{new})$ and a new relationship type R_{new} with $\mathit{comp}(R_{new}) = \{r_{new} : E_{new}, r'_{new} : E'_{new}\} = \mathit{key}(R_{new})$ and $\mathit{attr}(R_{new}) = \emptyset$. Add the cardinality constraints $\mathit{card}(R_{new}, E_{new}) = (1, 1)$ and $\mathit{card}(R_{new}, E'_{new}) = (1, \infty)$.

Rule 7. Replace a relationship type R with $comp(R) = \{r_1 : R_1, \dots, r_n : R_n\}$ and the cardinality constraints $card(R, R_i) = (x_i, y_i)$ by a new entity type E_{new} with $attr(E_{new}) = attr(R) = key(E_{new})$ and n new relationship types $R_{i,new}$ with $comp(R_{i,new}) = \{r_i : R_i, r : E_{new}\} = key(R_{i,new})$ and $attr(R_{i,new}) = \emptyset$. Replace the cardinality constraints by
$card(R_{i,new}, R_i) = (1, y_i)$ and $card(R_{i,new}, E_{new}) = (1, \infty)$.

In the case of rule 7 explicit knowledge of the key of R allows to sharpen the cardinality constraints.

Shifting Attributes. The second group of rules deals with the shifting of attributes. This will also be used in the homogenisation step 1 of our method. Rule 8 allows to shift a synonymous attribute occurring in two subtypes, i.e. whenever tuples agree on the key they also agree on that attribute, to be shifted to a supertype. This rule leads to a dominating schema. Conversely, rule 9 allows to shift an attribute from a supertype to subtypes, in which case schema equivalence can be verified.

Rule 8. For $comp(R_i) = \{r_i : R\}$ and $A_i \in attr(R_i) - key(R_i)$ $(i = 1, 2)$ together with the constraint $\forall t, t'.t[key(R_1)] = t'[key(R_2)] \Rightarrow t[A_1] = t'[A_2]$ replace the types R, R_1 and R_2 such that $attr(R_{new}) = attr(R) \cup \{A_i\}$, $comp(R_{i,new}) = \{r_i : R_{new}\}$ and $attr(R_{i,new}) = attr(R_i) - \{A_i\}$ hold.

Rule 9. For $comp(R_i) = \{r_i : R\}$ $(i = 1, \dots, n)$ and $A \in attr(R) - key(R)$ together with the constraint $\forall t \in R. \exists t' \in R_i. t'[r_i] = t$ replace the types such that $attr(R_{new}) = attr(R) - \{A\}$, $comp(R_{i,new}) = \{r_i : R_{new}\}$ and $attr(R_{i,new}) = attr(R_i) \cup \{A\}$ hold.

The next two rules 10 and 11 concern the reorganisation of paths and the shifting of attributes along paths. In both cases we obtain a dominating schema. Rule 10 could be split into two rules dealing separately with binary and unary relationship types R_n.

Rule 10. For a path $P \equiv R_1 - \cdots - R_n$ and a relationship type R with $r_n : R_n \in comp(R)$ together with path cardinality constraints $card(P, R_1) \leq (1, 1) \leq card(P, R_n)$ replace R such that $comp(R_{new}) = comp(R) - \{r_n : R_n\} \cup \{r_{1,new} : R_1\}$ with a new role $r_{1,new}$ holds.

Rule 11. For a path $P \equiv R_1 - \cdots - R_n$ with $A \in attr(R_n)$ and path cardinality constraints $card(P, R_1) \leq (1, 1) \leq card(P, R_n)$ replace R_1, R_n such that $attr(R_{1,new}) = attr(R_1) \cup \{A\}$ and $attr(R_{n,new}) = attr(R_n) - \{A\}$ hold.

Schema Extension. The third group of rules deal with the schema extensions. This either concerns new attributes, new subtypes or the simplification of hierarchies. These rules are needed in step 1 of our method.

Rule 12. Add a new attribute A to the type R, i.e. $attr(R_{new}) = attr(R) \cup \{A\}$. In addition, the new attribute may be used to extend the key, i.e. we may have $key(R_{new}) = key(R) \cup \{A\}$.

If the new attribute A introduced by rule 12 does not become a key attribute, we obtain a dominating schema.

The next two rules allow to introduce a new subtype via selection or projection on non-key-attributes. In both cases we have schema equivalence.

Rule 13. For a type R introduce a new relationship type R'_{new} with $comp(R'_{new}) = \{r : R\} = key(R'_{new})$ and add a constraint $R'_{new} = \sigma_\varphi(R)$ for some selection formula φ.

Rule 14. For a type R and attributes $A_1, \dots, A_n \in attr(R)$ such that there are no $B_i \in key(R)$ with $A_i \geq B_i$ introduce a new relationship type R'_{new} with $comp(R'_{new}) = \{r : R\} = key(R'_{new})$ and $attr(R'_{new}) = \{A_1, \dots, A_n\}$, and add a constraint $R'_{new} = \pi_{A_1,\dots,A_n}(R)$.

The last rule 15 in this group allows to simplify hierarchies. Here again we must exploit $\mathcal{H}_{ext}$ to obtain dominance.

Rule 15. Replace types R, $R_1, \dots, R_n$ with $comp(R_i) = \{r_i : R\} = key(R_i)$ and $card(R, R_i) = (0, 1)$ $(i = 1, \dots, n)$ by a new type R_{new} with $comp(R_{new}) = comp(R)$, $attr(R_{new}) = attr(R) \cup \bigcup_{i=1}^{n} attr(R_i)$ and $key(R_{new}) = key(R)$.

Type Integration. The fourth group of rules deals with the integration of types in step 4 of our method. Rule 16 considers the equality case, rule 17 considers the containment case, and rule 18 covers the overlap case. Note that these transformation rules cover the core of the approaches in [13,29,14].

Rule 16. If R_1 and R_2 are types with $key(R_1) = key(R_2)$ and we have the constraint $R_1[key(R_1) \cup X] = f(R_2[key(R_2) \cup Y])$ for some $X \subseteq comp(R_1) \cup attr(R_1)$, $Y \subseteq comp(R_2) \cup attr(R_2)$ and a bijective mapping f, then replace these types by R_{new} with $comp(R_{new}) = comp(R_1) \cup (comp(R_2) - Y - key(R_2))$, $attr(R_{new}) = attr(R_1) \cup (attr(R_2) - Y - key(R_2)) \cup \{D\}$ and $key(R_{new}) = key(R_1) \cup \{D\}$ and an optional new distinguishing attribute D.

Rule 17. If R_1 and R_2 are types with $key(R_1) = key(R_2)$ and the constraint $R_2[key(R_2) \cup Y] \subseteq f(R_1[key(R_1) \cup X]$ holds for some $X \subseteq comp(R_1) \cup attr(R_1)$, $Y \subseteq comp(R_2) \cup attr(R_2)$ and a bijective mapping f, then replace R_1 by $R_{1,new}$ with $comp(R_{1,new}) = comp(R_1)$, $attr(R_{new}) = attr(R_1) \cup \{D\}$ and $key(R_{new}) = key(R_1) \cup \{D\}$ and an optional new distinguishing attribute D. Furthermore, replace R_2 by $R_{2,new}$ with $comp(R_{2,new})\{r_{new} : R_{1,new}\} \cup comp(R_2) - Y - key(R_2)$, $attr(R_{2,new}) = attr(R_2) - Y - key(R_2)$ and $key(R_{2,new}) = \{r_{new} : R_{1,new}\}$.

Rule 18. If R_1 and R_2 are types with $key(R_1) = key(R_2)$ such that for $X \subseteq comp(R_1) \cup attr(R_1)$, $Y \subseteq comp(R_2) \cup attr(R_2)$ and a bijective mapping f the constraints

$$R_2[key(R_2) \cup Y] \subseteq f(R_1[key(R_1) \cup X] \quad ,$$
$$R_2[key(R_2) \cup Y] \supseteq f(R_1[key(R_1) \cup X] \quad \text{and}$$
$$R_2[key(R_2) \cup Y] \cap f(R_1[key(R_1) \cup X] = \emptyset$$

are not satisfied, then replace R_1 by $R_{1,new}$ with $comp(R_{1,new})\{r_{1,new} : R_{new}\} \cup comp(R_1) - X - key(R_1)$, $attr(R_{1,new}) = attr(R_1) - X - key(R_1)$ and $key(R_{1,new}) = \{r_{1,new} : R_{new}\}$, replace R_2 by $R_{2,new}$ with $comp(R_{2,new})\{r_{new} : R_{1,new}\} \cup comp(R_2) - Y - key(R_2)$, $attr(R_{2,new}) = attr(R_2) - Y - key(R_2)$ and $key(R_{2,new}) = \{r_{new} : R_{1,new}\}$ and introduce a new type R_{new} with $comp(R_{new}) = comp(R_1) \cup comp(R_2)$, $attr(R_{new}) = attr(R_1) \cup attr(R_2) \cup \{D\}$ and $key(R_{new}) = key(R_1) \cup \{D\}$ and an optional new distinguishing attribute D.

The rules 16-18 could each be split into several rules depending on f being the identity or not and the necessity to introduce D or not. In all cases we obtain dominance.

Rule 19 considers the case of a selection condition, in which case schema equivalence holds.

Rule 19. If R and R' are types with $comp(R') \cup attr(R') = Z \subseteq comp(R) \cup attr(R)$ such that the constraint $R' = \sigma_\varphi(\pi_Z(R))$ holds for some selection condition φ, then omit R'.

Handling Integrity Constraints. The fifth group of rules to be applied in step 5 of our method concerns transformations originating from path inclusion constraints. Rule 20 allows us to change a relationship type. This rule leads to equivalent schemata. Rule 21 allows to introduce a relationship type and a join dependency. Finally, rule 22 handles a condition under which a relationship type may be omitted. Both rules 21 and 22 guarantee dominance.

Rule 20. If there are paths $P \equiv R_1 - R - R_2$ and $P' \equiv R_2 - R' - R_3$ with $comp(R) = \{r_1 : R_1, r_2 : R_2\}$ and $comp(R') = \{r_3 : R_3, r_2' : R_2\}$ such that the constraint $P[R_2] \subseteq P'[R_2]$ holds, then replace R in such a way that $comp(R_{new}) = \{r_1 : R_1, r_{new} : R'\}$, $attr(R_{new}) = attr(R)$ and $key(R_{new}) = key(R) - \{r_2 : R_2\} \cup \{r_{new} : R'\}$ hold.

Rule 21. If there are paths $P \equiv R_1 - R - R_2$ and $P' \equiv R_2 - R' - R_3$ with $comp(R) = \{r_1 : R_1, r_2 : R_2\}$ and $comp(R') = \{r_3 : R_3, r_2' : R_2\}$ such that the constraint $P[R_2] = P'[R_2]$ holds, then replace R and R' by R_{new} such that $comp(R_{new}) = \{r_1 : R_1, r_{2,new} : R_2, r_3 : R_3\}$, $attr(R_{new}) = attr(R) \cup attr(R')$ and $key(R_{new}) = (key(R) - \{r_2 : R_2\}) \cup (key(R') - \{r_2' : R_2\}) \cup \{r_{2,new} : R_2\}$ hold. Add the join dependency

$R_{new}[r_1, r_{2,new}] \bowtie R_{new}[r_{2,new}, r_3] \subseteq R_{new}[r_1, r_{2,new}, r_3]$.

Rule 22. If there are paths $P \equiv R_1 - R_2 - \cdots - R_n$ and $P' \equiv R_1 - R - R_n$ with $comp(R) = \{r_1 : R_1, r_n : R_n\}$ such that the constraint $P[R_1, R_n] = P'[R_1, R_n]$ holds, then omit R.

The final group of transformation rules 23-26 permits to handle remaining constraints such as functional dependencies, path functional dependencies, and join dependencies. All these constraints are described in detail in [30]. The rules refer to step 6 of our method.

Rule 23 handles vertical decomposition in the presence of a functional dependency. Rule 24 allows to simplify a key in the presence of a path functional dependency. Rule 25 introduces a new entity type in the presence of a path functional dependency. Finally, rule 26 replaces a multi-ary relationship type by binary relationship types in the presence of a join dependency. The four rules lead to dominating schemata.

Rule 23. If a functional dependency $X \rightarrow A$ with a generalised subset X of $attr(E)$ and an attribute $A \in attr(E) - X$ holds on an entity type E, but $X \rightarrow key(E)$ does not hold, then remove A from $attr(E)$ and add a new entity type E'_{new} with $attr(E'_{new}) = X \cup \{A\}$ and $key(E'_{new}) = X$.

Rule 24. For a path $P \equiv R_1 - R - R_2$ with $comp(R) = \{r_1 : R_1, r_2 : R_2\}$ such that the path functional dependency $X \rightarrow key(R_2)$ holds for a generalised subset X of $attr(R_1)$ replace $key(R)$ by $\{r_1 : R_1\}$.

Rule 25. For a path $P \equiv R_1 - \cdots - R_n$ such that the path functional dependency $X \rightarrow A$ holds for a generalised subset X of $attr(R_1)$ and $A \in attr(R_n)$ add a new entity type E_{new} with $attr(E_{new}) = X \cup \{A\}$ and $key(E_{new}) = X$.

Rule 26. If R is an n-ary relationship type with $comp(R) = \{r_1 : R_1, \dots, r_n : R_n\}$ and $attr(R) = \emptyset$ such that the join dependency $R[r_1, r_2] \bowtie \dots \bowtie R[r_1, r_n] \subseteq R[r_1, \dots, r_n]$ holds, then replace R by n new relationship types $R_{1,new}, \dots, R_{n,new}$ with $comp(R_{i,new}) = \{r_1 : R_1, r_i : R_i\} = key(R_{i,new})$ and $attr(R_{i,new}) = \emptyset$.

4.3 View Cooperation

View cooperation [30] provides an alternative to view integration in which the integrated view is only virtual. That is the constituting views are kept and exchange functions are designed to provide the same functionality as if the views were integrated.

Definition 4.2. Let $V_i = (\mathcal{S}_{V_i}, q_{V_i})$ $(i = 1, 2)$ be views (on the same or different HERM schemata). V_1 *cooperates* with V_2 iff there are subschemata $\mathcal{S}'_{V_i}$ of $\mathcal{S}_{V_i}$ and functions $f_1 : inst(\mathcal{S}'_{V_1}) \rightarrow inst(\mathcal{S}'_{V_2})$ and $f_2 : inst(\mathcal{S}'_{V_2}) \rightarrow inst(\mathcal{S}'_{V_1})$, such that both $f_1 \circ f_2$ and $f_2 \circ f_1$ are the identity function.

Basically, view cooperation expresses that part of view V_1, exactly the one corresponding to the subschema $\mathcal{S}'_{V_1}$ can be expressed by the part of view V_2 corresponding to the subschema $\mathcal{S}'_{V_2}$.

Now, if we want to obtain a cooperation between given views V_1 and V_2, we may simply apply our view integration method to them using the same transformation rules. This will result in an integrated view $V = (\mathcal{S}_V, q_V)$. With respect to this integrated view both $\mathcal{S}'_{V_1}$ and $\mathcal{S}'_{V_2}$ will be identified with a subschema $\mathcal{S}'_V$. In particular we obtain functions $f'_i : inst(\mathcal{S}'_{V_i}) \to inst(\mathcal{S}'_V)$ and $g'_i : inst(\mathcal{S}'_V) \to inst(\mathcal{S}'_{V_i}))$ $(i = 1, 2)$ with $g'_i \circ f'_i = id$ and $f'_i \circ g'_i = id$. Thus, $f_1 = g'_2 \circ f'_1$ and $f_2 = g'_1 \circ f'_2$ define the view cooperation functions.

4.4 Case Study

In order to demonstrate our approach to view integration and cooperation we first use a second HERM schema $\mathcal{S}_2$, which is given by the HERM diagram in Figure 3. We may think of this schema being used by a mortgage broker who deals with mortgages from different banks. In addition, the assessment of the credibility of a customer may depend on securities owned by supporting relatives, so we add information about relatives to the schema. We omit the formal definition for this HERM schema.

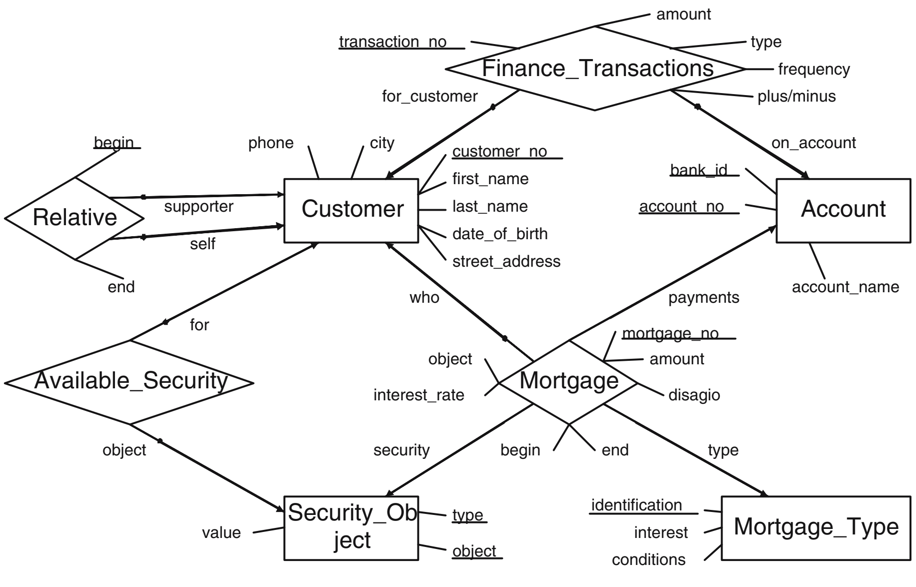


Fig. 3. HERM diagram for mortgage broker application

Example 4.2. Let us first address the integration of the HERM schemata $\mathcal{S}_1$ and $\mathcal{S}_2$ from Figures 1 and 3. In these schemata we mainly have to deal with the homogenisation of types in the two schemata, i.e. with type restructuring, and with type integration.

First we homogenise CUSTOMER appearing in both schemata adding attributes (rule 12) and turning the attributes name and address into tuple attributes with components first_name and last_name, and city and street_address, respectively – which means to apply rule 1. We obtain the integrated type

Customer = (∅, { customer_no, name(first_name, last_name), address(city, street_address), phone, date_of_birth }, { customer_no })

Next we split Finance_Transactions in $\mathcal{S}_2$ into Income and Obligation according to the value of the attribute plus/minus. This results from applying rule 13. Furthermore, we can rename role names and turn the attribute account for both Income and Obligation into a component on_account : Account.

The homogenisation of Mortgage_Type in $\mathcal{S}_1$ and Loan_Type in $\mathcal{S}_2$ reqires not much more than the renaming of the attribute type to identification. The homogenisation of Mortgage in both schemata requires first to introduce in $\mathcal{S}_2$ a new type Payment, which takes the components who : Customer and payments : Account from Mortgage, and add a component for : Mortgage. Similarly, we can split Security in $\mathcal{S}_1$ into Available_Security, for which we can drop the component for : Mortgage, and Security. In $\mathcal{S}_1$ we can rename Owes to Payment and add a component payments: Account. Finally, we replace Mortgage in Payment by the cluster Loan.

The HERM diagram of the final resulting schema is shown in Figure 4.

The work in [16] contains further examples for schema integration.

The integration of schemata turns each view over one of the source schemata into a view over the integrated schema. What is changed is the defining query.

Example 4.3. Consider the view V_1 from Example 4.1. If schema $\mathcal{S}_1$ is integrated with schema $\mathcal{S}_2$ into schema $\mathcal{S}_{12}$ in Figure 4, the defining query for V_1 changes as follows:

$$\begin{aligned}
C(i_c, c, n, b, O, L) \leftarrow & \textsc{Customer}(i_c, (c, n, a, b, ph)); \\
\hat{L}(\text{“mortgage”}, a, c) \leftarrow & \textsc{Mortgage}(i_m, (i_{lt}, m, a, d, p, b, e, o)), \\
& \textsc{Loan}(i_l, (\text{mortgage} : i_m)), \\
& \textsc{Payment}(i_p, (i_a, i_l, i_c), \\
& C(i_c, cc, nc, bc, O, L). \\
\hat{L}(\text{“personal”}, a, c) \leftarrow & \textsc{Personal\_Loan}(i_{pl}, (i_{lt}, l, a, p, b, e, t)), \\
& \textsc{Loan}(i_l, (\text{personal} : i_{pl})), \\
& \textsc{Payment}(i_p, (i_a, i_l, i_c), \\
& C(i_c, cc, nc, bc, O, L). \\
\hat{O}(\text{“income”}, a, f) \leftarrow & C(i_c, cc, nc, bc, O, L), \\
& \textsc{Income}(i_i, (i_c, i_a, t, t', a, f)). \\
\hat{O}(\text{“obligation”}, a, f) \leftarrow & C(i_c, cc, nc, bc, O, L), \\
& \textsc{Obligation}(i_i, (i_c, i_a, t, t', a, f));
\end{aligned}$$

$$\textsc{Customer\_Cr}(i, (c, n, b, \hat{O}, \hat{L})) \leftarrow C(i_c, c, n, b, O, L).$$

Example 4.4. Let us now address the integration of views. For this consider another view V_2 defined on the database schema $\mathcal{S}_2$. Let the target schema $\mathcal{S}_{V_2}$ consist again of a single entity-type

$$\begin{array}{l}\textsc{Customer\_Cr}{=}(\emptyset, \{\text{customer\_no}, \text{name}(\text{first\_name},\text{last\_name}), \text{date\_of\_birth}, \\ \qquad\qquad \text{inc\_oblig}\{(\text{type}, \text{amount}, \text{frequency})\}, \\ \qquad\qquad \text{securities}\{(\text{type}, \text{object}, \text{value}, \text{self})\}\}, \\ \qquad \{\text{customer\_no}\})\end{array}$$

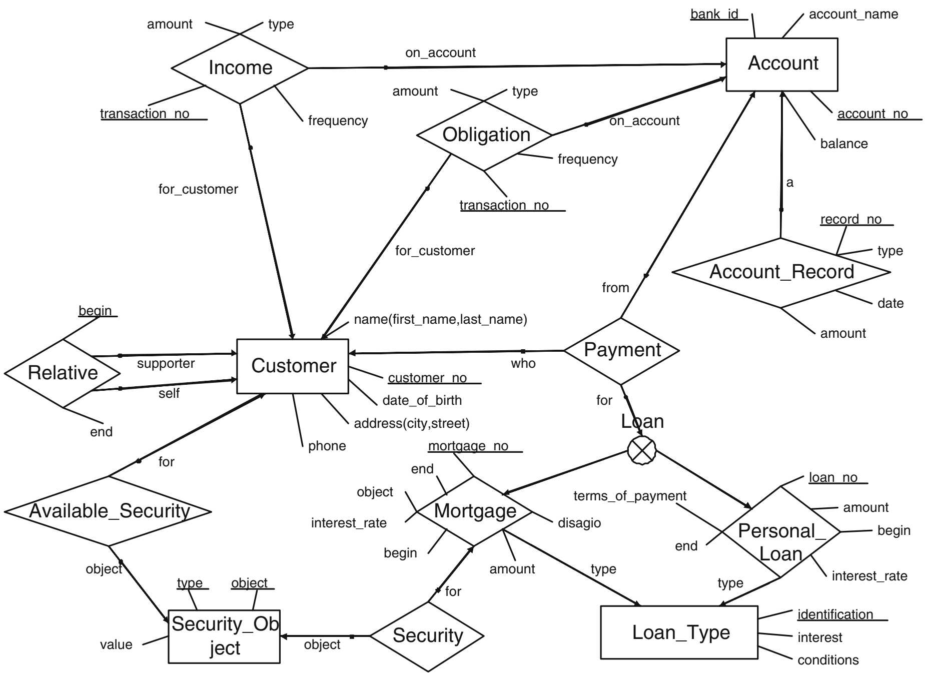


Fig. 4. HERM diagram for integrated HERM schemata

Analogous to Example 4.1 we want to obtain the set of all customers with their customer number, name and date of birth, plus their set of incomes and obligations (each described by their type, amount and frequency of payment), plus their set of securities (each described by their type, object, value and whether it is a security owned by the customer or a supporting relative). The defining query will be built analogously to the one in Example 4.1 – we omit the details. We also omit how this defining query will be translated into a query on the integrated schema $\mathcal{S}_{12}$.

Then we basically integrate the target schemata of V_1 and V_2, which results in a schema $\mathcal{S}_{V_{12}}$ with a single entity-type

$$\begin{array}{l}\textsc{Customer\_Cr}{=}(\emptyset, \{\text{customer\_no}, \text{name}(\text{first\_name},\text{last\_name}), \text{date\_of\_birth}, \\ \qquad\qquad \text{inc\_oblig}\{(\text{type}, \text{amount}, \text{frequency})\}, \\ \qquad\qquad \text{securities}\{(\text{type}, \text{object}, \text{value}, \text{self})\},\end{array}$$

$$\begin{aligned} &\text{loans}\{(\text{type}, \text{amount}, \text{account\_balance})\}\}, \\ &\{\text{customer\_no}\}) \end{aligned}$$

The defining query on $\mathcal{S}_{12}$ is built analogously to the ones for V_1 and V_2. We just obtain a third set-valued attribute that has to be constructed by some fixed-point computation.

Let us finally look at view cooperation, i.e. we do not integrate the views V_1 and V_2 on schema $\mathcal{S}_{12}$, but only determine view cooperation functions.

Example 4.5. The target schemata of the views V_1 and V_2 from Examples 4.1 and 4.4 share an equivalent subschema. Both $\mathcal{S}'_{V_1}$ and $\mathcal{S}'_{V_2}$ consist of a single entity-type

$$\begin{aligned} \textsc{Customer\_Cr} = (\emptyset, \{&\text{customer\_no}, \text{name}, \text{date\_of\_birth}, \\ &\text{inc\_oblig}\{(\text{type}, \text{amount}, \text{frequency})\}\}, \\ \{&\text{customer\_no}\}) \end{aligned}$$

The only difference is that in V_1 the attribute name is a simple attribute, while in V_2 it is a tuple attribute name(first_name, last_name). The corresponding equivalent subschema of $\mathcal{S}_{V_{12}}$ also contains this tuple attribute. Therefore, the cooperation functions $f_1 : \mathit{inst}(\mathcal{S}'_{V_1}) \to \mathit{inst}(\mathcal{S}'_{V_2})$ and $f_2 : \mathit{inst}(\mathcal{S}'_{V_2}) \to \mathit{inst}(\mathcal{S}'_{V_1})$ only affect the name attribute. That is, f_2 will concatenate the first_name and the last_name, while f_1 will decompose name according to some algorithm. For instance, if we assume that spaces are not allowed inside last names, we could simply search for the last space in a name string and take the substring preceding it for first_name, and the string following the space for last_name.

5 View Integration in Data Warehouses

In this section we apply our view integration method to data warehouses. For this we exploit the general idea of data warehouses, i.e. the separation of input from operational databases and output to data marts. This idea is illustrated using a three-tier architecture in Figure 5. In particular, we take up the idea from [17] to model OLAP applications by dialogue types [21].

The underlying idea is simply that the content presented to a user can be described by a value of a complex data type. In addition a user is offered operations – the OLAP operations in the case of data warehouses – that work on the presented value. This defines a *dialogue object*, and dialogue types arise from the usual abstraction. In addition, the values presented to the users depend on some database, so the dialogue types give rise to views that are extended by operations. We briefly illustrate these concepts, then extend our view integration method to this case.

The fact that view integration arises within the area of data warehousing arises from using the dialogue types approach to model the data marts for the OLAP applications. In developing an OLAP application we would first collect the user needs including ways how to work with a system – see [21] for more details. This gives a collection of views, while the warehouse itself is the result of integrating these views. The problems of deciding on materialised views that support the data marts efficiently and chosing view cooperation as an alternative are orthogonal to this integration problem.

5.1 Data Warehouse Architecture

On the bottom tier we have operational databases set up for purposes that are of no particular interest for the data warehouse. However, we assume that the data stored in the data warehouse is extracted from these operational databases. In other words, the input of data into the data warehouse is defined by update- or refresh operations, which take data from these operational databases and insert them into the data warehouse. In particular, these refresh-operations can be formulated by queries on the operational databases. Assuming that the HERM has been used for the operational databases, we can express the refresh-operations in (extended) HERM algebra or calculus. It is not very likely that the structure of a data warehouse will require more complicated (computable) queries.

In general, if there is some conflict between these operational data, i.e. data from different operational databases has to be integrated before it can be inserted into the data warehouse, we need an integrator component [36]. However, we may assume that this integrator is part of the extraction functions.

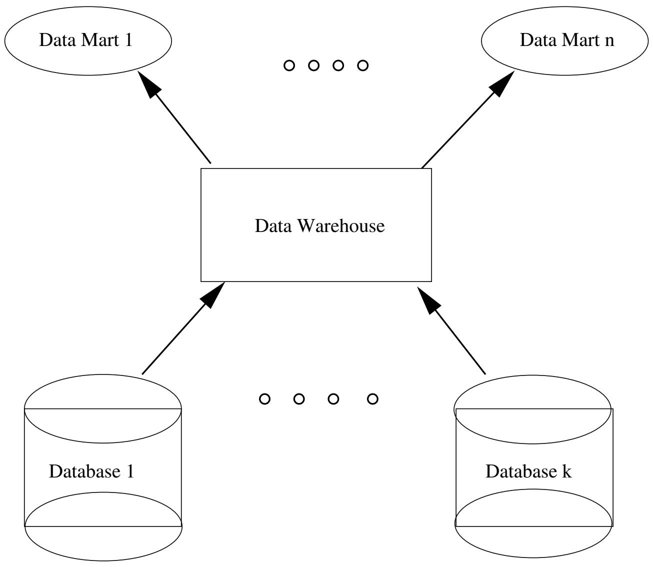


Fig. 5. General Data Warehouse Architecture

The second tier of the architecture in Figure 5 is made up by the data warehouse itself. In principle, we just have a database system here with the only differences that we may assume a simpler schema, i.e. star or snowflake schema, and we do not have to consider complex transactions. The only write-operations are the refresh operations that connect the data warehouse to the operational databases. All other operations only read data from the data warehouse. In fact, we just build views for the "data marts" or dialogue objects that are used as the OLAP interface. So, the view construction operations are the only ones that link the middle tier to the top tier, which deals with OLAP. So again, the major functionality is expressed by views.

The top tier itself is constructed out of the dialogue objects and the OLAP operations working on them. This tier realises the idea of using dialogue objects for this purpose. The general idea from [21] is that each user has a collection of open dialogue objects, i.e. data marts. At any time we may get new users, and each user may create new dialogue objects without closing the existing ones. Thus, we maintain the lists of users and their data marts in the system. Part of the functionality of the OLAP tier deals with adding and removing users and data marts. In particular, if a user leaves the system, all data marts owned by him/her must be removed as well.

The major functionality, however, deals with running operations on existing data marts or creating new data marts. In the latter case we have to use a view creation query as described for the middle tier. In this case we choose a new identifier for this data mart / dialogue object, and initialise its data content. If the user selects some of the data of the data mart and an operation other than quit (i.e. the user does not want to leave the system), close (i.e. the user does not want to finish work on the current data mart) or open (i.e. no new data mart is to be created), then we request to receive additional input from the user, before the selected operation will be executed.

5.2 Dialogue Types

We have seen that the major functionality on the OLAP tier is expressed by views that are extended by operations, which is exactly the idea underlying dialogue types. However, we simplify the original definition from [21] omitting the subtle distinction between hidden and visible parts. We also omit hierarchies of dialogue types.

Definition 5.1. A *dialogue type* D over a HERM schema $(\mathcal{S}, \Sigma)$ consists of a view $V_D = (\mathcal{S}_D, q_D)$, in which $\mathcal{S}_D$ consists of a single entity type E, and a set $\mathcal{O}$ of *dialogue operations*. Each dialogue operation (*d-operation* for short) in $\mathcal{O}$ consists of

– an operation name *op*,
– a list of input parameters $i_1 : D_1, \dots, i_k : D_k$ with domain names D_i,
– an (optional) output domain D_{out},
– a subattribute *sel* of the representing attribute X_E, and

– a d-operation body, which is built from usual programming constructs operating on instances over $(\mathcal{S}, \Sigma)$ and constructs for creating and deleting dialogue objects.

Whenever we are given an instance *db* over $(\mathcal{S}, \Sigma)$, the defining query q_D produces an instance over $\mathcal{S}_D$, i.e. a set of pairs (i, v) with $i \in ID$ and $v \in dom(X_E)$, each of which will be called a *dialogue object of type D*. At any time only a subset of $q_D(db)$ will be available, the *set of active dialogue objects.* These represent the active user dialogues in an abstract way.

The presented value v may be projected to $\pi_{sel}^{X_E}(v)$, which represents the data that must be selected by the user as a prerequisite for executing operation *op*. Once these data are selected and the operation *op* is started, further input for the parameters $i_1, \dots, i_k$ will be requested from the user – using e.g. so-called dialogue boxes [21] – and the execution of *op* will update the database *db* and result in a new set of active dialogue objects.

5.3 Transformation Rules for Dialogue Types

As user dialogues are an invaluable source of information in requirements engineering, we may usually assume that we know about dialogue objects before the defining queries and the underlying database schema is fixed. Therefore, view integration is an unavoidable design task. Integrating the views that underly the dialogue types leads to a design of the data warehouse. In addition, we will always be confronted with the desire to rearrange the "data marts", i.e. the dialogue types that define the OLAP functionality. This problem also appears in database applications other than OLAP.

Therefore, the additional problem is to adapt the d-operations that are used for the functionality presented to a database or data warehouse user. For this we define further transformation rules. However, these additional transformation rules have to be understood as follow-on rules for the case that one of the rules 1-26 is not just applied to schemata or views, but to dialogue types. Thus, we obtain additional changes to the selection attribute *sel* and the body of d-operations.

Rule 27. In case rule 1 is applied to a dialogue type, whenever $X(A_1', \dots, A_k')$ $(k \leq m)$ appears in *sel* or in the body of a d-operation, replace it by $A_1', \dots, A_k'$.

Rule 28. In case rule 2 is applied to a dialogue type, whenever r appears in *sel* or in the body of a d-operation, replace it by $r_1^{(r)}, \dots, r_n^{(r)}$.

We may ignore rules 3 and 4, as clusters have not been allowed in dialogue types. There is also nothing to add for rule 5, as this introduces a new type, so operations have to be defined for that type.

Rule 29. In case rule 6 is applied to a dialogue type omit A in *sel* or in the body of a d-operation, whenever it appears.

Rule 30. In case rule 6 is applied to a dialogue type omit $r_1, \dots, r_n$ in *sel* or in the body of a d-operation, whenever it appears.

These extensions capture the first group of rules dealing with schema restructuring. For the second group of rules dealing with the shifting of attributes we obtain the following extension rules in case the rules are applied to dialogue types.

Rule 31. In case rule 8 is applied to a dialogue type omit A_i in *sel* and the body of operations associated with $R_{i,new}$, whenever it appears in *sel* or the body of an operation associated with R_i.

Rule 32. In case rule 9 is applied to a dialogue type omit A_i in *sel* and the body of operations associated with R_{new}, whenever it appears in *sel* or the body of an operation associated with R.

Note that the last two extension rules have no effect on R_{new} or $R_{i,new}$, as the extension of the selection attribute or the body of an operation has to be defined for these new types.

Rule 33. In case rule 10 is applied to a dialogue type replace r_n by $r_{1,new}$ in *sel* and the body of operations associated with R_{new}.

Rule 34. In case rule 11 is applied to a dialogue type omit A in *sel* and the body of operations associated with $R_{n,new}$.

For the third group of rules, i.e. rules 12-15 dealing with schema extension we cannot define reasonable extension rules for dialogue types, as we always have to deal with completely new types. The same applies to rules 16-19, i.e. the group of rules dealing with type integration.

Finally, for the group of rules dealing with integrity constraints only rules 20, 21 and 23 give rise to the following three extension rules for dialogue types.

Rule 35. In case rule 20 is applied to a dialogue type replace r_2 by r_{new} in *sel* and the body of operations, whenever it appears.

Rule 36. In case rule 21 is applied to a dialogue type replace r_2 by $r_{2,new}$ in *sel* and the body of operations, whenever it appears.

Rule 37. In case rule 23 is applied to a dialogue type remove A in *sel* and the body of operations, whenever it appears.

5.4 Case Study

Let us continue the case study from the previous section. What we have to do is to study how operations associated with a view will be affected by the view integration process.

Example 5.1. Consider the view V_1 on schema $\mathcal{S}_1$ from Example 4.1. Let us associate a d-operation show_security with this view turning it into a dialogue type. The idea behind this operation is that a user who gets a presentation of a customer together with his/her obligations and loans, checks the securities for a mortgage in the list. So the user will select one of the loans, a mortgage, then choose the operation show_security, and receive a presentation of a security list, i.e. a new dialogue object will be opened. That is, the d-operation is defined as

$$\text{show\_security}[\text{mortgage\_no}]() = \text{d-open } (\textsc{Security\_List}[\text{mortgage\_no}])$$

This is a very simple d-operation, which only opens a d-object that is specified by another dialogue type SECURITY_LIST and the value for the key attribute mortgage_no. We omit the definition of this dialogue type.

The integration of schemata $\mathcal{S}_1$ and $\mathcal{S}_2$ and the integration of views V_1 and V_2 has only a marginal effect on this d-operation, which becomes a d-operation for the integrated view. As we only open a d-object, the change is already reflected in adapting the defining query for the underlying view of the dialogue type SECURITY_LIST.

Let us consider another d-operation loan_update with selection attribute *sel* = loan_no, i.e. we request again that a loan, this time a personal loan, be selected from the list of loans of a customer. For this operation we also request additional input from a user, which is specified by the parameters amount and interest, both with domain *Decimal*, and terms with domain *String*. So we may specify

$$\begin{aligned}
&\text{loan\_update}[\text{loan\_no}](\text{amount}{:}\mathit{Decimal}, \text{interest}{:}\mathit{Decimal}, \text{terms}{:}\mathit{String}) = \\
&\qquad \texttt{let } (i,(t',l,a,p,b,e,t)) = \mathbf{I}x \in \textsc{Personal\_Loan}. \\
&\qquad\qquad\qquad x.\text{loan\_no} = \mathit{loan\_no} \\
&\qquad \texttt{in } \textsc{Personal\_Loan} :\& \ (i,(t',l,\text{amount},\text{interest},b,e,\text{terms}))
\end{aligned}$$

That is, we select the data about the loan with the selected loan number from the database, which is expressed by "the unique x ...", written as $\mathbf{I}x \in$ PERSONAL_LOAN...., then update (: &) this tuple in the database using the input provided by the user, i.e. the values of the parameters amount, interest and terms.

In adapting this d-operation to the integrated view, we have to replace the specification of values in PERSONAL_LOAN according to the changes to the integrated schema.

6 View Integration in Web Information Systems

A *web information system* (WIS) is a database-backed information system that is realized and distributed over the web with user access via web browsers. Information is made available via pages including a navigation structure between them and to sites outside the system. Furthermore, there should also be operations to retrieve data from the system or to update the underlying database(s).

The methodology for WIS design in [22,25] emphasises abstraction layers and the co-design of structure, operations and interfaces. As WISs are open systems in the sense that everyone who has access to the web may turn up as a user, their design requires a clear picture of the intended users and their behaviour. This includes knowledge about the used access channels and end-devices. At a high level of abstraction this first leads to *storyboarding*, an activity that addresses the design of underlying application stories. Storyboarding first describes a *story space* by scenes and actions on these scenes. Furthermore, it describes the *actors* in these scenes, i.e. groups of users of the WIS. Actor modelling leads to roles, profiles, goals, preferences, obligations and rights. Finally, the actors are linked to the story space by means of *tasks*.

Further on in the development process the scenes in the story space have to be adequately supported. For this the methodology focuses on *media types*, which cover extended views, adaptivity and hierarchies. In a nutshell, a media type is an extended view on some underlying database schema. These views are built in a way that they capture the complex content and navigation structure that is to be presented to a user. In order to capture the navigation structure we use abstract identifiers in the views, both for having a unique handle for each object in the reult and for being able to reference this object. So the defining queries must be powerful enough to create these identifiers. As a consequence, views are no longer independent from each other in the sense that creating one view may require to create another one simultaneously. This is a well known problem from querying object oriented databases, for which the solution by IQL [1] can be adopted. An alternative would be to use the extended query algebra from [25].

The view integration (or cooperation) problem for WISs arises in the same way as the one for data warehouses. Storyboarding results among others in a collection of elementary scenes, each of which has to be supported by a media type, i.e. an extended view. Defining these views gives a collection of views, while the actual database support requires the integration of these views.

Adaptivity to users, channels and end-devices mainly concerns the question, whether all information or only the most important part of it is to be presented to a user. By specifying on a conceptual level what these "most important" parts are and which parts have to be kept together we may then leave the technical realisation of adaptivity to an algorithmic solution. Hierarchies enable the presentation of information at different levels of granularity allowing a user to switch between these levels. Such hierarchies are common in OLAP systems and the principles can be borrowed from there. We will not deal with hierarchies in this article.

Similar to OLAP systems it is likely that the content that is to be made available to users is modelled, before an underlying database schema and thus defining queries for media types have been defined. This leads unavoidably to a view integration problem. In addition, we have to cope with the implications for operations and for adaptivity – postponing the hierarchies to later. Operations can be dealt with in the same way as for data warehouses and OLAP systems.

The transformation rules in Section 4 already capture the core of the integration process, the integration of the views. So we only have to look at the extensions that arise from media types. Operations have already been dealt with in Section 5, so we are left with the adaptivity, i.e. we just explain in this section how the rules have to be extended to be applicable to media types. Thus, we may concentrate on the aspect of adaptivity.

6.1 Extended Views for Web Information Systems

At the core of a media type we have the same idea as for dialogue types with some subtle distinctions. The idea is the same as for the dialogue types: The content of a web page can be described by a complex value of some type, which may contain references to other values. These references abstract from links between pages. Then abstract to types and add operations. This leads first to the notion of interaction type as defined (in a simplified form) next.

Definition 6.1. An *interaction type* I over a HERM schema $(\mathcal{S}, \Sigma)$ consists of a view $V_I = (\mathcal{S}_I, q_I)$, and a set $\mathcal{O}$ of *dialogue operations.*

The difference between interaction and dialogue types is that the view schema $\mathcal{S}_I$ of an interaction type may be much more complicated. In particular, we permit references (or roles) between the *interaction objects* of any type in $\mathcal{S}_I$ that result from applying the defining query q_I to an instance over $(\mathcal{S}, \Sigma)$. This usually requires q_I to be written in a highly expressive query language. As already indicated above, we require languages that create abstract identifiers. It is known from [1] that this identifier creation may involve computing a fixed point, which is more expressive than what we would use in simpler SQL-like query languages.

Furthermore, for each interaction object (i, v) in $q_I(db)$ we interpret the abstract identifier i as a surrogate for a URL address. In this way the queries used in interaction types already define the navigation structure of the WIS. This is a fundamental difference to work by others as e.g. in [4], where views are only used to extract data, whereas the navigation structure is added later on in a separate design step.

Apart from these subtle differences media types extend interaction types by *cohesion* in order to enable adaptivity. *Cohesion* introduces a controlled form of information loss exploiting the partial order $\geq$ on nested attributes.

Definition 6.2. If X_M is the representing attribute of an interaction type M and $sub(X_M)$ is the set of all nested attributes Y with $X_M \geq Y$, then a preorder $\preceq_M$ on $sub(X_M)$ extending the order $\geq$ is called a *cohesion preorder*.

Large elements in $sub(X_M)$ with respect to $\preceq_M$ define information to be kept together, if possible. Clearly, X_M is maximal with respect to $\preceq_M$. This enables a controlled form of information decomposition [25]. So we obtain the following (simplified) definition of a media type.

Definition 6.3. A *media type* is an interaction type M together with an cohesion preorder $\preceq_M$.

As media types are extended views over some database schema, any instance of such a schema defines a set of objects for the media types, which we call *media objects.* The difference between these media objects and the interaction objects that we considered first is that the existence hierarchies leads to different version of these objects, while adaptivity replaces a single object by an interlinked sequence of objects, i.e. the information is fragmented. Nevertheless, these versions and sequences are determined by the media type, and for our purpose of integrating these types we do not have to look at the objects at all.

[25] contains an algorithmic approach to adaptivity based on cohesion preorders. This algorithm is not relevant for our purposes here. In that article also alternatives to cohesion preorders, which consist of proximity values, but lead to the same results, are discussed.

6.2 Transformation Rules for Media Types

The addition of cohesion preorders requires further extensions to our view integration methodology. We now need additional transformation rules dealing with the impact of the view integration on the cohesion. We may dispense with discussing operations as these have already been captured by the rules for dialogue types. So the following rules are either extension rules to the basic transformation rules 1-26 or to the rules 27-37 for dialogue types.

Rule 38. In case rules 1 and 27 are applied to a media type, if $X(A'_1, \dots, A'_k)$ $(k \leq m)$ appears in $Y \in sub(X_M)$, replace it by $A'_1, \dots, A'_k$.

Rule 39. In case rules 2 and 28 are applied to a media type, whenever r appears in $Y \in sub(X_M)$, replace it by $r_1^{(r)}, \dots, r_n^{(r)}$.

Same as for dialogue types we may ignore rules 3, 4 and 5 for media types.

Rule 40. In case rules 6 and 29 are applied to a media type omit A in $Y \in sub(X_M)$, whenever it appears.

Rule 41. In case rules 7 and 30 are applied to a dialogue type omit $r_1, \dots, r_n$ in $Y \in sub(X_M)$, whenever it appears.

These extensions capture the first group of rules dealing with schema restructuring. For the second group of rules dealing with the shifting of attributes we obtain the following extension rules in case the rules are applied to media types.

Rule 42. In case rules 8 and 31 are applied to a media type omit A_i in $Y \in sub(X_M)$ associated with $R_{i,new}$, whenever it appears..

Rule 43. In case rules 9 and 32 are applied to a media type omit A_i in $Y \in sub(X_M)$ associated with R_{new}, whenever it appears..

As for dialogue types the last two extension rules have no effect on R_{new} or $R_{i,new}$, as the extension of the cohesion preorder has to be defined for these new types.

Rule 44. In case rules 10 and 33 are applied to a media type replace r_n by $r_{1,new}$ in $Y \in sub(X_M)$ associated with R_{new}.

Rule 45. In case rules 11 and 34 are applied to a dialogue type omit A in $Y \in sub(X_M)$ associated with $R_{n,new}$.

For the third group of rules, i.e. rules 12-15 dealing with schema extension only rules 12 and 15 give rise to reasonable extension rules for media types. This defines the following two extension rules.

Rule 46. In case rule 12 is applied to a media type, add A to all $Y \in sub(X_M)$ associated with R_{new} and extend the now incomplete cohesion preorder.

Rule 47. In case rule 15 is applied to a media type, define the Cartesian product of the cohesion preorders and extend the resulting flawed cohesion preorder.

For the group of rules dealing with type integration, i.e. rules 16-19, no reasonable extension rules for media types can be defined, as we deal with new types. Finally, for the group of rules dealing with integrity constraints again only rules 20, 21 and 23 give rise to the following three extension rules for dialogue types.

Rule 48. In case rules 20 and 35 are applied to a media type replace r_2 by r_{new} in $Y \in sub(X_M)$, whenever it appears.

Rule 49. In case rules 21 and 36 are applied to a media type replace r_2 by $r_{2,new}$ in $Y \in sub(X_M)$, whenever it appears.

Rule 50. In case rules 23 and 37 are applied to a media type remove A in $Y \in sub(X_M)$, whenever it appears.

6.3 Case Study

Let us continue the case study from the previous sections. What we have to do is to study how a cohesion preorder associated with a view will be affected by the view integration process.

Example 6.1. Consider the view V_1 on schema $\mathcal{S}_1$ from Example 4.1. The representing attribute of the type CUSTOMER_CR is a tuple attribute with five components, two of which are set attributes. As a shortcut we may write $(C, N, D, O\{(T_1, A_1, F)\}, L\{(T_2, A_2, B)\})$ for this attribute.

Here C represents customer_no, N the name, D the date_of_birth, and O and L the income-obligations and loans, respectively. If this is used as the basis of a media type M, it is easy to calculate that there are 648 attributes in $sub(X_M)$. As this is a bit too big for our purposes here, let us concentrate only on one component, say on the attribute $O\{(T_1, A_1, F)\}$, for which T_1, A_1 and F represent the attributes type, amount and frequency, respectively.

The corresponding lattice is shown in Figure 6. Thus, we may define the following cohesion preorder:

$$O\{(T_1, A_1, F)\} \preceq O\{(T_1, A_1)\} \preceq O\{(T_1, F)\} \preceq O\{(T_1)\} \preceq O\{(A_1, F)\}$$
$$\preceq O\{(A_1)\} \preceq\succeq O\{(F)\} \preceq O\{()\} \preceq ()$$

If we now remove the attribute F, this preorder reduces to

$$O\{(T_1, A_1)\} \preceq O\{(T_1)\} \preceq O\{(A_1)\} \preceq\preceq O\{()\} \preceq ()$$

.

$$\begin{tikzcd}
& O\{(T_1, A_1, F)\} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
O\{(T_1, A_1)\} \arrow[d, no head] \arrow[dr, no head] & O\{(T_1, F)\} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & O\{(A_1, F)\} \arrow[dl, no head] \arrow[d, no head] \\
O\{(T_1)\} \arrow[dr, no head] & O\{(A_1)\} \arrow[d, no head] & O\{(F)\} \arrow[dl, no head] \\
& O\{()\} \arrow[d, no head] & \\
& () &
\end{tikzcd}$$

Fig. 6. Cohesion lattice

7 Conclusion

In this article we revisited schema and view integration and cooperation. We presented a method for schema (or view) integration following the framework in [16], i.e. we first "clean" given schemata by removing name conflicts, synonyms and homonyms, then we add inter-schema constraints, and to this schema we then apply formal equivalence transformation or augmentation rules. The transformation and augmentation rules are correct in the sense that they will always result in a new schema / view that is equivalent to the original one or dominates it. For this we introduced a new concept of schema equivalence and dominance based on computable queries.

This new notion of equivalence and dominance subsumes all reasonable existing ones, so we do not lose valuable transformation rules. On the other hand it is rather general, so that the set of transformation rules can be extended if needed without violating schema equivalence. It is an open problem to characterise exactly, which integration problems would require which notion of dominance and equivalence. The work in this article did not aim at solving this interesting theoretical problem. We concentrated instead on finding a reasonable approach to view integration and cooperation that is theoretically founded, but pragmatically oriented.

In follow-on steps we applied the view integration and cooperation framework to databases, data warehouses and web information systems. In the first two cases the key concept is that of a dialogue type, which is a view extended by operations. In the last case the key concept is that of a media type, which further extends dialogue types by cohesion in order to facilitate adaptivity. In all three cases the integration of the extended views requested additions to the transformation and augmentation rules.

The extension of view integration and cooperation to data warehouses and web information systems is completely new. So far, the relevant work in the literature concentrated exclusively on the structural aspect, leaving the implications to view extensions aside. As the extensions to the views are the core contribution of dialogue types (for the case of data warehouses) and media types (for web information systems), respectively, it is important to have a pragmatic view integration and cooperation method at hand. Without such a method the development methodology would be incomplete.

Thus, the presented approach to view integration and cooperation turns out to be general enough to be applicable to a large class of data-intensive information systems. By means of the areas for which we demonstrated this applicability, it contributes to a decisive aspect in systems development. We demonstrated this applicability by a case study for all three areas. However, the integration of view integration / cooperation into an overall framework for systems development has been left out of this article. For dialogue-oriented enterprise information systems we refer to [21], and for WISs to [25].

On a more theoretical basis the new concept of schema dominance and equivalence can be the subject of a deeper investigation. In particular, it opens the possibility to develop even more powerful transformation and augmentation rules and to investigate the implications on complexity. Such problems are left for future research.

References

1. Abiteboul, S., and Kanellakis, P. C. Object identity as a query language primitive. In *Proceedings SIGMOD 1989* (1989), pp. 159–173.
2. Atzeni, P., Gupta, A., and Sarawagi, S. Design and maintenance of data-intensive web-sites. In *Proceeding EDBT'98*, vol. 1377 of *LNCS*. Springer-Verlag, Berlin, 1998, pp. 436–450.
3. Biskup, J., and Convent, B. A formal view integration method. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1986, pp. 398–407.
4. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. *Designing Data-Intensive Web Applications.* Morgan Kaufmann, San Francisco, 2003.
5. Chandra, A., and Harel, D. Computable queries for relational data bases. *Journal of Computer and System Sciences 21* (1980).
6. Feyer, T., Kao, O., Schewe, K.-D., and Thalheim, B. Design of data-intensive web-based information services. In *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000)*, Q. Li, Z. M. Ozsuyoglu, R. Wagner, Y. Kambayashi, and Y. Zhang, Eds. IEEE Computer Society, 2000, pp. 462–467.
7. Feyer, T., Schewe, K.-D., and Thalheim, B. Conceptual modelling and development of information services. In *Conceptual Modeling – ER'98*, T. Ling and S. Ram, Eds., vol. 1507 of *LNCS*. Springer-Verlag, Berlin, 1998, pp. 7–20.
8. Garzotto, F., Paolini, P., and Schwabe, D. HDM - a model-based approach to hypertext application design. *ACM ToIS 11*, 1 (1993), 1–26.

9. Hull, R. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing 15*, 3 (1986), 856–886.
10. Hull, R., and Yap, C. K. The FORMAT model: A theory of database organisation. *Journal of the ACM 31*, 3 (1984), 518–537.
11. Inmon, W. *Building the Data Warehouse.* Wiley & Sons, New York, 1996.
12. Kedad, Z., and Métais, E. Dealing with semantic heterogeneity during data integration. In *Conceptual Modeling – ER'99* (1999), J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, Eds., vol. 1728 of *LNCS*, Springer-Verlag, pp. 325–339.
13. Koh, J., and Chen, A. Integration of heterogeneous object schemas. In *Entity-Relationship Approach - ER'93*, R. Elmasri, V. Kouramajian, and B. Thalheim, Eds., vol. 823 of *LNCS.* Springer-Verlag, 1994, pp. 297–314.
14. Larson, J., Navathe, S. B., and Elmasri, R. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering 15*, 4 (1989), 449–463.
15. Lehmann, T. *Ein pragmatisches Vorgehenskonzept zur Integration und Kooperation von Informationssystemen.* PhD thesis, TU Clausthal, 1999.
16. Lehmann, T., and Schewe, K.-D. A pragmatic method for the integration of higher-order Entity-Relationship schemata. In *Conceptual Modeling - ER 2000*, A. H. F. Laender, S. W. Liddle, and V. C. Storey, Eds., vol. 1920 of *LNCS.* Springer-Verlag, 2000, pp. 37–51.
17. Lewerenz, J., Schewe, K.-D., and Thalheim, B. Modelling data warehouses and OLAP applications using dialogue objects. In *Conceptual Modeling – ER'99* (1999), J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, Eds., vol. 1728 of *LNCS*, Springer-Verlag, pp. 354–368.
18. Ludäscher, B., and Gupta, A. Modeling interactive web sources for information mediation. In *Advances in Conceptual Modeling*, P. P.-S. Chen, Ed., vol. 1727 of *LNCS.* Springer-Verlag, 1999, pp. 225–238.
19. Qian, X. Correct schema transformations. In *Advances in Database Technology - EDBT'96*, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds., vol. 1057 of *LNCS.* Springer-Verlag, 1996, pp. 114–126.
20. Schewe, K.-D. The power of media types. In *Web Information Systems Engineering – WISE 2004*, M. Papazoglou, S. Su, and X. Zhou, Eds., vol. 3306 of *LNCS.* Springer-Verlag, 2004, pp. 233–238.
21. Schewe, K.-D., and Schewe, B. Integrating database and dialogue design. *Knowledge and Information Systems 2*, 1 (2000), 1–32.
22. Schewe, K.-D., and Thalheim, B. Modeling interaction and media objects. In *Natural Language Processing and Information Systems: 5th International Conference on Applications of Natural Language to Information Systems, NLDB 2000*, M. Bouzeghoub, Z. Kedad, and E. Métais, Eds., vol. 1959 of *LNCS.* Springer-Verlag, Berlin, 2001, pp. 313–324.
23. Schewe, K.-D., and Thalheim, B. Reasoning about web information systems using story algebras. In *Advances in Databases and Information Systems – ADBIS 2004*, G. Gottlob, A. A. Benczúr, and J. Demetrovics, Eds., vol. 3255 of *LNCS.* Springer-Verlag, 2004, pp. 54–66.
24. Schewe, K.-D., and Thalheim, B. Structural media types in the development of data-intensive web information systems. In *Web Information Systems*, D. Taniar and W. Rahayu, Eds. IDEA Group, 2004, pp. 34–70.
25. Schewe, K.-D., and Thalheim, B. Conceptual modelling of web information systems. *Data and Knowledge Engineering 54*, 2 (2005), 147–188.

26. Schewe, K.-D., and Zhao, J. Balancing redundancy and query costs in distributed data warehouses – an approach based on abstract state machines. In *Conceptual Modelling 2005 – Second Asia-Pacific Conference on Conceptual Modelling* (Newcastle, Australia, 2005), S. Hartmann and M. Stumptner, Eds., vol. 43 of *CRPIT*, Australian Computer Society, pp. 97–105.
27. Schwabe, D., and Rossi, G. An object oriented approach to web-based application design. *TAPOS 4*, 4 (1998), 207–225.
28. Sciore, E., Siegel, M., and Rosenthal, A. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM TODS 19*, 2 (1994), 254–290.
29. Spaccapietra, S., and Parent, C. View integration – a step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering 6*, 2 (1994), 258–274.
30. Thalheim, B. *Entity-Relationship Modeling: Foundations of Database Technology.* Springer-Verlag, 2000.
31. Thalheim, B., and Düsterhöft, A. SiteLang: Conceptual modeling of internet sites. In *Conceptual Modeling – ER 2001*, H. S. K. et al., Ed., vol. 2224 of *LNCS*. Springer-Verlag, Berlin, 2001, pp. 179–192.
32. Theodoratos, D., and Sellis, T. Data warehouse schema and instance design. In *Conceptual Modeling – ER'98* (1998), vol. 1507 of *LNCS*, Springer-Verlag, pp. 363–376.
33. Thomson, E. *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons, 2002.
34. Turull Torres, J. M. On the expressibility and computability of untyped queries. *Annals of Pure and Applied Logic 108*, 1-3 (2001), 345–371.
35. Van den Bussche, J. *Formal Aspects of Object Identity in Database Manipulation.* PhD thesis, University of Antwerp, 1993.
36. Widom, J. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management* (1995), ACM.
37. Zhao, J., and Ma, H. Quality-assured design of on-line analytical processing systems using abstract state machines. In *Proceedings of the Fourth International Conference on Quality Software (QSIC 2004)* (Braunschweig, Germany, 2004), H.-D. Ehrich and K.-D. Schewe, Eds., IEEE Computer Society Press.
38. Zhao, J., and Schewe, K.-D. Using abstract state machines for distributed data warehouse design. In *Conceptual Modelling 2004 – First Asia-Pacific Conference on Conceptual Modelling* (Dunedin, New Zealand, 2004), S. Hartmann and J. Roddick, Eds., vol. 31 of *CRPIT*, Australian Computer Society, pp. 49–58.

Semantic Integration of Tree-Structured Data Using Dimension Graphs*

Theodore Dalamagas[1], Dimitri Theodoratos[2], Antonis Koufopoulos[1], and I-Ting Liu[2]

[1] School of Electr. and Comp. Engineering, National Technical University of Athens, Athens, GR 15773
{dalamag, akoufop}@dblab.ece.ntua.gr

[2] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102
{dth, il2}@cs.njit.edu

Abstract. Nowadays, huge volumes of Web data are organized or exported in tree-structured form. Popular examples of such structures are product catalogs of e-market stores, taxonomies of thematic categories, XML data encodings, etc. Even for a single knowledge domain, name mismatches, structural differences and structural inconsistencies raise difficulties when many data sources need to be integrated and queried in a uniform way. In this paper, we present a method for semantically integrating tree-structured data. We introduce dimensions which are sets of semantically related nodes in tree structures. Based on dimensions, we suggest dimension graphs. Dimension graphs can be automatically extracted from trees and abstract their structural information. They are semantically rich constructs that provide query guidance to pose queries, assist query evaluation and support integration of tree-structured data. We design a query language to query tree-structured data. The language allows full, partial or no specification of the structure of the underlying tree-structured data used to issue queries. Thus, queries in our language are not restricted by the structure of the trees. We provide necessary and sufficient conditions for checking query satisfiability and we present a technique for evaluating satisfiable queries. Finally, we conducted several experiments to compare our method for integrating tree-structured data with one that does not exploit dimension graphs. Our results demonstrate the superiority of our approach.

1 Introduction

Nowadays, huge volumes of data are posted and retrieved through the Web. Despite this vast exchange of information, there is no consistent and strict organization of data, raising difficulties for its sharing and processing. For the Web to reach its full potential and become a universally accessible platform, it should

* Work supported in part by PYTHAGORAS EPEAEK II programme, EU and Greek Ministry of Education, co-funded by the European Social Fund (75%) and National Resources (25%).

support effective retrieval and integration of the posted data. Tree structures play an important role in this task, providing a means to organize the information on the Web. Taxonomies of thematic categories, concept hierarchies, e-commerce product catalogs are examples of such structures. The XML language [3] is nowadays the standard data exchange format on the Web for tree-structured data. The recent proliferation of XML-based standards and technologies for managing data on the Web demonstrates the need for effective and efficient management of tree-structured data. Even if data is not stored natively in tree structures, export mechanisms make data publicly available in tree structures to enable its automatic processing by programs, scripts, and agents on the Web [13].

Querying capabilities on these structures are provided either through browsing tools or through path expression queries. For the former, the user should navigate among nodes to identify data. For the latter, she should form queries using some of the query languages proposed in the literature (e.g. XPath [4], XQuery [5]). For example, /notebooks/new/ultralight[price<2000] is an XPath expression that will retrieve new, ultralight notebooks that cost less than \$2000.

The Problem. Under the information integration perspective [24], a challenging issue is to integrate and query in a uniform way many tree-structured data sources. Users should be able to pose a query on a 'global' tree structure. The answer of the query is formed using data retrieved from 'local' data sources. The whole process should be transparent to the user in the sense that she need not know details about the local data sources and the query processing. Even for a single knowledge domain, integrating tree-structured data turns out to be a hard task due to name mismatches, structural differences and structural inconsistencies. Name mismatches appear because tree structures lack semantic information. For example, laptop computers might be referred to as notebooks in one product catalog but as portables in another catalog. In this paper, we do not focus on this issue and we assume that it is resolved using well-known schema matching techniques [28]. Structural differences and, far more important, structural inconsistencies appear because of the different possible ways of organizing the same data in tree structures. For example, a structural difference exists when a category appears in a product catalog but does not appear in another. A structural inconsistency appears when a product catalog for notebooks classifies new, SONY notebooks with 10″ display in the path `/notebooks/new/SONY/10″`, while another catalog classifies the same products in the path `/SONY/notebooks/10″/new`. As a result, a path expression query in the form of `/notebooks/SONY/new/10″` should be reformulated to match the structure of each catalog.

Current tree-structured data query languages (e.g. Xquery) handle this issue in a procedural way, in the sense that the user should explicitly specify structural differences and irregularities as part of the query itself. For example, to identify new, SONY notebooks with 10″ display as in the previous example, the user should explicitly specify alternate sequences for categories and use disjunctions in her query. Requiring such a strict specification raises difficulties in forming queries.

A naive approach to cope with structural differences and inconsistencies is to generate different versions of the initial query, considering different subsets of nodes involved in its path expressions and their different orderings. Clearly this is not efficient due to the large number of queries that need to be generated. Instead of reordering the initial query, relaxing techniques can be used to change its form and search for answers in local data sources [8,19]. For example, query `/new/monitors/CRT/17″` can be relaxed to `/new/monitors//17″`, where '//' denotes ancestor-descendand relationship. The relaxed query asks for new 17″ monitors, without specifying whether these monitors are TFT or CRT, and permits other nodes to appear between nodes `monitor` and `17″`. Nevertheless, such techniques return approximate and not exact answers.

A traditional approach to information integration defines mapping rules between a global structure and the local structures used in the sources [15]. For example, given the rules `(notebooks/Sony)` → `(Sony/notebooks)` and `(new/10″)` → `(10″/new)` for one catalog, query `/notebooks/Sony/new/10″` will become `/Sony/notebooks/10″/new` in order to match its structure. Such approaches require extensive manual effort, since the global schema is difficult to contruct and the rules should be hard-coded in the integration application.

Our Approach. In this paper, we suggest a novel approach to the integration of tree-structured data. Tree-structured data provides mainly syntactic and not semantic information. However, there are inherent semantics, for instance, subcategories of a main category in a catalog are usually related under a semantic interpretation given by the author of that particular catalog. Subcategories `notebooks` and `desktops`, for example, indicate that there are certain items that can be characterized with a property that indicates their product type (i.e. notebook or desktop). Our approach captures and exploits such a semantic information for querying tree-structured data (called here value trees). Such a semantic information plays a two-fold way: (a) it becomes part of a query language itself and (b) it provide a means for optimizing queries in tree-structured data.

We introduce the concept of a dimension that groups together semantically related values (nodes). For instance, `IBM, Sony, HP` can be values of dimension `brand`. We assume that the semantic interpretation of the values is available. The nodes of a value tree are partitioned into dimensions. The different dimensions of a value tree are related through precedence relationships incurred by the parent-child and ancestor-descendant relationships of their nodes. We capture these precedence relationships between dimensions of a value tree into the concept of a dimension graph for the value tree. Dimension graphs can be automatically extracted from trees and abstract their structural information. They are semantically rich constructs that provide query guidance to pose queries, assist query evaluation and support integration of tree-structured data.

Query conditions involve dimensions, and thus query formulation is not dependent on the structure of value trees. For instance, the query above asking for new Sony notebooks with 10″ display would look like: `pc_type = Notebooks`, `brand = Sony`, `condition = New`, `display_size = 10″`, where `pc_type`, `bra-`

`nd`, `condition` and `display_size` are dimensions. No order is a priori specified among values (nodes), unless the user wants to impose a partial or a total order. The system uses the dimension graph of the value tree to identify orderings of the values that can possibly exist in the value tree. Only these value orderings will be used to compute the answer of the query on the value tree. This step of the computation of the query answer is performed before the query evaluation reaches the value tree which is, in general, much larger than its dimension graph. Dimension graphs provide also the means for integrating different data sources. This is achieved through the creation of a 'global dimension graph' which is a merging of the dimension graphs of the data sources. User queries are issued against the global graph and they are then translated into queries on the local dimension graphs where they are evaluated.

Contribution. The main contributions of the paper are the following:

- We introduce dimensions to record semantic information for the nodes of value trees. We also introduce dimension graphs to capture structural information on value trees. Given a partitioning of the nodes of value trees into dimensions, the dimension graphs for these value trees can be automatically extracted.
- We design a query language for this framework. Queries are not issued directly on value trees but on their dimensions. Therefore, queries are not cast on the structure of a specific value tree. The user can optionally specify parent-child and/or ancestor-descendent relationships between dimensions in a query.
- We show how queries on value trees can be evaluated. In this process, the dimension graph of the value tree plays a two-fold role. First, it allows identifying an unsatisfiable query, that is, a query that does not have an answer on any value tree underlying the dimension graph. Second, if the query is satisfiable, it allows determining those orderings of dimensions that can possibly generate non-empty answers. Thus, dimension graphs prune useless dimension orderings at an early stage of the evaluation of a query. We provide necessary and sufficient conditions for a query to be unsatisfiable, and we show how path expressions to be evaluated on value trees are generated from non-filtered dimension orderings.
- We present a method for integrating different data sources through the creation of a global dimension graph. We show how queries on the global dimension graph can be evaluated first at the global site and then at the data sources.
- We carry out several experiments to compare our approach to one that does not exploit dimension graphs in the integration of tree-structured data sources. Our results demonstrate the superiority of our approach.
- Our approach can be applied to the integration of tree-structured data in various application areas including the integration of product catalogs with different structures in e-commerce applications or the integration of XML data from similar knowledge domains that conform to different DTDs.

Outline. The rest of the paper is organized as follows. The next section discusses related work. In Section 3, we introduce dimensions and we define dimension graphs for value trees. Section 4 presents the query language used to pose queries on dimension graphs. It also shows how queries can be checked for unsatisfiability and how they are evaluated on the underlying value trees. In Section 5, we show how data sources can be integrated and queried in a uniform way. Section 6 presents the experimental evaluation of our approach. Finally, Section 7 concludes the paper and presents further work.

2 Related Work

Tree-structured data integration has become a popular research issue, especially after the latest increase in the use of XML to encode data. The vast majority of suggested techniques follow traditional information integration methods. Given a predefined virtual structure (i.e. a global schema), mapping rules are defined between the virtual structure and the local structures of the sources. The initial query is posed on the virtual structure and transformed to queries on local structures using the mapping rules.

The Xyleme system [15] copes with the problem of integrating XML data sources by defining and querying views. The user creates a DTD to act as the global schema. Queries are expressed using query pattern trees. Query evaluation exploits mapping rules in the form of path-to-path correspondences. The Agora system [25] integrates relational and tree-structured XML data sources under a global XML schema. Users express a query in the XQuery language over a given global XML schema. The query is translated to an intermediate SQL query, and then to SQL queries on local data sources. In [7], a methodology to integrate XML Web resources is presented. The global schema used to pose OQL queries is a lightweight ontology with object-oriented model primitives. Query evaluation is based on mapping rules in the form of path-to-path correspondences. In [14], an XML integration system based on the YAT model is described. YAT queries on a global schema are translated to an algebraic form based on a set of algebraic operators. The algebraic translations of the initial query are further processed using YAT mapping rules before they are evalutated in the data sources. In [26], an adaptive evaluation technique is presented for querying XML-based electronic catalogs. Given a global DTD, Xpath queries are formed and reformulated to queries on the local catalogs. Query evaluation is based on mapping rules between elements of the global DTD and their representation in local catalogs. A language for querying XML sources is presented in [12]. Queries are issued on a global conceptual schema and translated to local data sources using a pre-defined set of mappings. A detailed survey about general schema integration techniques can be found in [29]. Our approach differs than the aforementioned traditional information integration techniques in that it does not require the manual definition of hard-coded mapping rules between the virtual tree structure and the local structures. Further, our approach does not use a predefined virtual global schema. In contrast, a global dimension graph is constructed automatically from local data sources.

Relevant to our work are also integration techniques where a global structure is not predefined, but rather is constructed using schema descriptions extracted from local data sources. In [16], global schemas are generated from the semantic integration of conceptual schemas extracted from DTDs of XML data sources. Similarly, XClust [23] generates DTDs to act as glocal schemas, applying clustering methods to detect similar DTDs prior to their integration. Techniques that extract DTDs from collections of XML documents are also presented in [17]. In [9], a grammar-based model based on tree automata is used to integrate DTDs. Contrary to our approach, these papers do not deal with query evaluation.

Schema-based descriptions for data with little or no apparent structure have also been suggested for semistructured databases [6]. Dataguides are introduced in [18]. They are structural summaries for semistructured data, useful for formulating queries, storing statistics about paths and nodes, and enabling query optimization. Statistical synopses for graph-structured XML databases are suggested in [27]. In [11], graph schemas are introduced to formulate, optimize and decompose queries for semistructured data. Database conformity to a graph schema is based on notion of graph simulation. These approaches do not provide a direct solution to the problem of structural inconsistencies and differences in data sources that we address here. Further, they are purely syntactic. In contrast to our approach, they do not exploit semantic information.

Integrating tree-structured data is also a popular issue in e-commerce applications. A system to integrate product classification schemes is presented in [10]. Using the source descriptions, the system generates a glocal schema based on inter-schema and intra-schema relationships determined manually. In [22], a method to integrate e-commerce catalogs is presented. Related categories are organized in term vectors. Membership rules are defined to encode parent/child relationship between categories. The integrated catalog contains all information from the original catalogs, maintaining at the same time their structural information. Facet classification hierarchies [1,2] also exploit sets of semantically related categories. Facets provide different classification schemes for the same data. In [31], the authors present faceted taxonomies for Web catalogs. They investigate the problem of invalid navigation paths produced after the combination of taxonomies corresponding to different facets. None of these papers suggest query evaluation techniques. Preliminary work on querying tree-structured data using dimension graphs has also been presented in [30]. However, this paper does not consider the problem of integrating tree-structured data sources.

3 Data Model

In this section we present a data model for tree-structured data. We introduce a type of trees, called value trees, to represent tree-structured data. We also discuss the notion of a dimension, based on which a partitioning can be enforced on value trees.

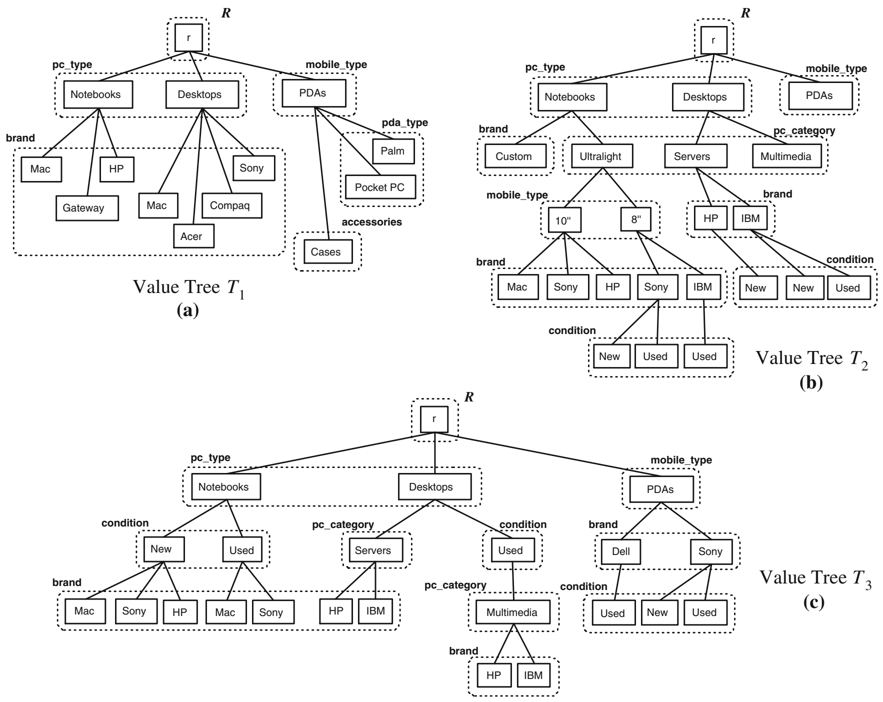


Fig. 1. Value trees T_1, T_2 and T_3

3.1 Value Trees and Dimensions

We assume a set of values V that includes a special value r. The elements of V are used to build value trees.

Definition 1. *A* value tree *is a rooted node-labeled tree T, such that:*
(a) Each node label in T belongs to V.
(b) Value r labels only the root of T.
(c) There are no sibling nodes in T labeled by the same value. □

Example 1. Figure 1 shows examples of value trees T_1, T_2 and T_3 (for the moment, the dotted labeled rectangles that group the nodes should be ignored). These value trees are parts of taxonomies used to categorize products related to computer equipment. The same value may label multiple nodes in a value tree. For example, value `HP` labels two nodes in T_2. Notice that there are stuctural differences and inconsistencies between value trees T_1, T_2 and T_3, although they refer to the same knowledge domain. For example, there are nodes labeled `Multimedia` or `Servers` in T_2 and T_3, even though no such nodes appear in T_1. Also, a node labeled `Used` is a child of a node labeled `Sony` in T_2, although the opposite holds in T_3. Note that we assume that naming mismatches have been resolved. For instance, nodes labeled by the same value in different trees refer to the same real world concept. □

Values in set V can be grouped to form dimensions. Intuitively, a dimension is a set of semantically related values. For instance, values `Mac`, `Acer` and `Compaq` can be interpreted as values of a dimension `brand`. A semantic interpretation of values is imposed by a user. A dimension can also be seen as a property with values.

Definition 2. *Let V be a set of values that includes a specific value r. A* dimension set *over V is a partition $\mathcal{D}$ of V that includes a set R whose single element is value r. Each element of $\mathcal{D}$ is called* dimension. □

Example 2. Figure 2 shows a dimension set $\mathcal{D}$ and the names of its dimensions. We use these dimensions and the value trees of Figure 1 as a running example in this paper. □

A dimension set also partitions the nodes of a value tree. We are interested in value trees where every path from the root to a leaf involves values from distinct dimensions. To describe this type of value trees we introduce the concept of *tree conformity* with respect to a dimension set.

Definition 3. *Let $\mathcal{D}$ be a dimension set over a value set V. A value tree T* conforms *to $\mathcal{D}$ iff there are no two nodes on a path in T labeled by values that belong to the same dimension in $\mathcal{D}$.* □

Example 3. Consider, for example, the value trees T_1, T_2 and T_3 of Figure 1. Dotted rectangles labeled by dimensions are used to show the partitioning of nodes into dimensions. The same dimension might label different rectangles in a value tree. In this case, this dimension comprises the nodes confined by all these rectangles. Dimension `pc_type` in T_1 refers to types of personal computers and includes nodes labeled by values `Desktops` and `Notebooks`. Dimension `brand` in T_3 refers to brand names and includes nodes labeled `Mac, Sony, HP, IBM` and `Dell`. All trees T_1, T_2, and T_3 conform to the dimension set $\mathcal{D}$ shown in Figure 2. □

Nodes labeled by values of the same dimension need not be in the same level of a value tree. For example, in T_2, the nodes labeled `10''` and `8''` of dimension

Dimension Set D = { R, pc_type, brand, mobile_type, pda_type, accessories, pc_category, condition }

Dimensions: pc_type = { Notebooks, Desktops }
brand = { Mac, Sony, HP, IBM, Gateway, Acer, Compaq }
mobile_type = { PDAs, 10", 8" }
pda_type = { Palm, Pocket_PC }
accessories = { Cases }
pc_category = { Ultralight, Multimedia, Server }
condition = { New, Used }

Fig. 2. A dimension set and its dimensions

`mobile_type` are not in the same level as the node labeled `PDAs` of the same dimension. A value of a dimension may not appear in a value tree. For example, the value `Ultralight` of dimension `pc_category` does not appear in value tree T_3 nor in T_1, although it appears in T_2. Further, a dimension may have no value in a value tree. For instance, no value of `pc_category` appears in T_1.

In the following we assume that a dimension set $\mathcal{D}$ is given and all value trees conform to $\mathcal{D}$.

3.2 Dimension Graphs

Values of one dimension can label children or descendants of nodes labeled by values of any other dimension in a value tree. However, there are cases where values of one dimension do not label descendants of nodes labeled by values of some other dimension. For example, none of the values `Pocket_PC` and `Palm` of dimension `pda_type` labels a descendant of the nodes labeled by the value `Desktops` or `Notebooks` of dimension `pc_type` in the value tree T_1 of Figure 1. To capture this type of relationship between dimensions in a value tree, we introduce the concept of a dimension graph. Dimension graphs can be automatically extracted from value trees and abstract their structural information. Moreover, they provide semantic query guidance to pose and evaluate queries on value trees (see subsequent sections). Before we give the formal definition of a dimension graph with respect to a value tree, we define dimension graphs as general structures.

Definition 4. *A* dimension graph *over dimension set $\mathcal{D}$ is a directed graph whose nodes are dimensions in $\mathcal{D}$.* □

A *path* in a dimension graph is a sequence $D_1, \ldots, D_k$ of distinct nodes such that there is a directed edge from D_i to D_{i+1}, where $1 \leq i \leq k-1$.

Based on the definitions of dimension graphs as general structures, we proceed to define formally dimension graphs with respect to a value tree.

Definition 5. *Let T be a value tree over a dimension set $\mathcal{D}$. A* dimension graph *of T is a dimension graph (N, E), where N is a set of nodes and E is a set of edges defined as follows:*

(a) There is a node D in N iff there is a value in T that belongs to dimension D.

(b) There is a directed edge in E from node D_i to node D_j iff there are nodes n_i and n_j in T labeled by values $v_i \in D_i$ and $v_j \in D_j$, respectively, such that n_j is a child node of n_i in T.

If $\mathcal{G}$ is a dimension graph of a value tree T, we say that T underlies *$\mathcal{G}$.* □

Example 4. Consider for example the value trees T_1, T_2 and T_3 of Figure 1. Figure 3 shows the dimension graphs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ of T_1, T_2 and T_3, respectively. There is an edge from dimension `mobile_type` to dimension `pda_type` in $\mathcal{G}_1$, since a node labeled `Palm` (a value of `pda_type`) is a child of a node labeled `PDAs` (a value of `mobile_type`) in value tree T_1. Looking at the lower left part of value tree T_3, we note that a node labeled `Mac` (a value of `brand`) is a child of a node labeled

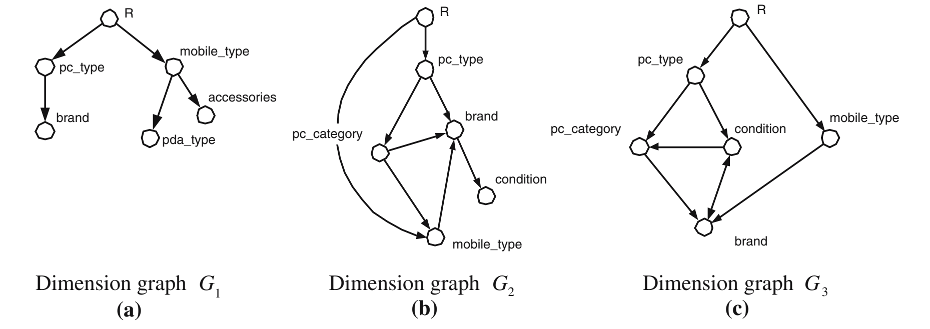


Fig. 3. Dimension Graphs

`New` (a value of `condition`). However, looking at the lower right part of T_3, a node labeled `New` is a child of a node labeled `Sony` (another value of dimension `brand`). Thus, there is an edge from dimension `condition` to dimension `brand` and an edge from `brand` to `condition` in $\mathcal{G}_3$ which are compactly shown in the figures by a double headed edge. □

The dimension graph of a value tree has a particular form. The following propositions describe some of its properties. Their proof is straightforward.

Proposition 1. There is exactly one node in the dimension graph of a value tree having only outgoing edges. □

This unique node is called *root* of the dimension graph. Dimension graphs can have cycles. For instance, in Figure 3(c), dimension graph $\mathcal{G}_3$ has two cycles: `pc_category, brand, condition, pc_category` and `condition, brand, condition`.

Proposition 2. For every node of a dimension graph, there is a path from the root to that node. □

The next proposition relates paths from the root in a value tree to paths from the root in its dimension graph.

Proposition 3. Consider a dimension graph $\mathcal{G}$ of a value tree T over a dimension set $\mathcal{D}$. Let $v_1, \ldots, v_k$ be values from the distinct dimensions $D_1, \ldots, D_k \in \mathcal{D}$, respectively. If $v_1, \ldots, v_k$ label, in that order, nodes on a path in T, then $D_1, \ldots, D_k$ appear in that order on a path from the root in $\mathcal{G}$. □

4 Queries

We present in this section a simple query language and we outline how queries can be evaluated. Our intension is not to provide a full-fledged language. For instance, it does not include selection predicates. Our goal is to show how dimensions can be used to query value trees. Queries in this language are defined on dimension graphs. Roughly speaking, a user poses a query by annotating some dimensions in a dimension graph with permissible sets of values. The answer comprises root-to-leaf paths on the underlying value tree that involve one value from each of these value sets. An interesting feature of the language is that the user has the choice of not specifying or partially specifying parent-child and ancestor-descendant relationships between the annotated dimensions in a query. The system can identify possible orderings of dimensions in the paths of the answer based on the dimension graph only. These orderings are used as patterns for constructing the path expressions that compute the answer of the query on the underlying value tree. All the other orderings of dimensions are excluded from consideration before the computation of the query answer reaches the value tree.

4.1 Syntax

A query on a dimension graph comprises annotations of the graph dimensions with sets of values and specifications of precedence relationships between the graph dimensions.

Definition 6. *Let $\mathcal{G}$ be a dimension graph over a dimension set $\mathcal{D}$. A* query *Q on $\mathcal{G}$ is a pair $(\mathcal{A}, \mathcal{P})$, where:*

(a) $\mathcal{A}$ is a set of expressions of the from $D_i = A_i$, where D_i is a dimension in $\mathcal{G}$ different than R, and A_i is a set of values of dimension D_i or a question mark ("?"). If $D_i = A_i$ belongs to $\mathcal{A}$, we say that D_i is annotated *in Q and A_i is called the* annotation *of D_i in Q. A dimension can be annotated only once in a query.*

(b) $\mathcal{P}$ is a set of precedence relationships *which are expressions of the form $D_i \rightarrow D_j$ or $D_i \Rightarrow D_j$, where D_i and D_j are annotated dimensions of Q.*

Sets $\mathcal{A}$ and $\mathcal{P}$ can be empty. □

We graphically represent a query $Q = (\mathcal{A}, \mathcal{P})$ on a dimension graph $\mathcal{G}$ by labeling its nodes by their annotations in $\mathcal{A}$ and by adding to it a single (resp. double) arrow from node D_i to node D_j for every precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in $\mathcal{P}$. Note that arrows are different than directed edges. A single (double) arrow $D_i \rightarrow D_j$ ($D_i \Rightarrow D_j$) denotes that the values used to annotate D_j should be children (descendants) of the values used to annotate D_i. The unqualified word "arrow" refers indiscreetly to a single or double arrow.

Example 5. Consider the dimension graphs $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$ of Figure 3. Figure 4 shows the graphical representation of different queries on these dimension graphs. Annotated nodes are shown in the figures with black circles. Precedence relationships are shown with single or double arrows from one node to another.

Figure 4(a) represents query $Q_1 = (\mathcal{A}_1, \mathcal{P}_1)$ on dimension graph $\mathcal{G}_1$, where $\mathcal{A}_1 = \{\texttt{brand} = \{\texttt{Mac}, \texttt{Sony}\}, \texttt{pc\_type} = \{\texttt{Desktop}\}\}$ and $\mathcal{P}_1 = \emptyset$. In Q_1 we do not specify any precedence relationships between the annotated notes.

Figure 4(b) represents query $Q_2 = (\mathcal{A}_2, \mathcal{P}_2)$ on dimension graph $\mathcal{G}_2$, where $\mathcal{A}_2 = \{\texttt{pc\_type} =?, \texttt{brand} = \{\texttt{Sony}, \texttt{IBM}\}, \texttt{condition} = \{\texttt{Used}\}\}$ and $\mathcal{P}_2 = \{\texttt{pc\_}$-

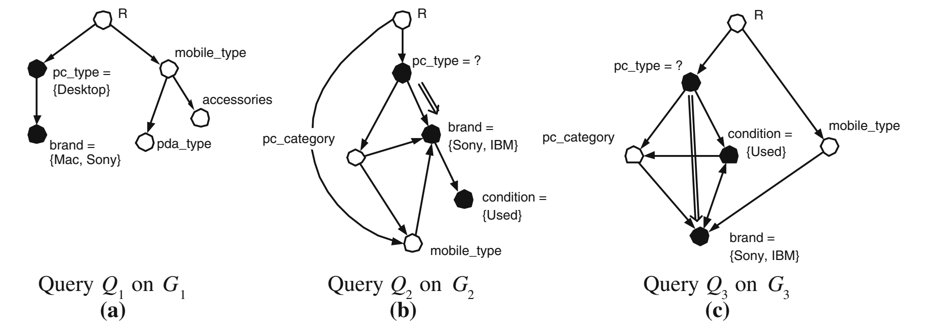


Fig. 4. Graphical Representation of Queries

type $\Rightarrow$ brand}. A double arrow from node pc_type to node brand denotes the precedence relationship in $\mathcal{P}_2$.

Figure 4(c) represents query $Q_3 = (\mathcal{A}_3, \mathcal{P}_3)$ on dimension graph $\mathcal{G}_3$, where $\mathcal{A}_3 = \{$pc_type $=?$, brand $= \{$Sony, IBM$\}$, condition $= \{$Used$\}\}$ and $\mathcal{P}_3 = \{$pc_-type $\Rightarrow$ brand$\}$. Query Q_3 is identical to Q_2 but it is defined on dimension graph $\mathcal{G}_3$. □

In the following we often identify a query with its graphical representation.

4.2 Semantics

The answer of a query on a value tree T is a set of root-to-leaf paths in T compactly represented as a subtree of T.

Definition 7. *Let $\mathcal{G}$ be a dimension graph of a value tree T over a dimension set $\mathcal{D}$, and Q be a query on $\mathcal{G}$. The* answer *of Q on T is the maximal*[1] *subtree T' of T such that:*

(a) T' and T have the same root R.
(b) Every leaf node of T' is a leaf node of T.
(c) Every path from the root to a leaf node in T' includes one value from every value set annotating a node in Q.
(d) Every path from the root to a leaf node in T' includes one value from every dimension naming a node annotated with a question mark in Q.
Therefore, for every annotated node (with a value set or a question mark) in Q, there is one value for the corresponding dimension appearing in every path from the root to a leaf node in T'.
(e) For every path p from the root to a leaf node in T', and for every precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in Q, the value for D_j is a child (resp. descendent) of the value for D_i in p.

If there is no such a subtree T', we say that the answer of Q on T is empty. *Symbol ϵ denotes an empty answer.* □

[1] Maximality is meant with respect to the number of nodes or edges.

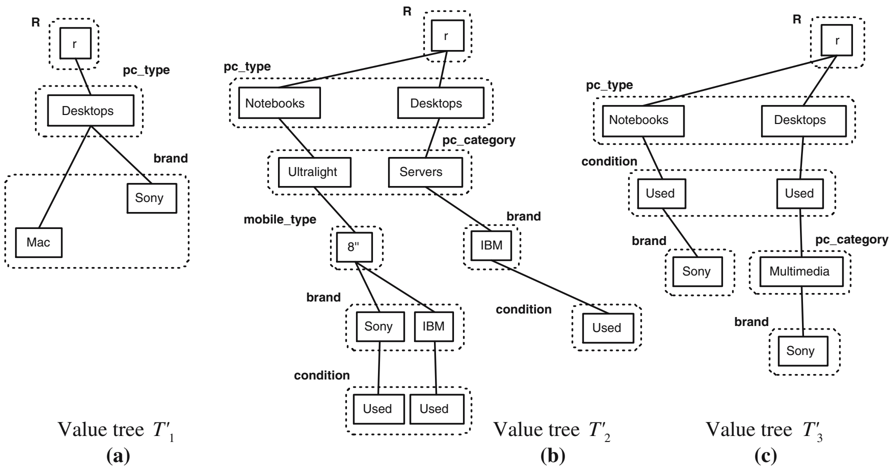


Fig. 5. Query Answers

Clearly, the answer of a query that does not involve any annotations or arrows (this is the query $(\emptyset, \emptyset)$) on a value tree T is T itself. Annotating a node with a "?" in a query is different than not annotating this node at all. In contrast to a non-annotated node, a node that is annotated with a "?" places a value of the corresponding dimension in every root-to-leaf path in the answer of the query.

Example 6. Consider the queries Q_1, Q_2 and Q_3 on the dimension graphs $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$, respectively, graphically shown in Figure 4. Consider also the value trees T_1, T_2 and T_3 of Figure 1. Figure 5 shows the answers T'_1, T'_2 and T'_3 of Q_1, Q_2 and Q_3 on T_1, T_2 and T_3, respectively.

Further, consider the query $Q_4 = (\mathcal{A}_4, \mathcal{P}_4)$, where $\mathcal{A}_4 = \{$ `pc_type = {Desktops}, brand = {HP,Gateway}`$\}$, and $\mathcal{P}_3 = \{$`pc_type` $\rightarrow$ `brand`$\}$ on the dimension graph $\mathcal{G}_1$ shown in Figure 3. In the value tree T_1 shown in Figure 1(a) there are values of dimension `brand` that are children of values of dimension `pc_type`. However, there is no root-to-leaf path that involves values `Desktops` and `HP`, or `Desktops` and `Gateway`. Therefore, the answer of Q_4 on T_1 is empty. □

4.3 Unsatisfiable Queries

A query on a dimension graph $\mathcal{G}$ is called *unsatisfiable* if its answer is empty on every value tree underlying $\mathcal{G}$. Otherwise, it is called *satisfiable.* Detecting the unsatisfiability of a query saves its evaluation on a value tree (which, in any case, produces an empty answer.) In general, this value tree is much larger than its dimension graph which might be needed for detecting the unsatisfiability of the query. The graphical representation of a query provides some intuition on unsatisfiable queries.

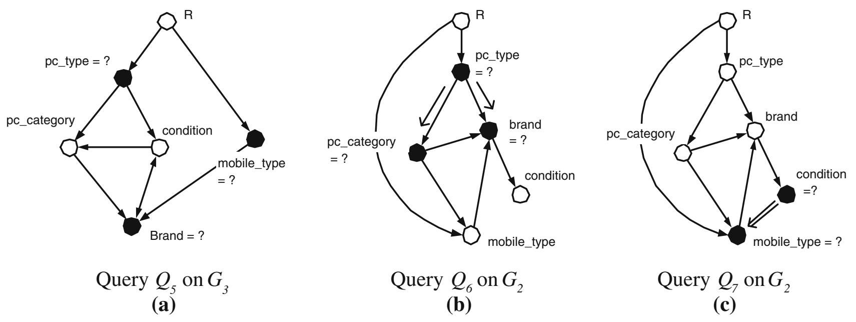


Fig. 6. Unsatisfiable Queries

Example 7. Consider the dimension graphs $\mathcal{G}_2$ and $\mathcal{G}_3$ of Figure 3, and the queries Q_5 on $\mathcal{G}_3$, and Q_6 and Q_7 on $\mathcal{G}_2$ graphically represented in Figure 6. These queries are unsatisfiable.

In query Q_5 of Figure 6(a), there is no path from the root of $\mathcal{G}_3$ that involves all the annotated nodes. By Proposition 3 there is no root-to-leaf path in a value tree underlying $\mathcal{G}_3$ that involves values for the annotated dimensions in Q_5.

In query Q_6 of Figure 6(b), there is a path from the root of $\mathcal{G}_2$ through all the annotated nodes (e.g. the path (R, `pc_type, pc_category, brand`)). However, there are two outgoing single arrows from the same node (node `pc_type`). Clearly, no two values can be children of the same node in a root-to-leaf path of a value tree underlying $\mathcal{G}_2$.

In query Q_7 of Figure 6(c), there is also a path from the root of $\mathcal{G}_2$ through all the annotated nodes (e.g. the path (R, `mobile_type, brand, condition`)). However, there is a double arrow from node `condition` to node `mobile_type` in Q_7 and no path from `condition` to `mobile_type` in $\mathcal{G}_2$. By Proposition 3 there is no root-to-leaf path in a value tree underlying $\mathcal{G}_2$ that involves a value for dimension `condition` preceding a value for dimension `mobile_type`. □

More generally, we can show the following result that provides sufficient conditions for a query to be unsatisfiable.

Proposition 4. A query Q on a dimension graph $\mathcal{G}$ is *unsatisfiable* if one of the following conditions holds:

(a) Arrows in Q form a directed cycle.
(b) There are precedence relationships $D \rightarrow D_i$ and $D \rightarrow D_j$ or precedence relationships $D_i \rightarrow D$ and $D_j \rightarrow D$ in Q ($D_i \neq D_j$).
(c) There is a precedence relationship $D_i \rightarrow D_j$ in Q but no edge from node D_i to node D_j in $\mathcal{G}$.
(d) There is a precedence relationship $D_i \Rightarrow D_j$ in Q but no edge from node D_i to node D_j in the *transitive closure* of $\mathcal{G}$ (in other words, no path from node D_i to node D_j in $\mathcal{G}$).
(e) The annotated nodes in Q are not on a path from the root of $\mathcal{G}$. □

Proof: Conditions (a), (b), (c) and (d) violate condition (e) of Definition 7, as follows:
(a) If there are arrows in Q that form a directed cycle, say $D_i \rightarrow D_j$ and $D_j \rightarrow D_i$, then a value of D_j should be the parent and the child of a value of D_i in the same path of the value tree, which is not possible since there could not be two nodes on a path in the value tree labeled by values that belong to the same dimension.
(b) If there are precedence relationships $D \rightarrow D_i$ and $D \rightarrow D_j$ in Q, then a value of D should be the parent of a value of D_i and a value of D_j in the same path of the value tree, which is not possible.
(c) If there is a precedence relationship $D_i \rightarrow D_j$ in Q but no edge from node D_i to node D_j in $\mathcal{G}$, then there is not any path with a value from D_i being the parent of a value from D_j in the value tree.
(d) If there is a precedence relationship $D_i \Rightarrow D_j$ in Q but no path from node D_i to node D_j in $\mathcal{G}$, then there is not any path with a value from D_i being the ancestor of a value from D_j in the value tree.
Condition (e) violates condition (a) of Definition 7: If the annotated nodes in Q are not on a path from the root of $\mathcal{G}$, then the resulting tree will not include value r as its root, and thus T' and T will not have the same root. □

In order to provide necessary conditions for query unsatisfiability, we introduce the concept of an answer path of a query.

Definition 8. *Let Q be a query on a dimension graph $\mathcal{G}$. An* answer path *of Q in $\mathcal{G}$ is a path p in $\mathcal{G}$ from the root of $\mathcal{G}$ such that:*
(a) All the annotated dimensions in Q are on p, and p ends on an annotated dimension of Q.
(b) If there is a precedence relationship $D_i \rightarrow D_j$ (resp. $D_i \Rightarrow D_j$) in Q, then D_j is a child (resp. descendent) of D_i in p. □

Example 8. Consider the query Q_2 on dimension graph $\mathcal{G}_2$ and the query Q_3 on dimension graph $\mathcal{G}_3$, which are shown in Figures 4(b) and 4(c), respectively. One can identify the following answer paths for query Q_2 in $\mathcal{G}_2$:

R, `pc_type, brand, condition`
R, `pc_type, pc_category, brand, condition`
R, `pc_type, pc_category, mobile_type, brand, condition`

The answer paths for query Q_3 in $\mathcal{G}_3$ are:

R, `pc_type, condition, brand`
R, `pc_type, condition, pc_category, brand`
R, `pc_type, pc_category, brand, condition` □

The following proposition provides necessary and sufficient conditions for a query to be unsatisfiable.

Proposition 5. A query Q on a dimension graph $\mathcal{G}$ is *unsatisfiable* iff there is no answer path of Q in $\mathcal{G}$. □

Proof: (If part) We show that if Q is satisfiable, there is an answer path of Q in $\mathcal{G}$. Assume that Q is satisfiable and let value tree T' be the answer of Q on a value tree T. Let $p' = r, v_1, \ldots, v_m$ be a root-to-leaf path in T'. By definition, p' is also a root-to-leaf path in T. Since p' satisfies the conditions of Definition 7, the path $p = R, D_1, \ldots, D_m$, where $v_i \in D_i,\ i = 1, \ldots, m$, satisfies the conditions of Definition 8. Therefore, p is an answer path of $\mathcal{G}$ in Q.

(Only if part) We show that if there is an answer path of Q in $\mathcal{G}$, Q is satisfiable. Let p be an answer path of Q in $\mathcal{G}$. Let T be the tree induced by a depth first traversal of $\mathcal{G}$ where every occurrence of a node D_i of $\mathcal{G}$ in T is labeled by a (the same) value v_i of dimension D_i. In particular, if D_i is a dimension in Q annotated by the set $\mathcal{A}_i$, $v_i \in \mathcal{A}_i$. If $p = R, D_1, \ldots, D_k$, there is a root-to-leaf path $p' = r, v_1, \ldots, v_m,\ k \leq m$, in T such that $v_i \in D_i,\ i = 1, \ldots, k$. Since p satisfies the conditions of Definition 8, p' satisfies the conditions of Definition 7. Therefore, p' is a root-to-leaf path in the answer of Q on T. Consequently, the answer of Q in T is non-empty. We conclude that Q is satisfiable. □

4.4 Query Evaluation

When evaluating a query, we first check it for satisfiability. If a query is satisfiable, we proceed to compute its answer on a value tree in three steps. In the first step, we compute all the answer paths of the query. In the second step, we generate path expressions based on the answer paths. In the third step we evaluate the path expressions on the value tree and compose the answer of the query.

To represent path expressions, we use a notation similar to that of XPath [4]. The fragment of XPath we use involves node names (v_i), child axis (/), descendant axis (//), wildcards ($*$), unions ($|$). The expression $(v_1|\ldots|v_m)$ represents any node name in the set $\{v_1, \ldots, v_m\}$. For a dimension D, we use the expression $*_D$ as an abbreviation for the expression $(v_1|\ldots|v_n)$, where $\{v_1, \ldots, v_n\} = D$.

Given an answer path, we construct a corresponding path expression as follows. Let $R, D_1, \ldots, D_k$ be an answer path of a query Q. The corresponding path expression has the form $r/\theta_1/\ldots/\theta_k$, where, for $i = 1, \ldots, k$,

$$\theta_i = \begin{cases} (v_1|\ldots|v_m) & \text{if } D_i \text{ is annotated with the value set } \{v_1, \ldots, v_m\} \\ *_{D_i} & \text{if } D_i \text{ is annotated with a "?" or if } D_i \text{ is not annotated} \end{cases}$$

Notice that even though nodes annotated with a "?" are treated the same way as non-annotated ones in the construction of path expressions for a query, they affect differently the answer of a query since they are taken into account in the identification of answer paths for that query.

Before showing what the result of a path expression on a value tree is, we introduce the concept of a merge of a set of value trees (or paths). Let $T_1, \ldots, T_k$ be a set of value trees having the same root r. The *merge* of $T_1, \ldots, T_k$, denoted $T_1 \cup \ldots \cup T_k$, is a minimal[2] value tree which has $T_1, \ldots, T_k$ as subtrees. It is not difficult to see that this value tree is unique.

[2] Minimality is meant with respect to the number of nodes or edges.

We show now what is the result of a path expression on a value tree. Let e be a path expression and T be a value tree. Let also P be the set of paths from the root of T to the leafs of T that satisfy e. The result $res(e, T)$ of a path expression e on a value tree T is the value tree $\bigcup_{p \in P} p$. Note that the result of a path expression is different than the result of the same XPath expression. The result of a path expression is a value tree while the result of the same XPath expression is a set of nodes [4]. We can use XQuery [5] to compute the result of a path expression as it is defined here.

The answer of a query on a value tree can be computed by merging the results of its path expressions on the value tree. Let $E = \{e_1, \ldots, e_n\}$ be the set of path expressions constructed from all the answer paths of a query Q. The answer of Q on a value tree T is the value tree $\bigcup_{i \in [1,n]} res(e_i, T)$.

Example 9. Consider the query Q_2 on dimension graph $\mathcal{G}_2$, which is shown in Figure 4(b). The answer paths for Q_2 in $\mathcal{G}_2$ are shown in Example 8. These answer paths generate the following path expressions:

$r/ *_{\texttt{pc\_type}} /$(Sony|IBM)/Used
$r/ *_{\texttt{pc\_type}} / *_{\texttt{pc\_category}} /$(Sony|IBM)/Used
$r/ *_{\texttt{pc\_type}} / *_{\texttt{pc\_category}} / *_{\texttt{mobile\_type}} /$(Sony|IBM)/Used

Evaluating these path expressions on the value tree T_2 of Figure 1(b), one can see that the result of the first path expression is an empty value tree. In contrast, the second path expression contributes one path, while the third one contributes two paths to the answer of Q_2 on T_2 (Figure 5(b)).

Consider also the query Q_3 on dimension graph $\mathcal{G}_3$, which is shown in Figure 4(c). The answer paths for Q_3 in $\mathcal{G}_3$ are shown in Example 8. They generate the following path expressions:

$r/ *_{\texttt{pc_type}} /$Used/(Sony|IBM)
$r/ *_{\texttt{pc_type}} /$Used$/ *_{\texttt{pc_category}} /$(Sony|IBM)
$r/ *_{\texttt{pc\_type}} / *_{\texttt{pc\_category}} /$(Sony|IBM)/Used

Of those path expressions, evaluating the third one on the value tree T_3 of Figure 1(c) results in an empty value tree. Only the first two contribute paths to the answer of Q_3 on T_3 (Figure 5(c)). □

5 Data Source Integration

There are two major approaches to integrating data sources: the materialized (or data warehousing) and the virtual (or mediated) [32]. In the materialized approach, data are extracted from the data sources, integrated, and stored in a repository (the data warehouse). User queries are addressed to the data warehouse and are evaluated locally. When the data sources change, the stored data need to be maintained. This is a drawback of the materialized approach for applications that require current data. In the virtual approach, a global (or mediated) logical schema is used purely for the purpose of user query formulation [20]. Mappings are established between the global schema and the schemas of

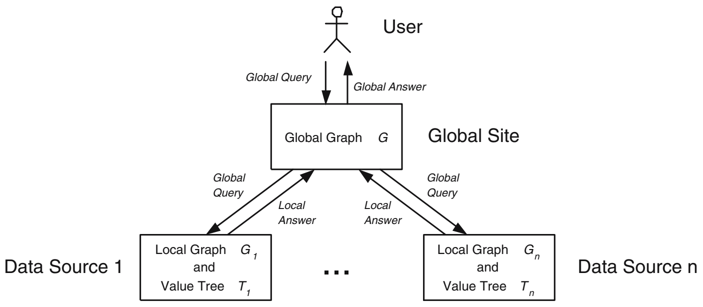


Fig. 7. A data integration system architecture

the data sources. User queries are posed on the global schema and they are transformed to queries on the data sources using these mappings. These later queries are sent to and evaluated at the data sources and their answers are sent back and combined for presentation to the user. The virtual approach guarantees that answer data are always current. Hybrid approaches combine features of the materialized and virtual approaches [21].

We follow a virtual approach adapted to the type of data structures we consider here. There are two key differences with respect to a traditional virtual approach:

(a) Since we are dealing with tree structures we do not have schemas. User queries are posed on a dimension graph constructed by integrating the dimension graphs of the value trees of the different data sources. We qualify this new dimension graph as *global* as opposed to the dimension graphs of the data sources which are characterized as *local*.
(b) Queries on the global dimension graph (*global queries*) are translated into queries on the dimension graphs of the data sources (*local queries*). However, a global query does not need any transformation to become a local query. It can either be defined on a local dimension graph or not.

Figure 7 shows the architecture of our data integration system. Each data source i contains a value tree T_i and its local dimension graph $\mathcal{G}_i$. Each value in T_i is mapped to a dimension. The global site contains the global dimension graph. We show how a global dimension graph is constructed in Section 5.1. Global queries are posed by the users on this dimension graph. They are formally defined in Section 5.2. Global queries are checked for satisfiability at the global site. They are potentially forwarded to the data sources where they are evaluated on the local dimension graphs and value trees, and their answers are sent back to the global site. The process of evaluating global queries is described in Section 5.3.

5.1 Global Dimension Graph Construction

The global dimension graph is constructed from the local dimension graphs.

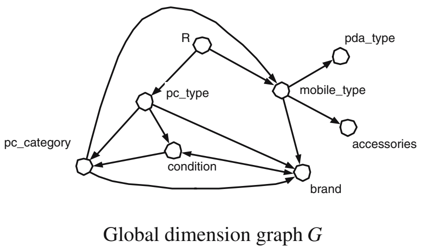


Fig. 8. The global dimension graph $\mathcal{G}$ for $\mathcal{G}_1, \mathcal{G}_2$ and $\mathcal{G}_3$

Definition 9. *Let* $\mathcal{G}_1 = (N_1, E_1)$*, ...,* $\mathcal{G}_n = (N_n, E_n)$ *be local dimension graphs of value trees over dimension set* $\mathcal{D}$*. A* global dimension graph $\mathcal{G}$ *for* $\mathcal{G}_1$*, ...,* $\mathcal{G}_n$ *is a dimension graph* (N, E)*, where* N *is a set of nodes and* E *is a set of edges defined as follows:*

(a) $N = N_1 \cup \ldots \cup N_n$.
(b) There is a directed edge in E *from node* D_i *to node* D_j *iff there is a directed edge from node* D_i *to node* D_j *in some local dimension graph.* □

Example 10. Consider the local dimension graphs $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$ of Figure 3. Figure 8 shows the global dimension graph $\mathcal{G}$ for $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$.

Notice that $\mathcal{G}$ has cycles (e.g. `pc_category`,`mobile_type`, `brand`, `condition`, `pc_category`) that do not appear in any of the local dimension graphs. □

5.2 Queries on Global Dimension Graphs

The syntax of global queries is identical to that of local queries (see Section 4.1). Global queries are posed on global graphs.

Example 11. Consider the global dimension graph $\mathcal{G}$ of Figure 8. Figure 9(a) shows a graphical representation of query Q_8 on $\mathcal{G}$. As with the representation of local queries, annotated nodes are shown with filled black circles. □

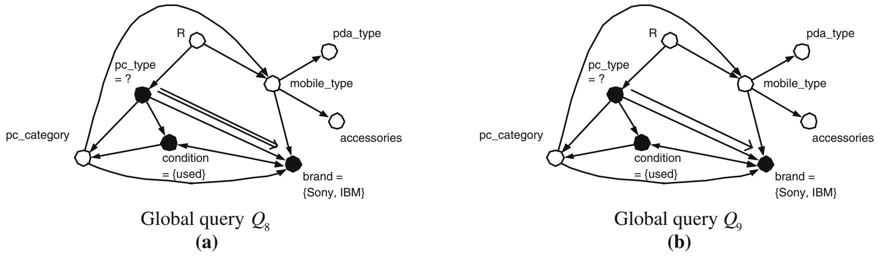


Fig. 9. Global queries Q_8 and Q_9

The answer of a global query is defined in terms of the answer of a local query.

Definition 10. *Let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be local dimension graphs of value trees $T_1, \ldots, T_n$ respectively over dimension set $\mathcal{D}$. Let also $\mathcal{G}$ be a global dimension graph for $\mathcal{G}_1, \ldots, \mathcal{G}_n$ and Q be a query on $\mathcal{G}$. The* answer *of Q on the list of value trees $T_1, \ldots, T_n$ is a list $T'_1, \ldots, T'_n$, where each T'_i, $i = 1, \ldots, n$, is defined as follows:*

1. *If there is an annotated dimension in Q that does not appear in $\mathcal{G}_i$, then T'_i is ϵ (empty answer). In this case, we say that the global query Q is not* definable *on the local dimension graph $\mathcal{G}_i$.*
2. *Otherwise, let Q_i be the query Q on the local dimension graph $\mathcal{G}_i$. T'_i is the answer of query Q_i on T_i.* □

Example 12. Consider the global query Q_8 of Figure 9(a) and the local value trees T_1, T_2 and T_3 of Figure 1. The local dimension graphs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ of the local value trees T_1, T_2 and T_3 are shown in Figure 3. Query Q_8 annotates dimension `condition` that does not appear in $\mathcal{G}_1$. All the annotated dimensions of Q_8 appear in $\mathcal{G}_2$ and $\mathcal{G}_3$. Query Q_8 on $\mathcal{G}_2$ is the local query Q_2 shown in Figure 4(b), while query Q_8 on $\mathcal{G}_3$ is the local query Q_3 shown in Figure 4(c). The answers of Q_2 and Q_3 on the local trees T_2 and T_3 underlying the local dimension graphs $\mathcal{G}_2$ and $\mathcal{G}_3$ are the value trees T'_2 and T'_3 shown in Figure 5(b) and 5(c) respectively. Therefore, the answer of Q on T_1, T_2, T_3 is ϵ, T'_2, T'_3. □

A (global) query on a global dimension graph $\mathcal{G}$ is called *unsatisfiable* if its answer is a list of empty answers on every list of value trees that have $\mathcal{G}$ as the global dimension graph of their local dimension graphs. Otherwise, it is called *satisfiable*. Clearly, the necessary and sufficient conditions provided by Proposition 5 for a local query to be unsatisfiable also hold for a global query.

5.3 Global Query Evaluation

Global query evaluation can be described in three phases. The first phase involves the global dimension graph, the second phase the local dimension graphs and the third phase the value trees.

In the first phase, the global query is checked for satisfiability at the global site. This check involves only the global query and the global dimension graph. If the query is satisfiable, it is sent to the data sources.

In the second phase, each data source checks if the global query is definable on its local dimension graph. If a global query is not definable at a data source, an empty answer is returned to the global site. Otherwise, it is checked for satisfiability as a local query. Notice that a query on a local graph may be unsatisfiable even if the same query on the global graph is satisfiable. The reason is that a local graph may be more restricted than the global graph: there may be paths or directed edges between two nodes in the global graph that do not exist between the same nodes on a local graph. Conditions (c), (d), and (e) of Proposition 4 show that the lack of these paths or edges may imply the unsatisfiability of the local query. If a local query is unsatisfiable, an empty answer is sent to the global site by the corresponding data source.

Example 13. Consider the global query Q_9 on the global dimension graph $\mathcal{G}$ graphically represented in Figure 9(b). This query involves an arrow from node `pc_type` to node `brand`. It is satisfiable since there is an edge in $\mathcal{G}$ from node `pc_type` to node `brand`. In contrast, the same query on the local dimension graph $\mathcal{G}_3$ of Figure 4(c) is unsatisfiable since there is no such an edge $\mathcal{G}_3$. □

In the third phase, satisfiable local queries are evaluated as described in Section 4, and the answers are sent to the global site for presentation to the user. Figure 10 outlines the different phases of the evaluation of a global query.

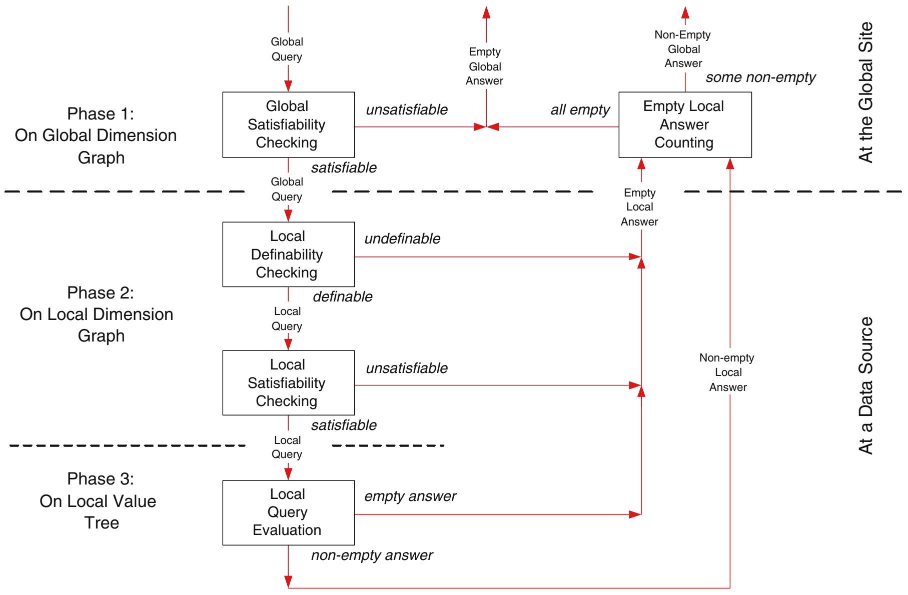


Fig. 10. Global Query Evaluation

6 Experimental Evaluation

We implemented and compared our integration strategy against one that does not exploit dimension graphs to query multiple tree-structured data sources. We briefly summarize the two integration strategies:

$A1$: Queries are formed on the global dimension graph. The evaluation of a query on a value tree is performed only if the query is satisfiable on the global dimension graph, as well as definable and satisfiable on the local dimension graph of this value tree. This is the approach suggested in this paper.

$A2$: Queries are formed directly using sets of path expressions (parent/child and ancestor/descendant relationships) that involve values from value trees. Given these sets, the system generates all the possible orderings of the values that respect the parent/child and ancestor/descendant relationships specified in the path expressions. Each one of these orderings corresponds to a single path expression to be evaluated on a value tree. This approach does not exploit dimension graphs for the evaluation of queries.

In order to maintain similar query sets for both approaches, our experimental platform transforms queries on the global dimension graph that involve dimensions into sets of simple path expressions to be matched by the same path of the value tree. Consider, for instance, the query $(\mathcal{A}, \mathcal{P})$, where $\mathcal{A} = \{\texttt{pc\_type} = \{\texttt{Notebooks}\}, \texttt{brand} = \{\texttt{Sony}, \texttt{IBM}\}, \texttt{condition} = \{\texttt{Used}\}\}$ and $\mathcal{P} = \{\texttt{pc\_type} \rightarrow \texttt{brand}\}$. The set of simple path expressions for approach $A2$ is $\{$`r//Notebooks/ (Sony|IBM)`, `r//Used`$\}$. The corresponding path expressions to be evaluated on the value tree are `r//Notebooks/(Sony|IBM)//Used` and `r//Used//Notebooks/(Sony|IBM)`.

We used a set of synthetic value trees, and we measured the execution time for evaluating queries on a global dimension graph. The set of value trees was constructed as follows. We generated a random set of 30 dimensions with 10 values each (a total of 300 distinct values), and we created 10 random value trees using those values. The actual number of values in the value trees ranges from 280 to 500. Queries were generated by randomly annotating dimensions in dimension graphs and adding arrows. In order to add arrows to the annotated dimensions of the query, the generator first creates a fully connected graph, involving only the annotated dimensions. Then, if n arrows need to be created, it removes arrows until n are left. The percentage of single arrows in the total number of arrows in the query is a system parameter and depends on the experiments.

6.1 Experiments and Results

We carried out three different types of experiments[3] to study the differences in the execution time of the two integration approaches. For every measure point in the x-axis, 10 queries were generated for each one of the 10 value trees. The recorded execution time per point is the average execution time.

Varying the Size of the Queries

Satisfiable and Unsatisfiable Queries. We measured the execution time varying the percentage of arrows (i.e. precedence relationships) for different numbers of annotated dimensions in the queries. The percentage of arrows is the ratio of the number of arrows to the total number of possible arrows in the query. Note that

[3] All the experiments were carried out on an AMD Sempron 2600 PC with 512MB RAM.

a percentage of arrows of 100% means that the arrows and the annotated dimensions of the query form a fully connected graph. In Figure 11, we present the results obtained for global queries having 2 to 8 annotated dimensions, varying the percentage of arrows. The y-axis is on a logarithmic scale. The number of dimensions is fixed to 30. In each query, 50% of the arrows were single (parent/child relationships) and 50% were double (ancestor/descendant relationships). In this particular experiment, 50% of queries executed were unsatisfiable. In any case, the approach $A1$ clearly outperforms $A2$.

For both approaches, as the percentage of arrows increases, the execution time drops. This is explained by the fact that, as the number of arrows increases, fewer path expressions are generated by both approaches to be matched on the value tree.

As the number of annotated dimensions increases, the execution time in approach $A1$ drops. This is expected, since for a fixed dimension graph, an increase in the number of annotated dimensions reduces the number of possible answer paths (recall that an answer path involves all the annotated dimensions). Therefore, the number of path expressions generated by approach $A1$ to match the values tree is reduced too.

In approach $A2$, as the number of annotated dimensions increases, the query execution time raises significantly for low arrow percentage, but the steepness of the fall of the curve raises too. The curve hits the x-axis closer to 0 as the number of annotations raises. This can be explained as follows. As the number of annotated dimensions increases, the number of possible value orderings increases exponentially. For a fixed set of arrows, this increase results in an increase on the number of path expressions generated. However, for a fixed percentage of arrows, the number of arrows increases too when the number of annotated dimensions increases. As we explained above, increasing the number of arrows reduces the number of path expressions generated. For a fixed percentage of arrows, after a certain threshold number of annotated dimensions, the increase in the number of arrows dominates and the number of generated path expressions drops.

Only Satisfiable Queries. We performed the previous experiment with only satisfiable queries. We observed that approach $A1$ outperforms approach $A2$ even in this case. Figure 12 shows, for example, the results obtained for queries having 3 and 4 annotated dimensions, varying the percentage of arrows.

Varying the Type of Arrows in the Queries

Satisfiable and Unsatisfiable Queries. We measured the execution time varying the percentage of single arrows in the total number of arrows in the global query for different pairs of numbers of annotated dimensions and arrows. In Figure 13, we present the results obtained for queries having (a) 6 annotated dimensions and 4 arrows, (b) 7 annotated dimensions and 4 arrows, and (c) 7 annotated dimensions and 5 arrows.

The higher the percentage of single arrows, the lower the number of possible orderings of values needed for $A2$. This is reflected in the diagram, since there

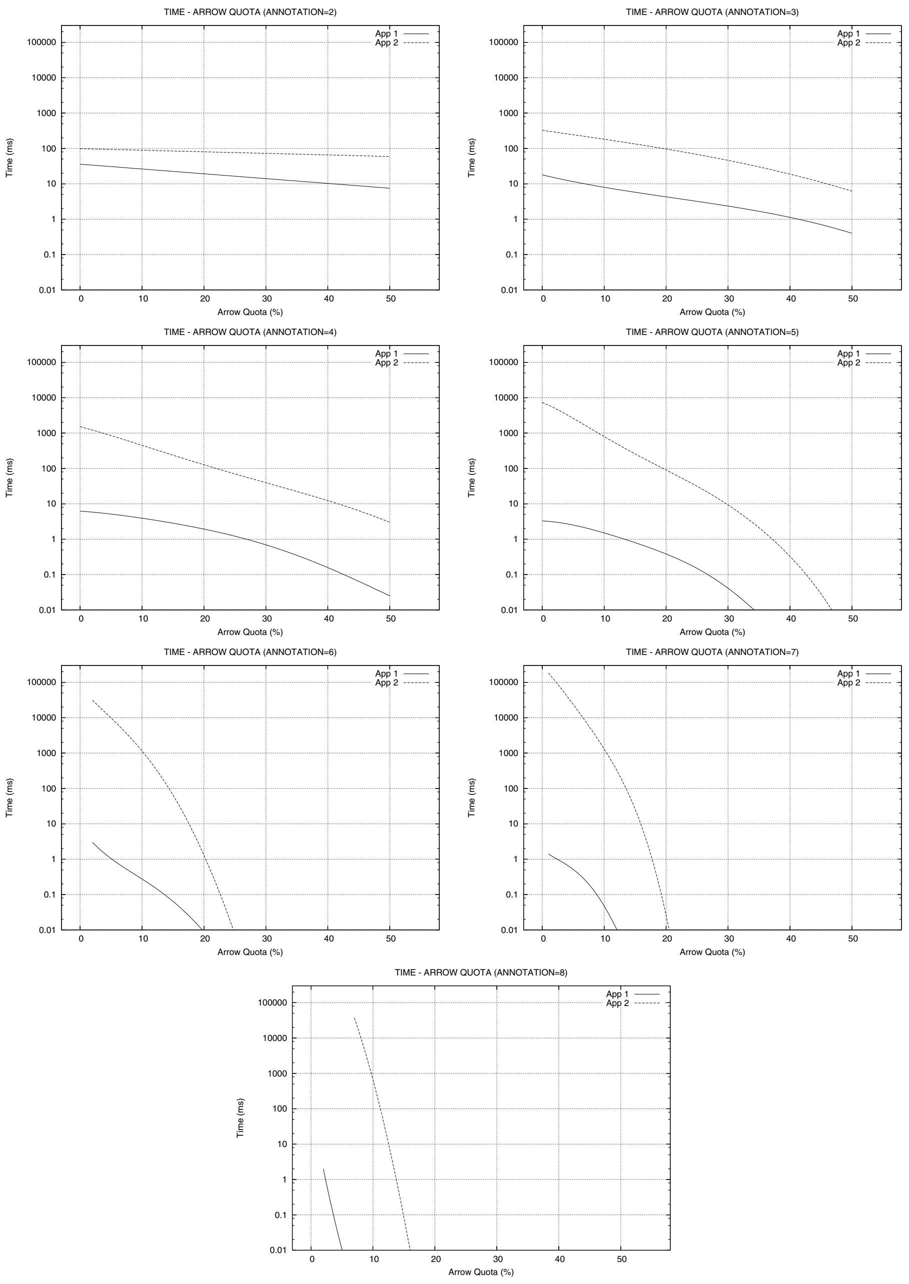


Fig. 11. Execution time varying the percentage of arrows for different numbers of annotated dimensions in the global query (50% of queries executed were unsatisfiable)

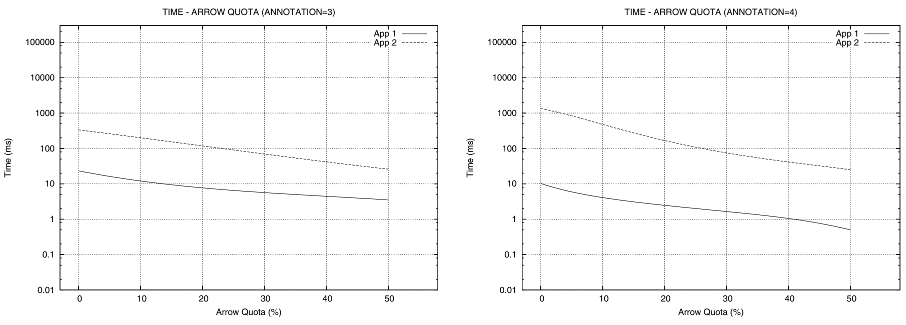


Fig. 12. Execution time varying the percentage of arrows for different numbers of annotated dimensions in the global query (all queries executed were satisfiable)

is a drop in the execution time as the percentage of single arrows increases for $A2$. For $A1$, higher percentage of single arrows means less answers paths. The reason is that the constraints imposed by single arrows are more restrictive than those of double arrows. This is also reflected in the diagram, since there is a drop in the execution time as the percentage of single arrows increases for $A1$. In any case, the approach $A1$ outperforms $A2$ since it is able to exploit the global dimension graph and the local dimension graphs to detect unsatisfiable queries, and to reduce the number of path expressions generated. The approach $A1$ outperforms $A2$ by three to four orders of magnitude.

Only Satisfiable Queries. We performed the previous experiment with only satisfiable queries. Even in this case approach $A1$ outperforms approach $A2$. Figure 14 shows for example the results obtained for queries having (a) 6 annotated dimensions and 4 arrows, and (b) 7 annotated dimensions and 5 arrows.

Varying the Size of Dimension Graphs

Satisfiable and Unsatisfiable Queries. We calculated the execution time for different sizes of global dimension graphs, varying the percentage of arrows. The number of annotated dimensions is fixed for different numbers of dimensions. In Figure 15, we present the results obtained for global dimension graphs having 30, 34, 38 and 42 dimensions. In each global query, the number of annotated dimensions was fixed to 6 and 50% of the arrows were single ones.

Before we discuss the results, we explain the way we increase the size of a global dimension graph. Starting from a fixed value tree, its partition and a set of annotated dimensions, we generate queries by randomly adding arrows between those annotated dimensions. At the next step, a dimension is randomly selected to be split, and produces two new dimensions. The arrows are re-assigned to the new dimensions that contain the annotated values. When the number of dimensions increases, their number of values per dimension decreases on the average. In general, this results in a sparser graph and reduces the number of

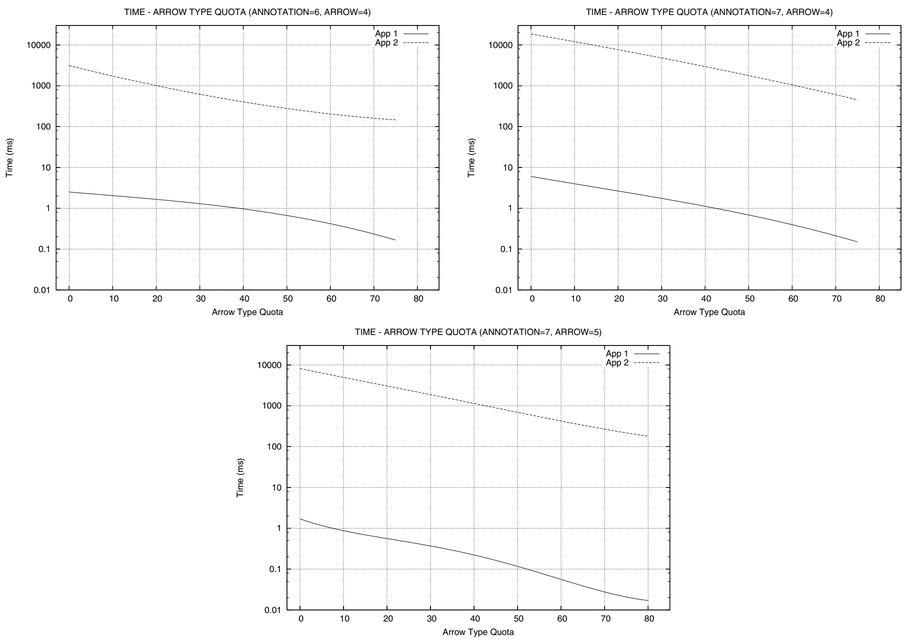


Fig. 13. Execution time varying the percentage of single arrows in the total number of arrows in the global query, for different pairs of numbers of annotated dimensions and arrows (50% of queries executed were unsatisfiable)

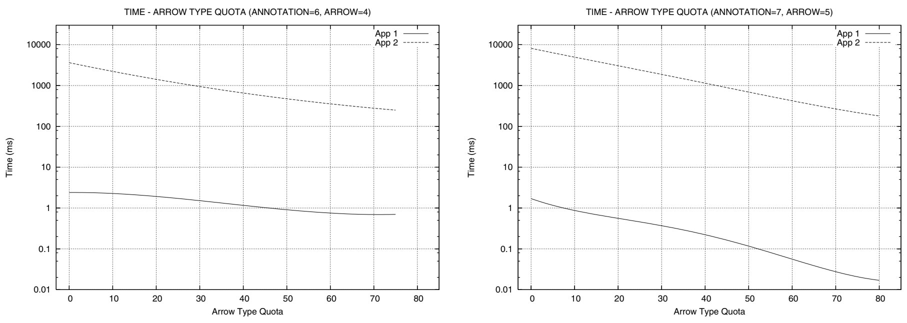


Fig. 14. Execution time varying the percentage of single arrows in the total number of arrows in the global query, for different pairs of numbers of annotated dimensions and arrows (all queries executed were satisfiable)

answer paths of a query. This is reflected in the diagram, since there is a drop in the execution time for $A1$ as the number of dimensions increases.

Note that in this experiment, the execution time of the approach $A2$ remains unaffected from the increase in the number of dimensions, since the queries do

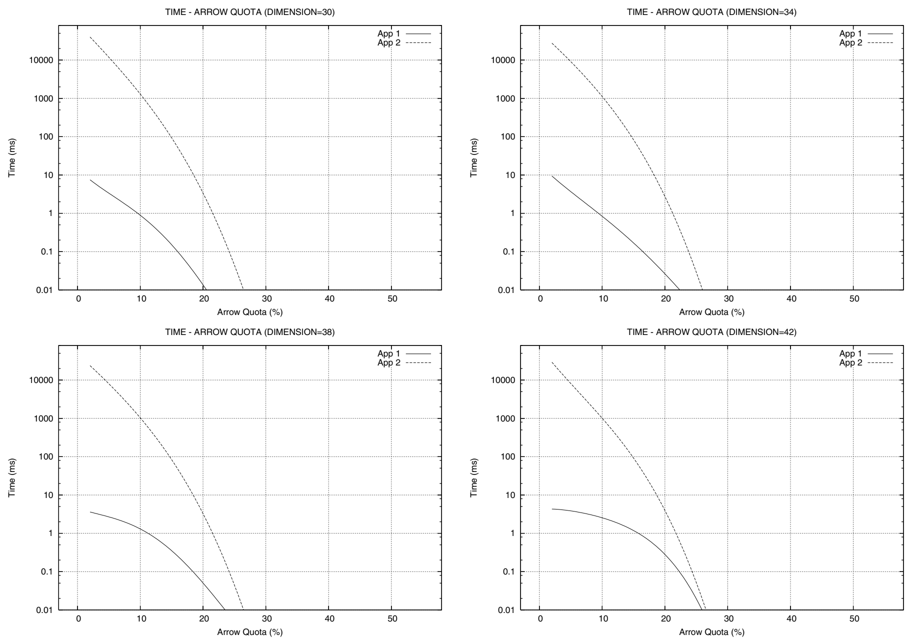


Fig. 15. Execution time varying the percentage of arrows for different numbers of dimensions in the global dimension graph (50% of queries executed were unsatisfiable)

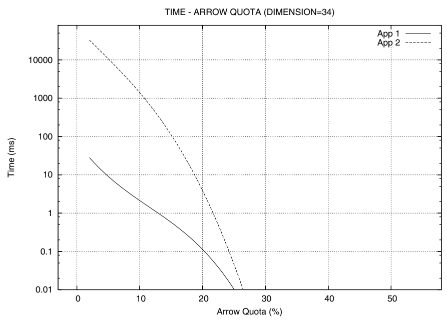


Fig. 16. Execution time for 34 dimensions in the global dimension graph (all queries executed were satisfiable)

not change and the approach $A2$ does not involve dimensions and the global dimension graph. The approach $A1$ clearly outperforms $A2$.

Only Satisfiable Queries. Approach $A1$ outperforms approach $A2$ even when the previous experiment is ran with only satisfiable queries. Figure 16 shows, for example, the results obtained for 34 dimensions.

7 Conclusions

We presented a method for integrating value trees that have structural differences and inconsistencies. Our approach exploits semantic information for the nodes of value trees. A semantic relationship between nodes in value trees was captured by the concept of a dimension. Dimension graphs were defined to capture structural information on the dimensions of a value tree. However, dimension graphs are not plain structural summaries of value trees, but rather semantically richer constructs that are able to support semantic integration of tree-structured data. We designed a query language to query value trees. Queries are specified on the dimensions of the value tree and can optionally involve parent-child and ancestor-descendant relationships between these dimensions. We provided necessary and sufficient conditions for query unsatisfiability and we presented a technique for evaluating satisfiable queries. We defined global dimension graphs by merging dimension graphs of local data sources. Queries are issued on the dimensions of a global dimension graph. Thus, in our approach for integrating data sources, a query is not restricted by the structure of a specific local value tree. We presented a technique to evaluate queries issued on global dimension graphs, first on the global site and then on local data sources. We conducted experiments to compare our integration method against one that does not exploit dimension graphs to query multiple tree-structured data sources. Our results demonstrated the clear superiority of our approach.

Our future work will focus on the problem of determining dimensions for the nodes of value trees. This is called dimensioning problem. Currently we assume that the assignment of nodes to dimensions is given. We are interested in studying how this process can be supported by the use of classification hierarchies and ontologies and how such a semi-automatic method would affect our approach.

References

1. Exchangeable Faceted Metadata Language, (XFML), 2003, http://www.xfml.org/.
2. XML Topic Maps (XTM), 2001, http://www.topicmaps.org.
3. World Wide Web Consortium site (W3C), http://www.w3c.org.
4. XML Path Language (XPath). World Wide Web Consortium site (W3C), 2003-2005, http://www.w3c.org/TR/xpath20/.
5. XML Query (XQuery). World Wide Web Consortium site (W3C), The Architecture Domain. 2003-2005, http://www.w3.org/XML/Query.
6. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, San Francisco, California, 2000.
7. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-based integration of XML web resources. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Sardinia, Italy, June 2002.
8. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *Proceedings of the 8th Conference on Extending Database Technology (EDBT'02)*, Prague, Czech Republic, Mar 2002.

9. R. Behrens. A grammar based model for XML schema integration. In *Proceedings of the 17th British National Conference on Databases (BNCOD'00)*, Exeter, UK, Jul 2000.
10. S. Bergamaschi, F. Guerra, and M. Vincini. A data integration framework for e-commerce product classification. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Sardinia, Italy, Jun 2002.
11. P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceeding of the 6th International Conference on Database Theory (ICDT'97)*, Delphi, Greece, Jan 1997.
12. S. D. Camillo, C. A. Heuser, and R. dos Santos Mello. Querying heterogeneous XML sources through a conceptual schema. In *Proceeding of the 22nd International Conference on Conceptual Modeling (ER'03)*, Chicago, IL, USA, Oct 2003.
13. A. B. Chaudhri, A. Rashid, and R. Zicari. *XML Data Management.* Addison Wesley, 2003.
14. V. Christophides, S. Cluet, and J. Simeon. On wrapping query languages and efficient XML integration. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'00)*, Dallas, Texas, USA, May 2000.
15. S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale XML repository. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, Rome, Italy, Sep 2001.
16. R. dos Santos Mello and C. A. Heuser. A bottom-up approach for integration of XML sources. In *Proceedings of the International Workshop on Information Integration on the Web (WIIW'01)*, Rio de Janeiro, Brazil, Apr 2001.
17. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from XML documents. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'00)*, Dallas, Texas, USA, May 2000.
18. R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, Aug 1997.
19. S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'02)*, Madison, USA, Jun 2002.
20. A. Halevy. Data integration: a status report. In *Proceedings of the Datenbanksysteme fur Business, Technologie und Web (BTW'03)*, 2003.
21. R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the 16th Symposium on Principles of Database Systems (ACM PODS'97)*, Tucson, Arizona, May 1997.
22. D. Kim, J. Kim, and S.-G. Lee. Catalog integration for electronic commerce through category-hierarchy merging technique. In *Proceedings of the 12th International Workshop on Research Issues in Data Engineering (RIDE'02)*, San Jose, USA, Mar 2002.
23. M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: Clustering XML schemas for effective integration. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02)*, McLean, Virginia, USA, Nov 2002.
24. M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the 21st Symposium on Principles of Database Systems (ACM PODS'02)*, Madison, Wisconsin, USA, Jun 2002.

25. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries over heterogeneous data sources. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, Rome, Italy, Sep 2001.
26. P. J. Marron, G. Lausen, and M. Weber. Catalog integration made easy. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03) (poster)*, Bangalore, India, Mar 2003.
27. N. Polyzotis and M. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD'02)*, Madison, USA, Jun 2002.
28. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334 – 350, 2001.
29. S. Ram and V. Ramesh. *Management of Heterogeneous and Autonomous Database Systems.* Morgan Kaufmann Publishers, 1999.
30. D. Theodoratos and T. Dalamagas. Querying tree-structured data using dimension graphs. In *Proceedings of 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, Porto, Portugal, Jun 2005.
31. Y. Tzitzikas, N. Spyratos, P. Constantopoulos, and A. Analyti. Extended faceted taxonomies for web catalogs. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE'02)*, Grand Hyatt, Singapore, Dec 2002.
32. J. Widom. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM'02)*, Baltimore, Maryland, USA, Dec 1995.

KDD Support Services Based on Data Semantics

Claudia Diamantini, Domenico Potena, and Maurizio Panti

Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione,
Università Politecnica delle Marche,
via Brecce Bianche, 60131 Ancona, Italy
{diamanti, d.potena, panti}@diiga.univpm.it

Abstract. The identification of valid, novel and interesting models from large volumes of data is the primary goal of Knowledge Discovery in Databases (KDD). In order to successfully achieve such a complex goal, many kinds of semantic information about the KDD and business domains is necessary. In this paper, we present an approach to the characterization of semantic domain information for a particular kind of KDD process: classification. In particular we show how, by estimating the properties of the true but unknown classification model, one can derive domain information on the classification problem at hand. We discuss how, by saving these properties with the data, users profit from this information and save time for experimenting with a lot of classifiers and parameters by accessing this knowledge.

Keywords: Data Mining, Data Semantics, Classification, Decision Border, User Support.

1 Introduction

Knowledge Discovery in Databases (KDD) emerged as a rapidly growing interdisciplinary field that merges together databases, statistics, machine learning and related areas in order to extract valuable information and knowledge in large volumes of data. KDD aims to overcome the limitations of traditional database queries, and of the more recent OLAP techniques, in order to support analysis and decision-making. These techniques can in fact help to extract information conforming to a predefined, previously known data model, but they do not allow us to identify novel, interesting models in data. For instance, these techniques cannot help in answering the following query: "Find network connection records indicating an intrusion", just because we do not have a model of what an intrusion is. However, even if our model of connection (e.g., the set of attributes 'connection length', 'protocol', 'service', 'number of failed login attempts', 'number of root accesses' and 'number of connections to the same host') does not explicitly contain a model of intrusion, we can assume that the latter can be established from the former, in terms of typical patterns representing relations among the basic model attributes. For instance, "If ('number of connections to the same host' ≥ 10) and ('protocol' = UDP) and ('service' = echo) and ('number of pending connections' = 'number of connections to the same host') Then (Prediction = DoS)" can be the model of one type of intrusion. This rationale underpins KDD, which studies techniques and methodologies to reveal unknown relations from available data. More formally, we define KDD as "the process of identifying valid, novel, potentially useful and ultimately understandable patterns/models in data", where data are defined as a set of facts F described by a database schema and patterns are defined as "an expression E in some language L describing facts in a subset F_E of F" [10, chap. 1]. Notice that, since patterns should be valid and potentially useful, an expression E should not limit itself to the description of the database instances, it should rather be capable of describing any new instance that could at any time be added to the database. In other words, the expression E describes the real phenomenon of which database instances are particular realizations. We often synthesize this by saying that KDD is a *model induction* activity. Different languages L define different kinds of models that can be induced. Models can be basically split into predictive ones (e.g., classification or regression models) and descriptive ones (e.g., clustering models or association rules). In the following we will consider classification models. Classification models give a description of a set of predefined classes, like e.g. the 'normal connection' and 'intrusion' classes, which are relevant for a given prediction or recognition task. Hence, in the same way a database schema defines the semantics of its instances, the schema of a classification model defines the semantics of the *classification problem*, that is of the set of classes chosen for the specific user goal (e.g., to detect intrusions).

Although a classification model is typically considered as the final result of a KDD process, it has yet another important role. As a matter of fact, to be effective, the model induction process must be guided by different kinds of domain information: information about how induction techniques work and how these can be applied in the specific business domain, the kinds of regularities one can expect to find in data and so on. In other words, we need to know the semantics of the KDD domain as well as the semantics of the business domain and how these interact. From this perspective, a classification model contains important business domain information that could be profitably used to guide the model induction process. In the next section, we will discuss the role of domain information in KDD in general, and in the classification task in particular, in deeper detail.

The contribution of this paper is an approach to characterize the semantics of classification problems, in terms of geometric properties of the classification model. The approach resorts on the relatively cheap Bayes risk weighted Vector Quantization (BVQ) learning algorithm [8] to define the analytic form of a good estimation of the true but unknown classification model. The analytical description is then used to derive some properties of the model that helps to choose probably the best classification method for the given problem, to appropriately prepare data and to train a classifier for the best classification method. By saving the analytical description of the classification model and its properties with the data, users profit from this information and save time in experimenting with a lot of classifiers and parameters by having access to this knowledge.

The rest of the paper is organized as follows. In section 2, we briefly introduce the KDD environment, highlighting the major sources of complexity in the design of a KDD process, and introducing the main approaches proposed in the literature to overcome this complexity by exploiting information about the KDD and business domains. We then narrow the discussion on the classification problem. In section 3 we introduce the definition of nearest neighbor Vector Quantizer (VQ) and its geometrical properties, and we show how to obtain the analytical description of a VQ. Then, in section 4 we describe our VQ-based learning approach to obtain an approximation of a classification model and its analytical description. Section 5 is devoted to the discussion on the possible types of deductions on the geometry of the classification model which is derived from the analytical definition, and how the accuracy of the approximation influences the validity of the discovered knowledge. In section 6 we empirically demonstrate the validity of the approach by means of real-world classification problems. Finally, in section 7 we cast a look on the possible application of the derived information regarding the classification model in the implementation of semantic KDD support services over a net. We will come to a conclusion in Section 8.

2 The Role of Semantics in KDD Process Design

KDD as a discipline studies the definition of reference models of a process of knowledge discovery from data. According to methodological standards [10,27], a process of discovery from data can be divided into six principal phases: Domain Understanding, Data Selection, Data Preprocessing, Transformation, Data Mining, Interpretation/Evaluation (see the schema in Figure 1). The core phase of the process is the *Data Mining* (DM) phase, where model induction is performed by one of the many available techniques. These techniques largely have a statistical foundation, and they require proper conditions of application. For this reason, the DM phase is preceded in any process model by a *data preparation*

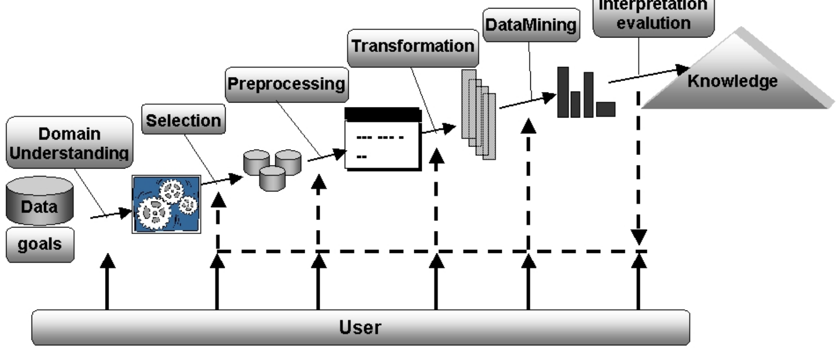


Fig. 1. The KDD process

phase, that can be further split in data and attribute (feature) selection, data preprocessing (e.g., the cleaning of data), data transformation (e.g., by normalization). Furthermore, it is known that, to be efficient and effective, statistical techniques should be guided by some model-driven hypothesis. So, the starting phase of the KDD process is devoted to domain and data understanding, where one has to find out the best representation of the business goal in terms of data mining goals, and envisage the best data structures and techniques to achieve that goal. In practice, this means drawing a rough model of the business and of the entities involved, to form some model-driven hypothesis on the kind of regularities which can be found in data and focus the search towards the most appropriate techniques. The ending phase is finally devoted to explanation and evaluation of the discovered knowledge inside the domain.

The intrinsic complexity of the design of a KDD process is due to the numerous degrees of freedom the user has to work with and to the goal-driven and domain dependent nature of the problem. When an analyst starts the discovery process, for example, she/he has to wonder: what are the database instances to select? How can I discriminate the noisy data from the informative one? Are all the data attributes equally important? Which is the best technique and algorithm to apply, and how the choice of the algorithm influences previous choices? How should one set the algorithm's parameters? Of course, the answer to these questions strictly depends on the problem and data at hand. On the other hand, the existence of a great amount of techniques and tools increases the complexity of choice, as it presupposes a certain degree of acquaintance with the mathematical theory underlying most of the techniques, so that they can be appropriately applied all together to the problem at hand, correctly and effectively used. Then, in order to design a KDD process, two kinds of expertise are needed: in the business domain and in the KDD domain. However, the user is typically a domain expert, but not a KDD expert, or viceversa. Another source of complexity is due to the intrinsic features of any discovery process, namely the lack of knowledge and consequently the difficulty to define the best plan to discover that knowledge beforehand. This fact is recognized in all the existing process models by accounting for the need of repeated backtracking to previous phases and repetition of certain actions: the knowledge acquired during a phase can suggest a revision of the choices taken at previous steps to enhance their results.

If no support is given to the user, then a blind search over the joint space of possible tasks, techniques, algorithms, parameters produces inevitably unsatisfactory results with great effort. Basically, the kind of support that a user can be given involves the following facilities:

- to understand the business domain and goal and to relate it to a suitable set of KDD tasks;
- to choose the more suitable tools for the user goals and business domain, on the basis of a number of characteristics:
  - performance (complexity, scalability, accuracy),
  - the kind of data they can be used for (textual/symbolic data, numerical data, structured data, sequences, ...),

  - the kind of goal they are written for (data cleaning, data transformation, data mining, visualization, ...),
  - the kind of data mining task (classification, rule induction, ...);

  this involves facilities to browse the tool repository and to obtain information about the tools;
- to set algorithm parameters, especially for Data Mining algorithms, in the appropriate manner with respect to the problem at hand;
- to manage all kinds of data involved in the KDD process, namely, raw and structured data, intermediate data and models, in terms of access, selection, preparation of data conforming to the tool input format, and so on;
- to design the KDD process by tool composition;

Most of the previous features rely on different forms of semantic information about data, tools and business domains. In the literature, the use of domain ontologies is largely proposed to guide the KDD process and to give support to domain experts. In [18, chap.23] and [25] a business domain ontology supports the extraction of novel features, by exploiting relations among domain concepts. In [16,32] the use of ontologies is proposed to refine the induced knowledge and to correctly interpret the results. [5] discussed the use of ontologies in the whole KDD process for the medical domain. Finally, in distributed environments, it has to be pointed out the role of business domain ontologies both in the search for appropriate data and in their integration [18, chap.23], [31]. A special kind of domain ontology is the KDD ontology. A KDD ontology is a conceptualization of the KDD domain in terms of tasks, techniques, algorithms, tools and tool properties like performance and the kind of data that can be used for [3,20,33]. As such, a KDD ontology has a similar role with respect to the business domain ontology: it helps the business expert to understand the KDD domain, so that he can either effectively collaborate with a KDD expert in the design of a KDD project, or design the KDD project on his own. In this case, it can support the user in browsing a tool repository organized with respect to the KDD ontology. Semantic description of tools is also adopted to support standardization and process design by tool composition [11,14]. However, to support the proper choice and use of tools, the semantics of the problem is also needed. For instance, in the classification task, it is known that different classification techniques work better on certain classes of classification problems than others. Hence, the classification model can be exploited as a fundamental domain information to semantically guide the KDD process design. This principle is the basis of *Meta-learning*, that refers to a bulk of techniques to discover domain semantics hidden in data, which is then used to guide the choice of Data Mining algorithms [2,17] or to form a prototype domain model that guides further investigations [30]. In the following subsection we examine this topic thoroughly, for the specific case of the classification task, by giving a more formal definition of the nature of a classification problem and discussing how it can support the user in dealing with each phase of a KDD classification process. At the end of the paper, we will also discuss the advantages of equipping data published over a net with this kind of domain semantics, in order to build case-based support services.

2.1 Evidence-Based Classification Process Design

Let us start with a formal definition of the classification problem:

Definition 1. *Given a set $O_s = \{o_1, \ldots, o_m\}$ of observation n-tuples and a set of classes $C = \{c_1, \ldots, c_k\}$, the classification problem is to define a* classification rule*, that is a mapping $\Phi : O_s \rightarrow \mathcal{C}$, where each n-tuple is assigned to a class. A class c_j contains precisely those tuples mapped to it; that is $c_j = \{o_i | \Phi(o_i) = c_j, 1 \leq i \leq m, o_i \in O_s\}$.*

Note that classes are predefined, non overlapping and they partition the entire set of n-tuples. In this sense, the classes of a classification problem are indeed *equivalence classes.*

In order to model and characterize the properties of a classification problem, we can take a geometric point of view. In a geometric model the observation n-tuples are represented as points in a $\mathcal{R}^n$ vector space. In this way, the mapping Φ defines a partition of the vector space, where each partition region defines an equivalence class. These regions are called *decision regions*, while the border between decision regions is called the *decision border* (see Figure 2).

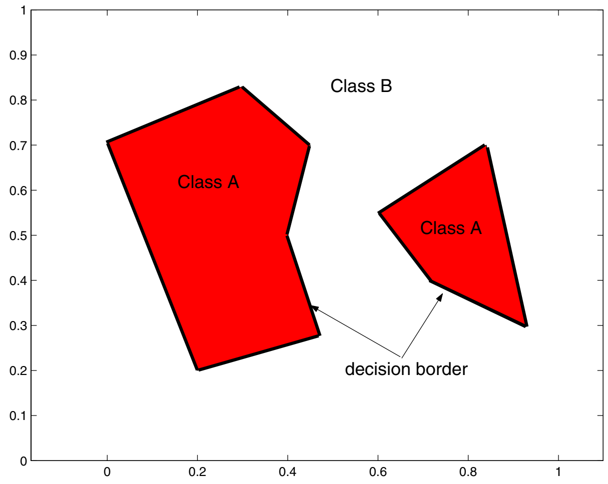


Fig. 2. A two dimensional vector space, with decision regions and the decision border for a two-class problem

The geometric properties of the decision regions have their natural interpretation as properties of the true classes or, in other terms, classification problems can be characterized in terms of the properties of the decision regions. For instance, we speak of linearly separable classification problem, when it is possible to separate classes without error by a hyperplane. On the other hand, when the decision border is a generic (possibly non connected) curve, then we speak of non linearly separable classification problems.

Let us illustrate how the decision regions (or, equivalently, the decision border) define an evidence that can guide the choices during the whole classification process. To start with, it is straightforward to observe that a general (and generic) knowledge of the shape of the decision regions and their localization in the vector space can be very useful in the preliminary data analysis phase. As a matter of fact, one of the most useful techniques adopted during this phase is visualization, since, it is said that, the most powerful data miner tool is the human eye. Unfortunately, visualization techniques can be directly applied only in problems of low dimensionality, say from one to three dimensions. The analytical form of the decision border could give important geometric information about the problem that cannot be visually inspected.

A particular form of data selection is called *data reduction* in the literature. It is a technique exploited in combination with non scalable algorithms, that is algorithms whose high cost make them unsuitable for the management of large amounts of data. Data reduction in classification problems can be performed by eliminating those samples falling far from the decision border, as less informative samples. This intuition dates back to the earliest work in Pattern Recognition, forming the basis of the condensed nearest neighbor method [15].

The decision border can support feature selection: in [23], a feature selection algorithm is shown which is based on the evidence that, following a direction not parallel to the decision border, the classification changes. Then, the direction normal to the decision border is the most informative one. By considering the principal components of the decision border, one can decide suitable transformations of data and the elimination of the less informative features. Unfortunately, the paper illustrates a method to derive the principal components that is computationally heavy and does not scale well.

The selection of the classifier architecture and the learning algorithm is based on the geometry of the decision border in the data mining step. For instance it is known that, if the border turns out to be parallel to the axes, then one can decide to use decision trees, that perform well on these problems and that have the advantage of a simple rule extraction. Other kinds of linear borders can suggest the use of Support Vector Machines (SVM) with linear kernel. Similarly, closed and convex decision regions would turn the choice towards SVM with gaussian kernel or Radial Basis Function (RBF) networks, while for open, non linear decision borders SVM with polynomial kernel or Multi-Layer Perceptron (MLP) would be preferable. To set the number of layers in MLP it is useful to know the type of concavity of the decision regions as well as the number of disconnected regions. One could even envisage a combined method where different types of architectures are used in different regions of the vector space, depending on the form of the border in that region. Finally, the initial state of a learning algorithm can be set in the regions of space near the decision border.

In the elicitation of the classification rule, the analytical representation of the decision border finds its natural application. It is known that one of the perceived limits of inductive techniques such as neural networks is their "black-box" nature: the classification rule is hidden in the structure of the network and a human expert has no element to validate it. In the literature, different techniques have been proposed to extract rules from MLP neural networks and decision trees. The method described in section 4 defines the basis for the elicitation of VQ-based classification rules.

3 Generalities on Nearest Neighbor Vector Quantizers

Definition 2. *A nearest neighbor Vector Quantizer (VQ) of dimension n and order M is a function $\Omega : \mathcal{R}^n \to \mathcal{M}$, $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}, m_i \in \mathcal{R}^n, m_i \neq m_j$, which defines a partition of $\mathcal{R}^n$ into M regions $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_M$, such that*

$$\mathcal{V}_i = \{x \in \mathcal{R}^n : d(x, m_i) < d(x, m_j),\ j \neq i\}, \tag{1}$$

where d is some distance measure.

$\mathcal{M}$ is called the *code*. It is a finite set of vectors in $\mathcal{R}^n$ called *code vectors* or *reference vectors*. The region $\mathcal{V}_i$ defined by (1) is called the *Voronoi region* of the code vector m_i.

Notice that, once the distance has been defined, $\mathcal{M}$ defines entirely the mapping Ω. If we choose as distance measure the usual squared Euclidean distance

$$d(x, y) = \| x - y \|^2 = (x - y)^T (x - y),\ x, y \in \mathcal{R}^n, \tag{2}$$

then it is particularly simple to describe the partition as a function of code vectors. In practice, we can reduce the definition of the Voronoi region $\mathcal{V}_i$ to the following system of constraints:

$$\mathcal{V}_i : \begin{cases} \| x - m_i \|^2 < \| x - m_{ci_1} \|^2 \\ \vdots \\ \| x - m_i \|^2 < \| x - m_{ci_l} \|^2, \end{cases} \tag{3}$$

where $Neigh(m_i) = \{m_{ci_1}, \ldots, m_{ci_l}\} \subset \mathcal{M}$ is the set of nearest code vectors to m_i. Region borders are defined as the geometric locus of points equidistant from at least a pair of code vectors. In particular, the region border is a piecewise linear surface, where each piece of hyperplane (excluding the extreme points) satisfies with the equal sign exactly one of the constraints in (3). Thus, if l is the number of constraints, $\mathcal{V}_i$ is a polytope with l faces. Figure 3 gives an illustrative example of a code of order 10 and of the relative Voronoi diagram in $\mathcal{R}^2$.

Points satisfying two or more constraints with the equal sign define the vertices of the polytope $\mathcal{V}_i$. Finally, let us notice that each constraint in (3) defines a half-space of the type

$$V_{i,j} = \{x \in \mathcal{R}^n : u_{ij}^T \cdot (x - \beta_{ij}) \geq 0\},$$

with $u_{ij} = m_i - m_j$ and $\beta_{ij} = \frac{m_j + m_i}{2}$. Each region $\mathcal{V}_i$ is thus defined by the intersection of a finite number of half-spaces, hence it is a regular, convex polytope. It is also simple to see that each code vector m_i belongs to the region $\mathcal{V}_i$.

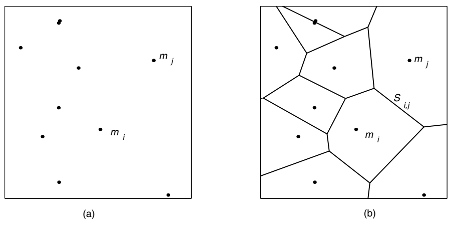
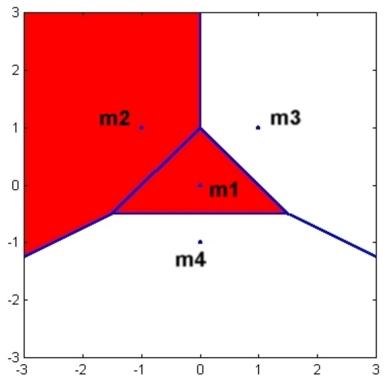


Fig. 3. (a) A code and (b) its Voronoi diagram. $\mathcal{S}_i, j$ is the border between m_i and m_j.

In fact, it holds $\| m_i - m_i \|^2 < \| m_i - m_j \|^2$ for each code vector $m_j \neq m_i$, then $m_i \in \mathcal{V}_i$. Regular VQs of this type are called polytopal VQs.

3.1 Voronoi Diagrams in n-Dimensional Space

In spite of the simple and well known theory illustrated above, few or no software for the practical calculus of Voronoi diagrams *in spaces of general dimensionality* $\mathcal{R}^n$, $n > 2$ exists (actually, the only one we found, which only approaches our needs is reported in [7]).



```
The Region of code vector m1 (0.000,0.000) label 2:
1.000*X(1) + -1.000*X(2) = -1.000 from (-0.000,1.000) to (-1.500,-0.500)
1.000*X(1) + 1.000*X(2) = 1.000 from (-0.000,1.000) to (1.500,-0.500)
0.000*X(1) + 1.000*X(2) = -0.500 from (-1.500,-0.500) to (1.500,-0.500)
The Region of code vector m2 (-1.000,1.000) label 2:
1.000*X(1) + -1.000*X(2) = -1.000 from (-0.000,1.000) to (-1.500,-0.500)
1.000*X(1) + 0.000*X(2) = -0.000 from (-0.000,1.000) to (-0.000,1001.000)
1.000*X(1) + -2.001*X(2) = -0.499 from (-1.500,-0.500) to (-667.059,-333.274)
The Region of code vector m3 (1.000,1.000) label 1:
1.000*X(1) + 1.000*X(2) = 1.000 from (-0.000,1.000) to (1.500,-0.500)
1.000*X(1) + 0.000*X(2) = -0.000 from (-0.000,1.000) to (-0.000,1001.000)
1.000*X(1) + 2.001*X(2) = 0.499 from (1.500,-0.500) to (667.059,-333.274)
The Region of code vector m4 (0.000,-1.000) label 1:
0.000*X(1) + 1.000*X(2) = -0.500 from (-1.500,-0.500) to (1.500,-0.500)
1.000*X(1) + -2.001*X(2) = -0.499 from (-1.500,-0.500) to (-667.059,-333.274)
1.000*X(1) + 2.001*X(2) = 0.499 from (1.500,-0.500) to (667.059,-333.274)
```

Fig. 4. An example of analytical description of a Voronoi Diagram and its graphical representation

We implemented such an algorithm, that is available as a web service at *http://babbage.diiga.univpm.it:8080/axis/services/voronoiWrapped*. The WSDL of the service can be downloaded at *http://babbage.diiga.univpm.it:8080/axis/-WSDL/voronoiWSDL.xml*. Definitions and technical details are given in Appendix A.

In Figure 4 a graphical example of a Voronoi diagram in $\mathcal{R}^2$ and the corresponding output of the algorithm is given . For each code vector m_i the equations of the pieces of lines and their extremes is reported (the points in the $from \ldots to \ldots$ expression). $X(i)$ represents the i-th dimension of the feature space. Notice the existence of extremes with very big values not comparable to values of the code vectors. These represent the approximation of "points at infinite" that are introduced for computational purposes and which are called *fictitious code vectors* in the Appendix. Notice that the use of fictitious code vectors introduces approximation errors in some equation (e.g., the 3rd equation of m2 should be X(1)-2*X(2)=-0.5).

4 An Approach to Decision Border Characterization

Statistical pattern classification is modeled by considering a pair $(\mathbf{x}, \mathbf{c})$ of random variables with values in $\mathcal{R}^n \times \mathcal{C}$. The continuous vector $\mathbf{x}$ is the *observed* vector (or *feature* vector), while the discrete random variable $\mathbf{c} \in \mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ is the *class* the observed vector belongs to. Each class c_i is characterized by a conditional density function $p_{\mathbf{x}|\mathbf{c}}(\mathbf{x} = x|\mathbf{c} = c_i)$, and by an *apriori probability* $P_{\mathbf{c}}(c_i)$, $\sum_{i=1}^{k} P_{\mathbf{c}}(c_i) = 1$. The best theoretical rule to assign a feature vector to a class is known as the *Bayes rule*. It reads:

$$c^* = argmax_{c_i}\{P_{\mathbf{c}}(c_i) * p_{\mathbf{x}|\mathbf{c}}(\mathbf{x} = x|\mathbf{c} = c_i)\}.$$

This rule produces in fact the minimum misclassification rate. For this reason, decision borders defined by the Bayes rule are considered the *true* (Bayes) decision border for the problem.

The form of the true decision border is generally unknown, since the Bayes rule is based on the definition of the unknown class conditional distributions. In the following we describe a method to estimate the form of the true but unknown decision border. It relies on the Labeled Vector Quantizer (LVQ) architecture as a classification architecture and on the BVQ algorithm [8] to design an LVQ approaching the Bayes rule.

Definition 3. *A Labeled Vector Quantizer (LVQ) is a pair $LVQ = <\Omega, \mathcal{L}>$, where $\Omega : \mathcal{R}^n \rightarrow \mathcal{M}$ is a vector quantizer, and $\mathcal{L} : \mathcal{M} \rightarrow \mathcal{C}$ is a labeling function, assigning to each code vector in $\mathcal{M}$ a class label.*

An LVQ defines a decision rule:

Definition 4. *The decision rule associated with a Labeled Vector Quantizer $LVQ = <\Omega, \mathcal{L}>$ is:*

$$\Phi_{LVQ} : \mathcal{R}^n \rightarrow \mathcal{C}, x \mapsto \mathcal{L}(\Omega(x)).$$

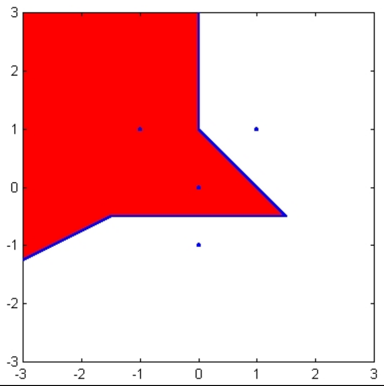
Notice the nearest neighbor nature of this decision rule: each vector in $\mathcal{R}^n$ is assigned to the same class as its nearest code vector. Thus, decision regions are defined by the union of Voronoi regions of code vectors with the same label. Notice also that decision borders are defined only by those hyperplanes $\mathcal{S}_{i,j}$ such that m_i and m_j have different labels.

The design of an LVQ decision rule approaching the Bayes rule is practically realized by supervised inductive learning algorithms, based on a set of examples of known class. Among the learning algorithms for LVQ classifier design, BVQ turns out to be the one with the best overall performances [8].

In the following we summarize the steps of the proposed method to extract the analytical description of the decision border. The method is valid, and we tested it, in general $\mathcal{R}^n$ space and for k-class problems. However in the following, for sake of simplicity and to give a visual support to the reader, the examples are given considering two-class problems in a two-dimensional space.

Let $T = \{(x_1, l_1), \ldots, (x_N, l_N)\}$ be a set of N labeled samples, where x_i is the feature vector and $l_i \in \{c_1, \ldots, c_k\}$ is its class.

1. **BVQ Training:** Use the N samples to train an LVQ. In this phase we have to set up the parameters of the LVQ and BVQ algorithm, that are principally the number of code vectors, the width of the windows Δ, the learning rate γ and the number of iterations. The tuning of the BVQ parameters is usually done experimentally, by trying different n-tuples of parameter values and evaluating the classification error for each. The total time and effort we put into the parameter setting depends on the quality of the classifier that we want to design and, of course, on the difficulty of the classification problem. The output of this step is the set of code vectors used to approach the Bayes border.



```
Decision Border:
1.000*X(1) + 0.000*X(2) = 0.000 from (-0.000,1.000) to (-0.000,1001.000)
1.000*X(1) + 1.000*X(2) = 1.000 from (-0.000,1.000) to (1.500,-0.500)
0.000*X(1) + 1.000*X(2) = -0.500 from (-1.500,-0.500) to (1.500,-0.500)
1.000*X(1) + -2.001*X(2) = -0.499 from (-1.500,-0.500) to (-667.059,-333.274)
```

Fig. 5. An example of Decision Border equations

2. **Analytical Voronoi Description:** Apply the algorithm described in Section 3.1 and Appendix A to the trained code vectors, to obtain the equations of hyperplanes and circumcenters representing the Voronoi border surfaces and their vertices respectively.
3. **Decision border extraction:** Starting from the equations of the Voronoi diagram, obtain the decision border description by deleting all the borders dividing two regions with the same label. This can be done in practice by merging the pieces of hyperplane of all the code vectors with the same label, deleting those appearing twice. The result of this step for the example in Figure 4 is reported in Figure 5.

5 Analysis

Having the analytical definition of decision borders, one can develop any kind of geometrical analysis. We hasten to point out that the results of the analysis depend on the quality of the classifier w.r.t. (1) correctness and (2) simplicity. Correctness is related to the accuracy in Bayes decision border approximation. We will show that valid analysis can be carried out, even if the classifier is not accurately designed. More refined analysis cannot be guaranteed to be valid unless the classifier is near-optimal. Simplicity of the classifier depends on the number of constraints used to describe the decision border, and hence indirectly on the number of code vectors. Of course, following the Occam's razor principle, such number should not be greater than the minimum number of constraints necessary to guarantee a given level of correctness, since each constraint beyond this number contributes reducing the human understanding of the model. In the following we will consider different types of analysis drawing attention to these issues.

5.1 Topological Properties of Regions

One simple analysis is that on the qualitative shape of the curve, that is of topological properties such as the Connected/Disconnected and the Open/Close properties of the decision regions. This information is derivable simply from the analysis of vertices. A decision region is Connected (Disconnected) if, denoted by V any of its vertices, it is (not) possible, starting from V, to pass along all the other vertices, moving along the pieces of hyperplanes of the region surface. To evaluate the connected property of a decision region we can use any algorithm to evaluate the connection of a graph. To evaluate the open/close property of the region (or of its sub-regions, if it is disconnected), it is sufficient to evaluate if a fictitious vertex exists in the set of its vertices. Since this vertex does not really exist, the region turns out to be unlimited (for instance, Figure 10 shows an example of two closed decision regions, while Figure 7 shows an example of open decision region). We illustrate the method for extracting the open and connected properties by the following simple algorithm:

let S be the list of all vertices V_i, $i \in \{1, \ldots, M\}$ of the decision regions. With a little abuse of notation, $S[i]$ will denote the i-th element of the list S, while $|S|$ will denote the number of elements in S. Let A be the matrix of links between any pair of vertices, such that $A[i, j] = 1$ iff $S[i]$ and $S[j]$ both belong to a piece of hyperplane delimiting the decision region and $i \neq j$. Finally, let C be an array such that $C[t]$ contains the list of vertices of the region analysed at the t-th iteration.

```
1. $t = 0$;
2. $t = t + 1$;
3. Set $X = S[1]$ and set $l[t] =$'close';
4. $i = 0$;
5. While (($i < |S|$) and ($l[t] =$'close')) do
   (a) $i = i + 1$;
   (b) if $S[i]$ is a fictitious vertex then $X = S[i]$ and $l[t] =$'open';
6. if ($X \notin C[t]$) then insert $X$ in $C[t]$;
7. $j = 1$;
8. While ($j \leq |S|$) do
   (a) if $A[i, j] = 1$ then do
         i. $A[i, j] = 0$ and $A[j, i] = 0$;
        ii. $X = S[j]$;
       iii. $i = j$ and $j = 1$;
        iv. if ($X \notin C[t]$) then insert $X$ in $C[t]$;
   (b) else $j = j + 1$;
9. delete from $S$ vertices in $C[t]$;
10. If $S$ is not empty go to step 2;
```

If $t = 1$ the decision region is connected, otherwise it is disconnected and formed by t sub-regions. The pairs $(C[i],\ l[i])$, with $i = 1, \ldots, t$ individuate the t sub-regions and if they are closed or open. For the example shown in Figure 5 the input to the algorithm can be

$$\begin{array}{l} S[1] = (-0.000, 1.000), \\ S[2] = (-0.000, 1001.000), \\ S[3] = (1.500, -0.500), \\ S[4] = (-1.500, -0.500), \\ S[5] = (-667.059, -333, 274) \end{array}, \ A = \begin{pmatrix} 0&1&1&0&0 \\ 1&0&0&0&0 \\ 1&0&0&1&0 \\ 0&0&1&0&1 \\ 0&0&0&1&0 \end{pmatrix},$$

while C is empty. At the end, the algorithm produces the following structure:

$$\left(C[1] = \begin{array}{l} (-0.000, 1001.000) \\ (-0.000, 1.000) \\ (1.500, -0.500) \\ (-1.500, -0.500) \\ (-667.059, -333, 274) \end{array}, \ l[1] = \text{'}open\text{'} \right),$$

so we get an open, connected decision region (the grey region in Figure 5). The other decision region (the white one) is obviously obtained by complement.

For this analysis even rough classifiers allow us to obtain correct deductions. Figures 6-10 show five classification problems characterized each by a different topology of the true decision regions. In the figures the thin lines represent the true Bayes decision borders, while the thick lines are the borders found without stressing the design of the classifier by the BVQ, that is, by choosing a uselessly high number of code vectors, by running the BVQ for a limited number of iterations and without tuning the parameters γ and Δ. We can notice that the decision borders found by BVQ differ considerably from the true ones, nevertheless, topological properties are preserved. This is especially evident in Figures 7, 8 and 10.

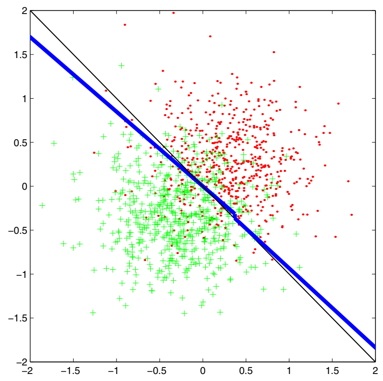


Fig. 6. A problem with a linear decision border

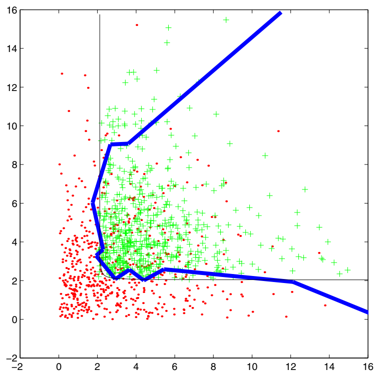


Fig. 7. A problem with a hyperbolic decision border

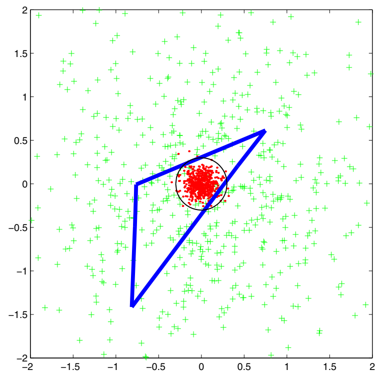


Fig. 8. A problem with a circular decision border

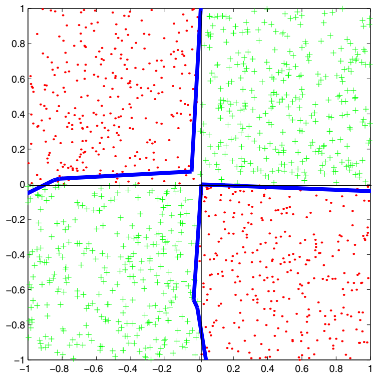


Fig. 9. The XOR problem

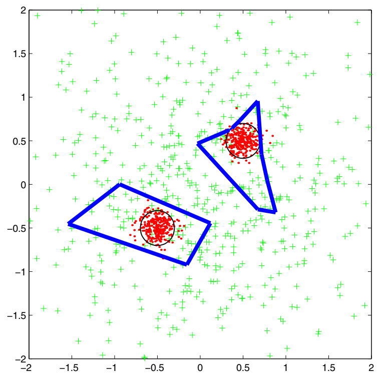


Fig. 10. A problem with a decision border formed by two disconnected circles

5.2 Extraction of Geometrical Properties of Regions

For any (sub)regions, and specially for closed ones, we can extract a number of geometrical features to refine the regions characterization. These features include the surface area, volume, principal components ratio, convexity, volume/surface ratio, position in $\mathcal{R}^n$ and so on. Principal component analysis, in the classification domain, aims to measure the most informative direction for the classification task. Following [23], it is quite simple to calculate principal components for the piecewise linear borders of an LVQ. Convexity of decision regions can be established if, for each pair of vertices a, b their convex combination $\alpha a + (1 - \alpha)b$ belongs to the decision region, or to the decision border, for each $0 \leq \alpha \leq 1$. For lack of space, we omit the description of the algorithms to calculate all these geometrical features.

The precision of most of the geometrical features depends on the quality of the classifier. Consider for example the convexity property: from Figures 7 and 10, we can deduce that convexity is not easily preserved. Notice however that in Figures 7 and 10 convexity is not preserved only in limited and local regions of space, so that in a global analysis they can be ignored. This is true in general: further elaborations and approximations of a rough classification would allow us to enhance the quality of the deductions. Notice also that comparison between regions properties, like the relative dimensions and the relative positions of the (sub)regions remains valid (see Figure 8, 9 and 10).

6 Case Studies

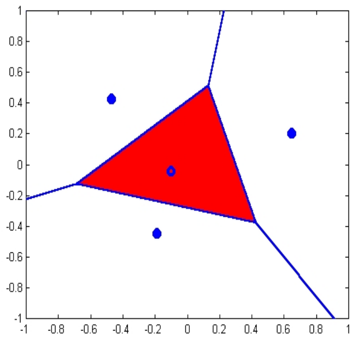
In this section we show how the knowledge given by the decision border can be exploited to support the user in the design of classification KDD processes. To this end, we consider two classification problems, the gaussian signals and the echocardiogram from the Irvine "UCI Machine Learning Repository" [1].

The former problem is to classify between two bivariate gaussian signals having the same mean and different covariance matrices. This model describes the non coherent reception of two equiprobable, noisy, binary modulated signals, and in the past years it was widely adopted as a benchmark for many recognition tasks, like feature selection [24] and classification [19]. We set the following values for means and covariance matrices:

$$\mu_1 = \mu_2 = \binom{0}{0}, \ \Sigma_1 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Sigma_2 = I * 0.01$$

The Bayes border is a circle centered in the origin of axes with radius around 0.2, Bayes error probability is 2.7%. From this population, a training set TS of size 100.000 is used to design the classifiers: 50.000 to generate the model and the leftover instances to test it. First, we apply the proposed approach to extract a rough analytical description of the decision border. The outcomes are showed in Figure 11, where the BVQ is trained with 4 code vectors.

The database of the latter classification problem, extracted from the UCI repository, represents patients who suffered heart attacks at some point in the



Decision Border:
1.000*X(1) + 0.334*X(2) = 0.302 from (0.42767,-0.37584) to (0.13028,0.51438)
1.000*X(1) + 4.457*X(2) = -1.247 from (0.42767,-0.37584) to (-0.6852,-0.12613)
1.000*X(1) + -1.273*X(2) = -0.525 from (0.13028,0.51438) to (-0.6852,-0.12613)

Fig. 11. The Decision Border equations related to the two bivariate gaussian signals problem

Decision Border:
-0.981*X(1)+ 0.130*X(2)+ -0.045*X(3)+ -0.092*X(4)+ 0.038*X(5)+
-0.004*X(6)+ -0.060*X(7)+ 0.021*X(8)+ -0.064*X(9) = -0.174

Fig. 12. The Decision Border equations related to echocardiogram database

past. The goal of this problem is to predict whether or not the patient will survive. The most difficult part of this problem is related to the size of the data set. In fact, after the data cleaning phase, the database consists of 9 features plus the class, and it contains only 132 instances. In this case, the analytical form of the decision border is obtained initializing the BVQ with only 2 code vectors. The results are shown in Figure 12.

In the following, we show how the knowledge of the decision border helps the user in the choice of the appropriate classification algorithm, in the data selection phase and for feature selection.

6.1 Choosing the Data Mining Algorithm

Support Vector Machines is a kernel-based learning methodology introduced by Vapnik [29] that finds many important applications in the Data Mining field. Varying the form of the kernel parameter, the algorithm can simulate different classifiers, from linear classifiers, to RBF and MLP neural networks. If no knowledge about the data is given, the different kernels have to be evaluated extensively, while some knowledge about the border could limit the search to the most promising ones. To prove this statement, let us consider the bivariate gaussian signals problem. Analyzing the equations of the decision border we see that it is a connected closed one (see Figure 11), so we deduce that it is not convenient to use a linear classifier, but we have to train a classifier by an al-

Table 1. Using the decision border characterization for data selection

training instances	50000	5000	2000	1000	500	100
runtime in seconds	87.44	3.38	0.65	0.22	0.07	0.03
number of support vectors	4335	1812	790	520	279	135
Accuracy on test set	97.14%	97.02%	96.95%	95.48%	94.02%	02.54%

gorithm that builds a non-linear decision rule. As a matter of fact, training the SVM algorithm with RBF kernel, we obtained an accuracy of 97.14% on the test set, and the training was carried out in 87.44 seconds. On the other hand, using a linear kernel, the same implementation of the algorithm runs on the same machine in 13683.82 seconds with an accuracy on the test set of 49.90%.

6.2 Data Selection

Data reduction techniques are exploited in combination with not scalable algorithms, that is algorithms whose high cost makes them unsuitable for the management of large amounts of data. The major information for the classification task is concentrated in the instances close to the decision border. Thus, data reduction can be performed by eliminating those samples falling far from the decision border, as less informative samples. This intuition dates back to the earliest work in Pattern Recognition, forming the basis of the condensed nearest neighbor method [15]. In the following, we show that SVM could also gain from the application of this data reduction technique. To this end, we consider again the bivariate gaussian signals problem, and we build six different experiments by training a SVM with a RBF kernel on the whole training set and on the first 100, 500, 1.000, 2.000 and 5.000 samples falling close to the decision border found by the BVQ algorithm. The results are shown in Table 1. It is noted that reducing the number of training instances from 50.000 to 2.000, the accuracy on the test set does not substantially change (it varies from 97.14% to 96.95%). On the other hand, the number of support vectors is reduced from 4.335 to 790, considerably reducing the model complexity.

Reduction in model complexity has a great impact on the training time, as shown in the Table, and also in the time needed to classify a new sample. Furthermore, the simpler the model is, the simpler it is to try to validate it and to extract symbolic rules from it.

6.3 Feature Selection

The echocardiogram database allows us to show how the characterization of the decision border provides information about the most informative features. Starting from the equations of the decision border (see Figure 12), we are able to extract the weight of any feature with respect to its contribution to classification accuracy. According to the EDBFE method [23], these weights can be obtained simply, by analyzing the vector normal to the decision border, that represents the most informative direction. Table 2 reports the feature information weights

Table 2. Weight of the features, of echocardiogram database, with respect to the classification task. Acc. W. is accumulation of weights.

#	Feature	Weight (%)	Acc. W.
1	survival	0.684	0.684
2	age-at-heart-attack	0.090	0.775
4	fractional-shortening	0.064	0.839
9	mult	0.045	0.884
7	wall-motion-score	0.042	0.925
3	pericardial-effusion	0.031	0.956
5	epss	0.026	0.983
8	wall-motion-index	0.015	0.997
6	lvdd	0.003	1.000

in decreasing order, together with the cumulative weights for all the features. Notice that, even with the simple linear border found by the BVQ, it is possible to derive results that are consistent with the domain knowledge: the most important feature to predict whether or not the patient will survive is the number of months that the patient survived immediately after the attack (the first period is clearly the most critical), while the second one is the age at which the heart attack occurred. This can be observed also empirically, since experiments performed show that training a classifier on the whole vector space and on the space formed by the 'survival' and 'age-at-heart-attack' features only, leads to the same classification accuracy.

Note that the EDBFE method extracts the information analyzing the whole data set, thus requiring a lot of computational resources, while extracting this information from the analytical form of the decision border is a straightforward and cheap operation.

The knowledge derived from this analysis can be useful for itself, giving information as to which variables mainly influence the problem, or it can be exploited to train a classifier on a limited set of features, thus reducing the so called *curse of dimensionality.*

7 Semantic Annotation of Data on a Net

Business and scientific organizations can have numerous advantages in the definition of distributed KDD processes over a network: they can share data, algorithms and computational resources, as well as methodological practices. Also isolated users can exploit the network environment to retrieve and reuse useful algorithms, tools, data and discovered models.

In this perspective, a number of proposals have been made, to define effective infrastructures for KDD process design over a net. Data and Knowledge Grids have been defined as a means to support high-performance distributed data mining in federated environments [4,6]. The service oriented paradigm is the natural extension to open environments [28,13,9,22,26,21]. This calls for languages and standards to describe resources, in order to facilitate their discovery, comprehension, exploitation, interoperability. For instance, Grossman introduced the Predictive Model Markup Language (PMML) [14] that, in its latest version, supports the description of a classification model, as well as of data transformation activities that precedes model induction in KDD. For a survey on the development of data mining related standards see [12].

The annotation of data with semantic information about the decision border can leverage the development of a class of services of particular interest in the field of KDD, that of high-level services to give support to the users in the mapping between his business goal and the Data Mining tasks, in the choice, retrieval and correct use of techniques and tools, in their efficient composition and in the understanding of the final results. In fact, the sharing of such data, together with adopted techniques and experimental results allow us to accumulate knowledge to build the knowledge-base of an intelligent support system. In the envisaged scenario, such knowledge-base manages the relationships among three different registries, representing information about the business, the data mining task and the data (Figure 13). In particular, for any performed classification experiment the knowledge-base registry stores information about the business domain, the business goal, the dataminer and his annotations, the data structure and the feature semantics, the properties of the decision border and the relations with both the methods, the algorithms and the tools used for the classification task, in accordance with a data mining ontology. The information on the decision border represents its analytical form, its geometrical and topological properties and the accuracy of the BVQ classifier used for the decision border definition. The accuracy of describing the decision border can be measured by the classification error, so if it turns out to be close to the Bayes classification error, then we can be quite confident that the decision border found is close to the Bayes border. Furthermore, the knowledge-base registry contains information about the quality of the experiment, in terms of performances of the classification algorithm: error

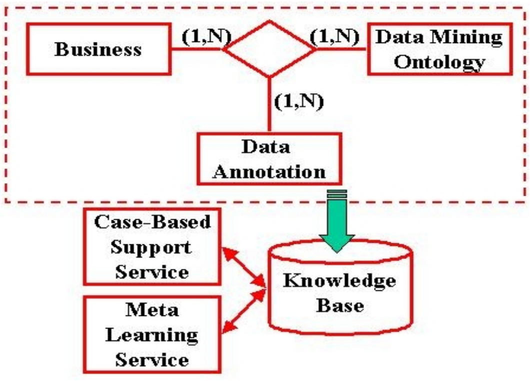


Fig. 13. The knowledge-base registry and the intelligent services of an KDD support system

probability, precision, recall, dimension of the inducted model, computation time and used memory. The accuracy of proposed method, the dataminer reliability and the performances of the algorithm are a measurement of the quality of the data semantic annotations. This information can be available over a network of virtual organizations as results of a single classification process or as similar distributed knowledge-base registries. Then, *Meta Learning Service* can collect and analyze this information to find similarities between data, by clustering the input datasets on the basis of their decision border characteristics. By mapping a cluster to the classification methods and algorithms used for the datasets belonging to the cluster, the meta-learning service can establish a relationship between data characteristics and algorithms performances. The results of the analysis performed by this service is also stored in the knowledge-base repository. In turn, this information can be exploited by another class of intelligent services, that of *Case-Based Support Services* which, querying the knowledge-base repository for Meta Learning information, can return the set of algorithms, or the typical parameter setting for a given algorithm, that have demonstrated the best performances on the data cluster similar in characteristics to a given dataset.

8 Conclusions

This work investigated the utility to exploit data semantics in the development of KDD processes. We considered a special kind of semantics for classification problems, given by the decision border. We showed that it is possible to derive some knowledge about the characteristics of the decision border and decision regions of a classification problem, starting from the geometrical properties of Vector Quantizers. Then, we showed that this knowledge can effectively help the user to take decisions and to limit the efforts in the implementation of inductive classification methodologies. We also discussed the introduction of intelligent services to collect, to analyze and to manage the semantics of data distributed over a network of virtual organizations.

An important issue related with this approach is the accuracy of the decision border needed for the different applications. By the term knowledge, it is commonly meant “a set of assertions about a phenomenon which are true to some extent and which are useful to take decisions”. The set of true assertions represents a model for the phenomenon. Different models for the same phenomenon can exist, which are valid as long as they are applied to take certain kinds of decisions. Consider for instance the Earth model given by the most common cartography. This model approximates pieces of the globe as it was flat. This is a useful model to trace the shortest route between two points which are not too far from each other, however in an Atlantic crossing it would introduce a non negligible error, and the Earth sphericity should be taken into account. A similar situation applies to our approach. The experiments reported showed that, in an evidence-based methodology to support the user in a classification KDD process, an accurate model of the decision border is not needed to understand

and to pre-process the input data. We obtained interesting experimental results for data selection, feature reduction and for the selection of the kernel parameter of the SVM algorithm even with a rough model of the classification problem. However, it is clear that, at least for the definition of the correct classification rule an accurate decision border is needed. Also, in order to build up a valid knowledge-based registry, the decision border description should guarantee an good accuracy. We plan to deeply analyze the issue of decision border accuracy in future works.

Acknowledgements

We thanks the anonymous reviewers that helped with their suggestions to improve the final quality of this manuscript.

This work is dedicated to the memory of our dear co-author Maurizio Panti, who passed away during the last revision of the paper. His vision of semantics in the KDD field deeply inspired and guided our research.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
2. Brazdil, P., Soares, C. and Costa, J. Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning*, 50(3):251–277, 2003.
3. Cannataro, M. and Comito, C. A Data Mining Ontology for Grid Programming. In *Proc. 1st Work. on Semantics in Peer-to-Peer and Grid Computing*, pages 119–130, 2003.
4. Cannataro, M. and Talia, D. The Knowledge Grid. *Comm. of the ACM*, 46(1):89–93, Jan. 2003.
5. Cespivova, H., Rauch, J., Svatek, V., Kejkula, M. and Tomeckova, M. Roles of Medical Ontologies in Association Mining CRISP-DM Cycle. In *ECML/PKDD Workshop on Knowledge Discovery and Ontologies*, pages 1–12, Pisa, Italy, 2004.
6. Chervenak, A., Foster, I., Kesselman, C. and Tuecke, S. Protocols and Services for Distributed Data-Intensive Science. In *Proc. Advanced Computing and Analysis Techniques in Physics (ACAT2000)*, pages 161–163, 2000.
7. Clarkson, K. A program for convex hulls. http://cm.bell-labs.com/netlib/voronoi /hull.html.
8. Diamantini, C. and Spalvieri, A. Quantizing for Minimum Average Misclassification Risk. *IEEE Trans. on Neural Networks*, 9(1):174–182, Jan. 1998.
9. Diamantini, C., Potena, D. and Panti, M. Developing an Open Knowledge Discovery Support System for a Network Environment. In *Proc. of the 2005 International Symposium on Collaborative Technologies and Systems*, page to appear, Saint Louis, Missouri, USA, May 15-19 2005.
10. Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.
11. Fermandez, C., Martinez, J.F., Wasilewska, A., Hadjimichael, M. and Menasalvas, E. Data Mining - a Semantic Model. In *IEEE International Conference on Fuzzy Systems*, volume 2, pages 938–943, May 2002.

12. Robert Grossman, editor. *Proc. of the Second Annual ACM KDD Workshop on Data Mining Standards, Services and Platforms*, Seattle, WA, Aug. 2004.
13. Grossman, R. and Mazzucco, M. DataSpace: a Data Web for the Exploratory Analysis and Mining of Data. *IEEE Computing in Science and Engineering*, 4(4):44–51, July-Aug. 2002.
14. Grossman, R., Hornik, M. and Meyer, G. Emerging Standards and Interfaces in Data Mining. In Nong Ye, editor, *Handbook of Data Mining.* Kluwer Ac. Pub., Apr. 2003.
15. Hart, P. E. The Condensed Nearest Neighbor Rule. *IEEE Trans. on Information Theory*, 14:515–516, 1968.
16. Hotho, A., Staab, S. and Stumme, G. Ontologies Improve Text Document Clustering. In *IEEE International Conference on Data Mining*, pages 541–544, Nov. 2003.
17. Kalousis, A. and Hilario, M. Model Selection via Meta-Learning. *Int. Journal on Artificial Intelligence Tools*, 10(4), 2001.
18. Kargupta, H., Joshi, A., Sivakumar, K. and Yesha, Y. *Data Mining, Next Generation Challenges and Future Directions.* AAAI/MIT Press, 2004.
19. Kohonen, T., Barna, G. and Chrisley, R. Statistical Pattern Recognition With Neural Networks: Benchmarking Studies. In *IEEE International Conference on Neural Networks*, pages 61–68, San Diego CA, 24-27 Jul 1998.
20. Kotasek, P. and Zendulka, J. An XML Framework Proposal for Knowledge Discovery in Databases. In *European Conference on Principles and Practice of Knowledge Discovery in Databases, Workshop on Knowledge Management: Theory and Applications*, pages 143–156, Lyon, France, 2000.
21. Krishnaswamy, S., Zaslasvky, A., and Loke, S, W. Internet Delivery of Distributed Data Mining Services: Architectures, Issues and Prospects. In V.K. Murthy and N. Shi, editors, *Architectural Issues of Web-enabled Electronic Business*, chapter 7, pages 113–127. Idea Group Publishing, 2003.
22. Kumar, A. Kantardzic, M., Ramaswamy, P. and Sadeghian, P. An Extensible Service Oriented Distributed Data Mining Framework. In *Proc. IEEE/ACM Intl. Conf. on Machine Learning and Applications*, Louisville, KY, USA, 16-18 Dec. 2004.
23. Lee, C. and Landgrebe, D.A. Feature Extraction Based on Decision Boundaries. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(4):288–400, April 1993.
24. Morgera, S.D. and Datta , L. Towards a Fundamental Theory of Optimal Feature Selection: Part I. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6(5):601–616, Sept. 1984.
25. Phillips, J. and Buchanan, B.G. Ontology-Guided Knowledge Discovery in Databases. In *1st ACM Int. Conf. on Knowledge Capture*, pages 123–130, Victoria, Canada, October 2001.
26. Sarawagi, S. and Nagaralu, S.H. Data Mining Models as Services on the Internet. *ACM SIGKDD Explorations*, 2(1):24–28, June 2000.
27. Shearer, C. The CRISP-DM Model: The new Blueprint for Data Mining. *Jour. of Data Warehousing*, 5(4), Fall 2000.
28. Talia, D. The Open Grid Services Architecture: Where the Grid Meets the Web. *IEEE Internet Computing*, 6(6):67–71, Nov/Dec 2002.
29. V. Vapnik. *Statistical Learning Theory.* J. Wiley and Sons, New York, 1998.

30. Varde, A., Rundensteiner, E., Ruiz, C., Maniruzzaman, M. and Sisson, R. Data Mining Over Graphical Results of Experiments With Domain Semantics. In *ACM 2nd Internationa Conference on Intelligent Computing and Information Systems*, Cairo, Egypt, 5-7 March 2005.
31. Verschelde, J., Casella Dos Santos, M., Deray, T., Smith, B. and Ceusters, W. Ontology-Assisted Database Integration to Support Natural Language Processing and Biomedical Data Mining. *Journal of Integrative Bioinformatics*, Jan. 2004.
32. Wang, B., McKay, R., Abbass, H. and Barlow, M. A Comparative Study for Domain Ontology Guided Feature Extraction. In *26th Australasian Computer Science Conference*, pages 69–78, Adelaide, Australia, 2003.
33. Yuhua Li and Zhengding Lu. Ontology-Based Universal Knowledge Grid: Enabling Knowledge Discovery and Integration on the Grid. In *IEEE International Conference on Services Computing*, pages 557–560, Sept. 2004.

A Algorithm for the Calculus of Voronoi Diagrams

The Appendix illustrates the algorithm for the analytical description of a Voronoi diagram, starting from the set of code vectors in general dimensional spaces $\mathcal{R}^n$. The algorithm is based on the notion of *Delaunay triangulation.*

Definition A-1. *Given a set of points S in $\mathcal{R}^2$, the* convex hull *of S is the smallest convex set in $\mathcal{R}^2$ containing S.*

Definition A-2. *Given a set of points S in $\mathcal{R}^2$, and its convex hull $\mathcal{H}$, the* Delaunay triangulation of *S is defined as the the unique triangulation of $\mathcal{H}$ such that the points in S are the vertices of the triangles, and no point of S falls inside the triangle's circumcircle.*

The definition of Delaunay triangulation can be extended to any space $\mathcal{R}^n$. In this case, it is also called *Complex of Delaunay*, and triangles are called *simplices.* The circumscribing spheres are called *circospheres*, and their centers are named *circumcenters.*

The following algorithm, starting from the Complex of Delaunay, allows to calculate the Voronoi diagram of the convex hull of a given set S.

Let be given the set $S = S_1 \cup S_2$, where $S_1 = \{m_1, m_2, \ldots, m_M\} \subseteq \mathcal{R}^n$ and $S_2 = \{m_{M+1}, m_{M+2}, \ldots, m_{M+2^n}\} \subseteq \mathcal{R}^n$, where the elements of S_2 are the vertices of an hypercube Φ, such that $m_i \in \Phi$ and the side of $\Phi \gg \| m_i - m_j \|^2$, $\forall i, j \in [1, M]$.

1. Calculate all the hyperplanes
$$\mathcal{S}_{i,j} : \| x - m_i \|^2 = \| x - m_j \|^2, i, j = 1, \ldots, M$$
2. Extract all the $\binom{M+2^n}{n+1}$ permutations of $n+1$ points belonging to S^1;

[1] Notice that $n+1$ points in $\mathcal{R}^n$ univocally define a simplex.

3. For each permutation $P_i = \{m_{Pi_1}, \ldots, m_{Pi_{n+1}}\}$, if its elements satisfy the definition A-2, calculate the circumcenter C_{P_i} and the radius r_{P_i} of the circosphere. Let H be the number of different circospheres found;

4. For each point $m_i \in S$:
   (a) The vertices of the Voronoi region $\mathcal{V}_i$ are all the circumcenters C_h, $h = 1, \ldots, H$, such that: $\| C_h - m_i \|^2 = {r_h}^2$;

   (b) Extract all the $\binom{H}{n}$ permutations of n vertices of $\mathcal{V}_i$; [2]

   (c) For each permutation $Q_i = \{C_{Qi_1}, \ldots, C_{Qi_n}\}$, calculate the hyperplane including the points $C_{Qi_1}, \ldots, C_{Qi_n}$. If this hyperplane belongs to the set of hyperplanes generated in step 1, this is a piece of the border of the $\mathcal{V}_i$ region, that is bounded by the circumcenters in Q_i.

Notice that we define the set S of input points as the union of two sets: S_1, which represents the code vectors of the VQ we want calculate the Voronoi diagram of, and S_2, which represents the 2^n vertices of a hypercube containing the real code vectors and having the side much wider than the maximal dimension of the convex hull of S_1. These points represent "points at infinite" and are named *fictitious code vectors*. In this way, the convex hull of S is an expansion of the convex hull of S_1 and the outcome diagram contains all the Voronoi borders of S_1. However, by introducing S_2 we get also fictitious circumcenters and fictitious hyperplanes. It is not difficult to individuate and eliminate fictitious circumcenters and hyperplanes: the former are represented by values which are closer to the values of fictitious code vectors than to real code vectors. The latter are identified as the hyperplanes separating the Voronoi regions of two code vectors at least on of which is a fictitious code vector. Finally, notice that the result is affected by an error which depends on the distance between the elements of S_2 and S_1. For these reasons, it is important that fictitious code vectors are chosen to be very far from the real code vectors.

[2] The choice of the set S guarantees that n such circumcenters always exist.

Integrating the Two Main Inference Modes of NKRL, Transformations and Hypotheses

Gian Piero Zarri

LaLICC, Université Paris4-Sorbonne, Maison de la recherche, 28 rue Serpente, 75006 Paris, France
gpzarri@paris4.sorbonne.fr

Abstract. An application of NKRL (Narrative Knowledge Representation Language) techniques on terrorism documents supplied by the Greek Ministry of Defence (MoD) has been carried out in the context of the IST Parmenides project; this application has required implementing the integration between the two main inferencing modes of NKRL, 'hypotheses' and 'transformations'. 'Hypothesis rules' allow retrieving automatically from an NKRL knowledge base the information that can supply a *context* or a *causal explanation* for some known event. 'Transformation rules' facilitate the recovery of information from the base by '*adapting*' (transforming) query/queries that failed to the real contents of this base. Integrating the two classes of rules means using the transformations to *automatically modify* the *pre-defined reasoning steps* of a hypothesis to build up more flexible and complete 'explanation' scenarios.

1 Introduction

According to the paper "Countering Terrorism Through Information Technology" in the CACM Issue on Homeland Security of March 2004, information technologies that are considered as important for counterterrorism include, amongst other things, "… categorize [meaning and key concepts] via an information model (taxonomy, ontology) [and] cluster documents with similar content … index, store, retrieve, extract, integrate, analyse, aggregate, display and distribute semantically enhanced information from a wide variety of sources … provide an overall view of the different topics related to the request, along with the ability to visualize the semantic links relating the various items of information to each other … use Semantic Web, associative memory, and related technologies to model and make explicit … an analyst's personal preferences … and [the] tacit understanding of a problem domain " [1: 39]. It seems then reasonable to assume that the faculties of NKRL, the Narrative Knowledge Representation Language, of representing in-depth and in a computer-exploitable way the semantic properties of complex 'narrative documents' (in the widest meaning of these words), of associating automatically disjoint semantic information, of representing faithfully and efficiently multifaceted (and sometimes contradictory) strategies of search and inferencing, see [2, 3, 4], could make it a good candidate for exploiting in an intelligent way terrorism information.

An in-depth experiment concerning the use of the inferencing capabilities of NKRL on a corpus of news supplied by the Greek Ministry of Defence (MoD) and related to terrorism in Philippines between 1993 and 2000 has been carried out in the context of the EC-supported Parmenides project (IST 2001-39023), see, e.g., [5]. To get the best from the modelling capabilities of NKRL, this experiment has required the integration of the two main modalities of inferencing of NKRL, 'hypotheses' and 'transformations', see below Section 2. 'Transformation rules' try to automatically replace some retrieval queries that failed with one or more different queries that are not strictly 'equivalent' but only 'semantically close' to the original one. The queries – called 'search patterns' according to the NKRL technical jargon – are represented in NKRL terms; they operate by unification/filtering on the contents of a knowledge base formed by the NKRL representations of the 'semantic content' (the *meaning*) of multimedia original documents. 'Hypothesis rules' allow building up causal-like explications of given events according to pre-defined reasoning schemata – when these reasoning schemata are activated by an inference engine, they are converted into search patterns that try to find an unification with the contents of the base. Integrating the two inferencing modes corresponds to allow the 'transformations' to modify in an unpredictable way the reasoning steps to be executed within a 'hypothesis' context. This is equivalent to 'break' the predefined scenarios proper to the hypothesis rules and to augment then the possibility of discovering 'implicit information' within the knowledge base – or, in terms of the CACM paper mentioned before, to increase the possibility of automatically "visualiz(ing) the semantic links relating the various items of information to each other".

In this paper, we recall firstly, in Section 2, the main principles underpinning NKRL and, in particular, the functioning of its inference engine(s). We will then illustrate, in Section 3, the theoretical problems that the integration implies – e.g., that of finding a correspondence between the hypothesis and transformation variables in an 'integrated' context – and the solutions adopted. Section 4 will illustrate briefly how these solutions have been practically implemented. Section 5 will deal with future work; Section 6 will make some comparison with related research. Section 7, a short Conclusion, will end the paper. Many of the examples used in the paper for illustration purpose will refer to the MoD corpus. Using these examples does not mean that the underlying assumptions are shared or supported by the author of the paper or the editor of JoDS. Neither the author, not the editor, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy or completeness of such information.

2 NKRL and the NKRL Inference Techniques

2.1 General Information About NKRL

NKRL innovates with respect to the current ontological paradigms, both the 'traditional' ones – see, e.g., [6] – and those inspired by the Semantic Web research – see [7, 8] – by adding to the usual ontologies of concepts an 'ontology of events', i.e., a new sort of hierarchical organization where the nodes correspond to *n*-ary structures called 'templates'. The term 'event' is taken here in its more general meaning, covering also strictly related notions like fact, action, state, situation etc., see [3]. In the NKRL environment, the 'ontology of concepts' is called HClass (hierarchy of classes): HClass is not fundamentally different from one of the 'traditional' ontologies that can be built up by using tools in the Protégé style, see again [6]. The 'ontology of events' – HTemp, hierarchy of templates – is, on the contrary, basically different from classical ontologies; the two hierarchies, HClass and HTemp, operate in a strictly integrated way in an NKRL context. A partial representation of HClass is given in Figure 1; a full description is given in [9]. See [2] for a discussion about concepts like `non_sortal_concept` (the specialisations of this concept, i.e., its subsumed concepts, cannot be endowed with direct instances), `sortal_concept` or `substance_`. Figure 2 represents part of the `Produce:` branch of HTemp, see Table 1a below for details about the 'internal' structure of templates. Both HClass and HTemp are the 'pragmatic' result of many years of successful experiments in dealing with complex narrative information by using high-level conceptual tools.

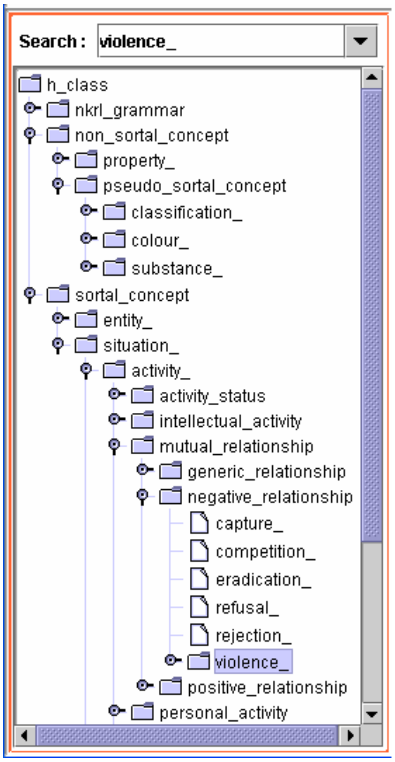
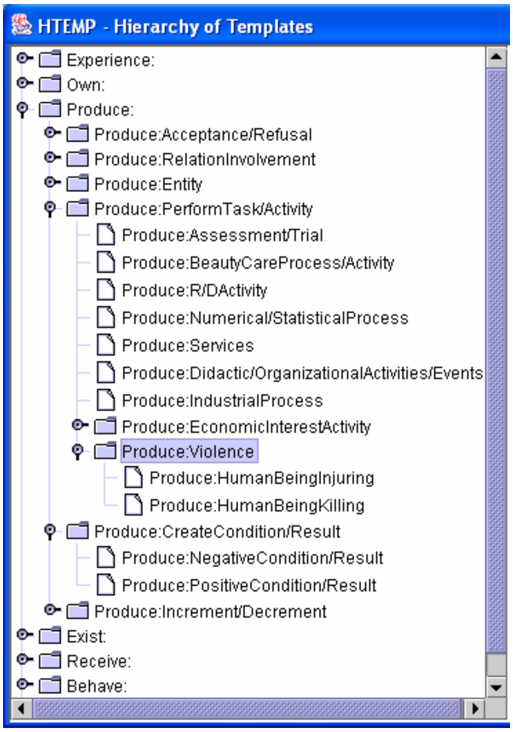


Fig. 1. Partial representation of HClass, the 'traditional' ontology of concepts

Instead of using the traditional *object (class, concept) – attribute – value* organization, templates are generated from the association of *quadruples* connecting together the *symbolic name* of the template, a *predicate*, and the *arguments* of the predicate introduced by named relations, the *roles*. The different quadruples making up a given template have in common the 'name' and 'predicate' components. If we denote then with L_i the generic symbolic label identifying a specific template, with P_j the predicate, with R_k a generic role and with a_k the corresponding generic argument, the NKRL core data structure for templates has the following general format:

$$(L_i\ (P_j\ (R_1\ a_1)\ (R_2\ a_2)\ \ldots\ (R_n\ a_n)))\ . \tag{1}$$

Predicates pertain to the set {`BEHAVE, EXIST, EXPERIENCE, MOVE, OWN, PRODUCE, RECEIVE`}, and roles to the set {`SUBJ(ect), OBJ(ect), SOURCE, BEN(e)F(iciary), MODAL(ity), TOPIC, CONTEXT`}; predicates and roles are then 'primitives'. An argument a_k of the predicate denotes indirectly through a 'variable' (see below) either a simple 'concept' (according to the traditional, 'ontological' meaning of this word) or a structured association ('expansion') of several concepts. In both cases, the concepts can only be chosen among those included in the HClass hierarchy; this fact, linked with the 'primitive' character of predicates and roles, allows to reduce considerably the potential combinatorial explosion associated with formulas like (1).


HTEMP - Hierarchy of Templates

- Experience:
- Own:
- Produce:
  - Produce:Acceptance/Refusal
  - Produce:RelationInvolvement
  - Produce:Entity
  - Produce:PerformTask/Activity
    - Produce:Assessment/Trial
    - Produce:BeautyCareProcess/Activity
    - Produce:R/DActivity
    - Produce:Numerical/StatisticalProcess
    - Produce:Services
    - Produce:Didactic/OrganizationalActivities/Events
    - Produce:IndustrialProcess
    - Produce:EconomicInterestActivity
    - Produce:Violence
      - Produce:HumanBeingInjuring
      - Produce:HumanBeingKilling
  - Produce:CreateCondition/Result
    - Produce:NegativeCondition/Result
    - Produce:PositiveCondition/Result
  - Produce:Increment/Decrement
- Exist:
- Receive:
- Behave:

Fig. 2. Partial representation of the `PRODUCE` branch of HTemp, the 'ontology of events'

Templates represent formally generic classes of elementary events like "move a physical object", "be present in a place", "produce a service", "send/receive a message", "build up an Internet site", etc. – for additional details and a full description of HTemp, see again [9]. When a particular event pertaining to one of these general classes must be represented, the corresponding template is 'instantiated' to produce what, in the NKRL's jargon, is called a 'predicative occurrence'. To represent a simple narrative – extracted from one of the new stories of the MoD corpus – like: "On November 20, 1999, in an unspecified village, an armed group of people has kidnapped Robustiniano Hablo", we must then select firstly in the HTemp hierarchy the template corresponding to 'execution of violent actions', see Figure 2 and Table 1a below. This template is a specialization (as documented by the 'father' code in Table 1a) of the particular `PRODUCE` template corresponding to "perform some task or activity".

As it appears from Table 1a, the arguments of the predicate (the a_k terms in (1)) are represented in practice by variables with associated constraints – which are expressed as HClass concepts or combinations of HClass concepts. When deriving a predicative occurrence (an instance of a template) like `mod3.c5` in Table 1b, the role fillers in this occurrence must conform to the constraints of the father-template. For example, in this occurrence, `ROBUSTINIANO_HABLO` (the '`BEN(e)F(iciary)`' of the action of kidnapping) and `INDIVIDUAL_PERSON_20` (the unknown '`SUBJECT`', actor, initiator etc. of this action) are both 'individuals', instances of the HClass concept `individual_person`: this last is, in turn, a specialization of `human_being_or_social_body`, see, in Table 1a, the constraint on the variables *`var1`* and *`var6`*. `kidnapping_` is a concept, specialization of `violence_`, see *`var3`*, etc. Throughout this paper, we will use small letters to represent a `concept_`, capital letters to represent an `INDIVIDUAL_`.

The constituents (as `SOURCE` in Table 1a) included in square brackets are optional. A 'conceptual label' like `mod3.c5` is the symbolic name used to identify the NKRL code corresponding to a specific predicative occurrence; see [10: Appendix B] for details about the representation of NKRL data structures onto ORACLE databases. The first component of this 'pointed' notation identifies the original document represented into NKRL format, in this case, the third news story of the MoD corpus. The second component, `c5`, tells us that the corresponding (predicative) occurrence is the fifth within the set of predicative and binding (see below) occurrences that represent together the NKRL 'image' of the original document. This complete image is called a '*conceptual annotation*' of the document in the NKRL's jargon.

The 'attributive operator', `SPECIF(ication)`, is one of the four operators that make up the AECS sub-language, used for the construction of 'structured arguments' ('complex fillers' or 'expansions'); apart from `SPECIF(ication)` = S, AECS includes also the disjunctive operator, `ALTERN(ative)` = A, the distributive operator, `ENUM(eration)` = E, and the collective operator, `COORD(ination)` = C. The interweaving of the four operators within an expansion is controlled by the so-called 'precedence rule', see [2, 4]. The `SPECIF` lists, with syntax (`SPECIF` e_i p_1 ... p_n), are used to represent the properties or attributes that can be asserted about the first element e_i, concept or individual, of the list – e.g., in the `SUBJ` filler of `mod3.c5`, Table 1b, the attributes `weapon_wearing` and `(SPECIF cardinality_ several_))` are both associated with `INDIVIDUAL_PERSON_20`. `weapon_wearing` is a specialization of the `dressing_attribute` concept of HClass, i.e., via the generalizations `dressing_attribute` and `physical_aspect_attribute`, it is also a specialization of `animate_entity_` property. (SPECIF cardinality_ several_)), which is in turn a SPECIF lists, is the standard way of representing the 'generic plurality' in NKRL, see [2]. several_ – a concept of the logical_quantifier branch of HClass – 'gives details' about cardinality_, a quantifying_property, etc.

Table 1. Building up and querying predicative occurrences

a)

```
name: Produce:Violence
father: Produce:PerformTask/Activity
position: 6.35
NL description: 'Execution of Violent Actions on the Filler of the
BEN(e)F(iciary) Role'

PRODUCE    SUBJ          var1: [(var2)]
           OBJ           var3
           [SOURCE       var4: [(var5)]]
           BENF          var6: [(var7)]
           [MODAL        var8]
           [TOPIC        var9]
           [CONTEXT      var10]
           {[modulators], ≠abs}

var1  =  <human_being_or_social_body>
var3  =  <violence_>
var4  =  <human_being_or_social_body>
var6  =  <human_being_or_social_body>
var8  =  <criminality/violence_related_tool> | <machine_tool> |
      <general_characterising_property> | <violence_> | <weapon_> |
         <small_portable_equipment>
var9  =  <h_class>
var10 =  <situation_> | <spatio/temporal_relationship> |
      <symbolic_label>
var2, var5, var7 = <geographical_location>
```

b)

```
mod3.c5)   PRODUCE    SUBJ (SPECIF INDIVIDUAL_PERSON_20 weapon_wearing
                      (SPECIF cardinality_ several_)): (VILLAGE_1)
           OBJ        kidnapping_
           BENF       ROBUSTINIANO_HABLO
           CONTEXT    #mod3.c6
           date-1:    20/11/1999
           date-2:

Produce:Violence (6.35)
```

*On November 20, 1999, in an unspecified village (*VILLAGE_1*), an armed group of people has kidnapped Robustiniano Hablo.*

c)

```
PRODUCE
SUBJ : human_being :
OBJ :   violence_
BENF : human_being :
{}
date1 : 1/1/1999
date2 : 31/12/1999
```

There is any information in the system concerning violence activities during 1999?

The 'location attributes', represented in the predicative occurrences as lists, are linked with the arguments of the predicate by using the colon operator, '**:**', see the individual VILLAGE_1 in Table 1b. In the occurrences, the two operators date-1, date-2 materialize the temporal interval normally associated with narrative events, see again Table 1b; a detailed description of the methodology for representing temporal data in NKRL can be found in [3].

Until now, we have evoked the NKRL solutions to the problem of representing *elementary (simple) events*. To deal now with those *'connectivity phenomena'* that arise when several elementary events are connected through causality, goal, indirect speech, co-ordination and subordination etc. links, the basic NKRL knowledge representation tools have been complemented by more complex mechanisms that make use of second order structures created through *reification* of the conceptual labels of the predicative occurrences, see [2, 3] for further details. A simple example concerns the filler of the CONTEXT role in the occurrence mod3.c5 of Table 1b: in this case ('completive construction'), the 'context' of the kidnapping is supplied by a whole predicative occurrence, mod3.c6, telling us that the kidnapping happened when Robustiniano Hablo was on his way home with his father. The code '#' indicates to the NKRL interpreter that the associated item is a conceptual label and not a concept or individual.

More complex examples of second order constructions are the 'binding occurrences', i.e., NKRL structures consisting of lists of symbolic labels of predicative occurrences; the lists are differentiated making use of specific binding operators like GOAL, COND(ition) and CAUSE. Let us suppose we would now state that: "...an armed group of people has kidnapped Robustiniano Hablo *in order to* ask his family for a ransom", where the new elementary event: "the unknown individuals will ask for a ransom" corresponds to a new predicative occurrence, e.g., mod3.c7. To represent this situation completely, see Table 2, we must add to the occurrences that make up the 'conceptual annotation' for the Robustiniano Hablo's story a new *binding occurrence*, e.g., mod3.c8, to link together the conceptual labels mod3.c5 (corresponding to the kidnapping predicative occurrence, see also Table1b) and mod3.c7 (corresponding to the new predicative occurrence describing the intended result). mod3.c8 will have then the form: "mod3.c8)(GOAL mod3.c5 mod3.c7)", see Table 2. The meaning of mod3.c8 can be paraphrased as: "the activity described in mod3.c5 is focalised towards (GOAL) the realization of mod3.c7". We can note that (see again [3] for more details):

- The second predicative occurrence mentioned in a GOAL binding structure (the second argument of a GOAL operator, i.e., the intended result), which corresponds to an elementary events that will happen '*in the future*' with respect to the event described by the first predicative occurrence (the first argument of GOAL, i.e., the action), is systematically marked as '*hypothetical*' by the presence of an uncertainty validity attribute, code '*', see occurrence mod3.c7 in Table 2.
- The (unknown) date of the (possible) ransom request is simulated by a 'fork' in the first date field (date-1) of mod3.c7, where the first term of the fork is the kidnapping date, and the second (reconstructed, i.e., not specifically attested

in the original documents, see the 'parenthesis' codes) corresponds to a month after. The meaning of the global `date-1` filler amounts then to say that the request will be (possibly) made at an unknown date included within this interval. Note that the insertion of `(20/12/1999)` in the second date field (`date-2`) of `mod3.c7` would have led to this interpretation of `mod3.c7`, see [3]: the (possible) action of sending the ransom's request to the Robustiniano Hablo's family will take a full month.

Table 2. Predicative and binding occurrences

```
mod3.c8)      (GOAL  mod3.c5  mod3.c7)
```

The information conveyed by `mod3.c8` *says that the aim of the 'behaviour' related in* `mod3.c5` *is to arrive at the situation described in* `mod3.c7`.

```
mod3.c5) PRODUCE  SUBJ    (SPECIF INDIVIDUAL_PERSON_20 weapon_wearing
                           (SPECIF cardinality_ several_)): (VILLAGE_1)
                  OBJ      kidnapping_
                  BENF     ROBUSTINIANO_HABLO
                  CONTEXT  #mod3.c6
                  date-1:  20/11/1999
                  date-2:

Produce:Violence (6.35)
```

On November 20, 1999, in an unspecified village (`VILLAGE_1`*), an armed group of people has kidnapped Robustiniano Hablo.*

```
*mod3.c7)  MOVE  SUBJ     (SPECIF INDIVIDUAL_PERSON_20 (SPECIF
                             cardinality_ several_))
                 OBJ      RANSOM_DEMAND_1
                 BENF     (SPECIF family_ ROBUSTINIANO_HABLO)
                 TOPIC    (SPECIF hostage_release ROBUSTINIANO_HABLO)
                 date-1:  20/11/1999, (20/12/1999)
                 date-2:

Move:GenericInformation (4.41)
```

The kidnappers will (possibly) send a ransom demand to the family of Robustiniano Hablo about his release.

The NKRL software environment, see [10], is endowed with several tools designed to facilitate the set up of large knowledge bases of NKRL annotations/occurrences. For example, two different 'step-wise' programs – that differ mainly with respect to the organization of their interface modules – both allow a progressive construction of the predicative and binding occurrences by answering a set of interwoven queries. The two programs include a facility to accelerate the set up of the occurrences by simply 'adapting' to the situation to be represented – e.g., by changing some individuals or locations – 'standard' examples of well-formed predicative occurrences associated with the different templates. Another program that is particularly useful for experienced NKRL operators is a sort of 'parser' that takes as input a Word/RTF file of NKRL structures represented in 'external' format (like that used in Tables 1 and 2) and automatically produces the corresponding 'internal' format, i.e., the format to be concretely used for the querying/inferencing operations – this internal format is exhaustively described in [10: Appendix A]. The interest of this program consists in the possibility of making use of the usual Word editing facilities for generating quickly new 'external' NKRL files from the many external files concerning past NKRL applications that already exist. Moreover, the software environment is already configured for taking into account the possibility of a synthesis of NKRL structures directly from Natural Language (NL) descriptions; however, this possibility is not fully operational at present. Even if the theoretical principles for performing the synthesis of NKRL-like structures from NL descriptions are very well understood, see, e.g., [11, 12], NKRL's designers think that the amount of manual post-editing operations to be executed on the outputs of the NL/NKRL 'translation' modules is still too important to make these modules concretely useful. However, thanks to the Parmenides project, progresses have been recently accomplished in this context along the traditional MUC (Message Understanding Conferences) lines see, e.g., [13].

2.2 'Search Patterns' and Low-Level Inference Procedures

The *basic building block* for all the NKRL querying and inference procedures is the `Fum`, Filtering Unification Module, see also [4]. It takes as input specific NKRL data structures called 'search patterns'.

Search patterns can be considered as the NKRL counterparts of natural language queries; they offer then the possibility of querying *directly* an NKRL knowledge base of conceptual annotations. Formally, these patterns correspond to *specialized/partially instantiated templates* pertaining to the HTemp hierarchy, where the '*explicit variables*' that characterize the templates (var_i, see Table 1a) *have been replaced by concepts/individuals compatible with the constraints imposed on these variables in the original templates*. In a search pattern, the concepts are used as '*implicit variables*'. When trying to unify a search pattern with the predicative occurrences of the knowledge base, a concept can then *match* the individuals representing its own instances and all its subsumed concepts in HClass with their own instances. The set of predicative occurrences unified by a given search pattern constitute the *answer* to the query represented by the pattern.

Note that the unification/filtering operations executed by `Fum` are 'oriented', which means that *all the terms* used to build up a search pattern must be *explicitly found* in the matched occurrences, either in an identical form (e.g., predicate and roles), or as subsumed concepts or instances of the implicit variables. Additional terms – roles, fillers and part of fillers – with respect to those explicitly declared in the pattern can be freely found in the occurrences. Moreover, the unification of complex fillers (expansions) built up making use of the AECS operators, see the previous Section, must take into account the NKRL criteria for the creation of well-formed expansions. This implies that, during the unification, the complex fillers of search pattern and occurrences must be decomposed into tree structures labeled with the four operators, and that the unification of these tree structures must follow the constraints defined by the 'priority rule' already mentioned, see again [2, 4]. The algorithmic structure of Fum is, eventually, quite complex, but this complexity is totally transparent for the user, who is only required to specify in the search pattern the *essential elements* he wants to retrieve in the occurrences, without being bothered, as in OWL-QL for example, see [14], by the need of declaring 'Must-Bind Variables', 'May-Bind Variables', 'Don't-Bind Variables', 'Answer Patterns', etc.

A simple example of search pattern, translating the query: "There is any information in the system about violence events occurred during the year 1999?" is reproduced in Table 1c, producing the occurrence mod3.c5 (Table 1b) as one of the possible answers. Note that the two timestamps, *date1* and *date2* associated with the pattern constitute now the 'search interval' used to limit the search for unification to the slice of time that the user considers as appropriate to explore, see [3]. Note also that the search pattern of Table 1c – as, by the way, the answer mod3.c5 of Table 1b – derives from the template Produce:Violence (6.35) of Table 1a where all the explicit variables var_i have been replaced by HClass concepts (implicit variables) corresponding to some of the original constraints. As already stated, unification is executed by assuming that a generic HClass concept (implicit variable, e.g., violence_ in 1c) included in the pattern can unify all its subsumed concepts (e.g., kidnapping_ in 1b) or instances (individuals) that are present in the corresponding positions of the occurrences – this correspond to a semantic/conceptual 'expansion' of the original query. For example, both INDIVIDUAL_PERSON_20 and ROBUSTINIANO_HABLO have been registered in HClass as instances of individual_person – specific term of human_being – at the moment of the coding in NKRL terms of the corresponding news story. 'Generic' and 'specific' refer, obviously, to the structure of HClass. Because of this semantic/conceptual expansion of the proposed pattern, we can define the process of search patterns unification as a *first level of inferencing* of NKRL.

2.3 'Hypothesis' Rules

We have already mentioned, in the 'Introduction', that the *high-level inferencing operations* of NKRL correspond normally to the use of two complementary classes of inference rules, hypotheses and transformations – other sorts of rules, e.g., 'filtering' rules, have been employed for particular applications, see [15] for some information in this context. Execution of both hypotheses and transformations require the use of a real InferenceEngine, having Fum as its core mechanism. To describe the functioning of InferenceEngine in a 'hypothesis' environment we will make use of a 'standard', very simple example that, at the difference of the complex hypotheses proper to a terrorism context, see also next Sections, implies only two steps of reasoning and can be then described in some detail in a concise way.

Let us then suppose we have directly retrieved, thanks to an appropriate search pattern, the occurrence conc2.c34, see Table 3a, which corresponds to the information: "Pharmacopeia, an USA biotechnology company, has received 64,000,000 USA dollars from the German company Schering in connection with a R&D activity". We will suppose, moreover, that this occurrence is not *explicitly* related with other occurrences in the base by second order elements (e.g., binding occurrences), see Section 2.1 above. Under these conditions, we can activate the InferenceEngine module, asking it to try to *link up automatically* the information found by `Fum` with other information present in the base. If this is possible, this last information will represent, in a way, a sort of ‘causal explanation’ of the information originally retrieved – i.e., in our example, an ‘explanation’ of the money paid to Pharmacopeia by Schering. A rule that fits our case is hypothesis *h1* reproduced in Table 3b.

Table 3. An example of hypothesis rule

a)

```
conc2.c34)  RECEIVE  SUBJ  (SPECIF PHARMACOPEIA_
                             (SPECIF biotechnology_company USA_))
                     OBJ   (SPECIF money_ usa_dollar (SPECIF amount_
                             64,000,000))
                     SOURCE  (SPECIF SCHERING_
                             (SPECIF pharmaceutical_company GERMANY_))
                     TOPIC  r_and_d_activity
                     date1 :
                     date2 :
```

b)
HYPOTHESIS *h1*

premise :

```
RECEIVE    SUBJ     var1
           OBJ      money_
           SOURCE   var2

var1 = company_
var2 = human_being, company_
```

A company has received some money from another company or a physical person.

first condition schema (*cond1*) :

```
PRODUCE      SUBJ    (COORD var1 var2)
             OBJ     var3
             BENF    (COORD var1 var2)
             TOPIC   (SPECIF process_ var4)

var3 = mutual_relationship, business_agreement
var4 = artefact_
```

The two parties mentioned in the premise have concluded an agreement about the creation of some sort of ‘product’.

second condition schema (*cond2*) :

```
PRODUCE      SUBJ       var1
             OBJ        var4
             MODAL      var5
             CONTEXT    var3

var5 = industrial_process, technological_process
```

The company that received the money has actually created the product mentioned in the first condition schema.

From an algorithmic point of view, InferenceEngine works according to a backward chaining approach with chronological backtracking, see, e.g., [16]. The differences with respect to other applications of this approach (Mycin, PROLOG …) are mainly linked with the *complexity* of the NKRL data structures that implies, after a deadlock, the execution of difficult operations of restoration of the program environment to return to the previous choice point. Four '*environment variables*' are used:

- VALAFF (*valeurs affectables* in French), holds the values provisionally affected to the variables var_i of the three schemata of Table 3 (*premise*, *cond1* and *cond2*) that implement the reasoning steps of the hypothesis: these values can be deleted after a backtracking operation;
- DESVAR holds the final values associated with the variables var_i when the successful processing of one of the reasoning schemata has been completed;
- RESTRICT holds all the constraints (HClass terms) associated with the variables var_i of the different reasoning schemata: these constraints will be used to build up systematically *all* the search patterns that can be derived from these schemata, see below;
- OCCUR holds the list of the symbolic names of all the occurrences retrieved by the search patterns derived from the reasoning schemata: the values bound to var_i that have been retrieved in these occurrences are used to build up the VALAFF lists.

The first set of operations corresponds to the execution of the Exeprem sub-module of InferenceEngine, and consists in trying to unify, *using Fum*, the *premise* of the hypothesis, see Table 3b, and the event (the payment in our case, see conc2.c34) to be 'explained' – more exactly, in trying to unify (using Fum) *the event and the different search patterns derived from the premise by systematically substituting to the variables var1 and var2, see Table 1b, the associated constraints*. As already stated, search pattern processed by Fum can only include implicit variables (concepts). This first step allows then i) to verify that the hypothesis tested is, in principle, suitable to 'explain' the particular event at hand, and ii) to obtain from the external environment (the event, i.e., conc2.c34) some values for the premise variables *var1*, *var2*. In our case, the premise variable *var1* can only be substituted by the constraint company_; on the contrary, two substitutions, *var2* = human_being and *var2* = company_ are possible for the variable *var2*. A first search pattern will be then built up by substituting human_being for *var2* (a value human_being is provisionally associated with *var2* in VALAFF), i.e., a first unification with the event to explain will be tried by using a search pattern corresponding to a payment done by an *individual person* instead of a *company*. This unification obviously fails.

The engine then 'backtracks' making use of a second sub-module of InferenceEngine, Reexec: Reexec is systematically used during the execution of a hypothesis rule i) to backtrack when a deadlock occurs, and ii) to reconstruct, making use of the environment variables, the data structures (environment) proper to the previous choice point. The association `var2 = human_being` is removed and, using the constraint values stored in `RESTRICT`, the engine builds up a new pattern making use now of the value `var2 = company_`, that will unify the value `SCHERING_` in `conc2.c34`. The engine can then continue the processing of the hypothesis `h1`; the two values `var1 = PHARMACOPEIA_` and `var2 = SCHERING_` will then be stored in `DESVAR` and passed to the first condition schema (`cond1`), see Table 3b. The search patterns derived from this condition schema – by taking into account the values already bound in `DESVAR` to `var1` and `var2` and by replacing systematically, as usual, all the other variables with the associated constraints – will be tested by a third sub-module of `InferenceEngine`, `Execond`. This last is called *whenever there exist conditions favourable for advancing in the hypothesis*, in other words, for being able to process a new condition schema. `Exeprem` and `Execond` perform then the forward traversal of the choice tree, with `Reexec` being systematically called whenever the conditions for a backtracking exist. The difference between `Exeprem` and `Execond` consists mainly in the fact that, in an `Execond` context, the unification of the search patterns derived from the condition schemata is tested *against the general knowledge base of predicative occurrences to (try to) find possible unifications with these occurrences* while, in an `Exeprem` context, the unification concerns only the search patterns derived from the premise and the (unique) starting occurrence.

As usual, many deadlocks are generated in the course of the `Execond` operations. Some are due, as in the premise case, to the *chronological utilization* of the constraints. For example, when trying to make use of a pattern derived from `cond1` where the variable `var3` has been substituted by its first constraint `mutual_relationship`, see Table 3b (and Figure 1), a failure will be generated and `Reexec` will be invoked again. The occurrences we must retrieve in the knowledge base about the relationships between Pharmacopeia and Schering concern, in fact, possible sorts of *commercial* agreements between Pharmacopeia and Schering – e.g., `r_and_d_agreement` and `sale_agreement`, see below, both specific terms in HClass of `business_agreement` (the second constraint on `var3`) – and not a *private* arrangement like `mutual_relationship`. We will, eventually, find in the base an instantiation of `cond1` corresponding to an event of the form: "Pharmacopeia and Schering have signed two agreements concerning the production by Pharmacopeia of a new compound, `COMPOUND_1`". The values associated with the variables `var3` (`r_and_d_agreement` and `sale_agreement`) and `var4` (`COMPOUND_1`) in `cond1` will then be used to create the search patterns derived from `cond2`. It will then be possible to retrieve an occurrence corresponding to the information: "In the framework of an R&D agreement, Pharmacopeia has actually produced the new compound", see [4] for more details. The global information retrieved through the execution of the hypothesis, see Table 4, can then supply a sort of 'plausible explanation' of the Schering's payment: Pharmacopiea and Schering have concluded some agreements for the production of a given compound, and this compound has been effectively produced by Pharmacopeia.

2.4 'Transformation' Rules

The 'transformation rules' are used to obtain a *plausible answer* from a repository of predicative occurrences also in the absence of the explicitly requested information (i.e., when a direct query formulated in `Fum` terms fails), by searching *semantic affinities* between what is requested and what is really present in the repository. The principle employed consists in using these rules to automatically 'transform' the original query (i.e., the original search pattern) into one or more different queries (patterns) that *are not strictly 'equivalent' but only 'semantically close' to the original one.*

Table 4. Final results for hypothesis *h1*

<u>The start occurrence</u> :

```
conc2.c34) RECEIVE  SUBJ    (SPECIF PHARMACOPEIA_
                              (SPECIF biotechnology_company USA_))
                    OBJ     (SPECIF money_ usa_dollar
                              (SPECIF amount_ 64,000,000))
                    SOURCE  (SPECIF SCHERING_
                              (SPECIF pharmaceutical_company GERMANY_))
                    TOPIC    r_and_d_activity
                    date1 :
                    date2 :
```

Pharmacopeia, an USA biotechnology company, has received 64,000,000 dollars by Schering, a German pharmaceutical company, in relation to R&D activities.

<u>The result for level 1</u> :

```
conc13.c3)  PRODUCE  SUBJ    (COORD PHARMACOPEIA_ SCHERING_)
                     OBJ     (COORD r_and_d_agreement sale_agreement)
                     BENF    (COORD PHARMACOPEIA_ SCHERING_)
                     TOPIC   (SPECIF synthesis_
                               (SPECIF COMPOUND_1 new_))
                     date1 :
                     date2 :
```

Pharmacopeia and Schering have signed two agreements (have produced two agreements having themselves as beneficiaries) concerning the production of a new compound .

<u>The result for level 2</u> :

```
conc13.c7)  PRODUCE   SUBJ      PHARMACOPEIA_
                      OBJ       COMPOUND_1
                      MODAL     biotechnology_process
                      CONTEXT   r_and_d_agreement
                      date1 :
                      date2 :
```

In the framework of an R&D agreement, Pharmacopeia has actually produced the new compound .

As a first example, let us suppose that, working in the context of a hypothetical knowledge base of NKRL occurrences about university professors, we want to ask a query like: “Who has lived in the United States”, even without an explicit representation of this fact in the base. If this last contains some information about the degrees obtained by the professors, we can tell the user that, although we do not explicitly know who lived in the States, we can nevertheless look for people having an American degree. This last piece of information, obtained by transformation of the original query, would indeed normally imply that some time was spent by the professors in the country, the United States, which issued their degree. To pass now to a MoD example, suppose we ask: “Search for the existence of some links between ObL (a well-known ‘terrorist’) and Abubakar Abdurajak Janjalani, the leader of the Abu Sayyaf group” – the Abu Sayyaf group is one of the Muslim independence movements in Southern Philippines. In the absence of a direct answer, the corresponding search pattern can be transformed into: “Search for the attestation of the transfer of economic/financial items between the two”, which could lead to retrieve this information: “During 1998/1999, Abubakar Abdurajak Janjalani has received an undetermined amount of money from ObL through an intermediate agent”.

From a formal point of view, transformation rules are made up of a left-hand side, the ‘*antecedent*’ – i.e. the formulation, in search pattern format, of the ‘query’ to be transformed – and one or more right-hand sides, the ‘*consequent(s)*’ – the NKRL representation(s) of one or more queries (search patterns) that must be substituted for the given one. A transformation rule can, therefore, be expressed as: A (antecedent, left-hand side) $\Rightarrow B$ (consequent(s), right-hand side). The ‘transformation arrow’, ‘$\Rightarrow$’, has a double meaning:

- operationally speaking, the arrow indicates the *direction* of the transformation: the left-hand side A (the original search pattern) is removed and replaced by the right-hand side B (one or more new search patterns);
- the logical meaning of the arrow is that the information obtained through B *implies* (in a weak sense) the information we should have obtained from A.

Some formal details can be found in [4]. A representation of the previous ‘economic/financial transfer’ transformation is given in Table 5. Note that the left-hand side (antecedent) of this transformation corresponds to a partial instantiation of the template `(1.3112) Behave:FavourableConcreteMutual` that is routinely used to represent into NKRL format a (positive) mutual behaviour among two or more entities. With respect now to the implementation details, the `InferenceEngine` version to be used for transformations is quite identical to that used for executing the hypothesis rules. The sub-module `Antexec` (execution of the antecedent) corresponds, in fact, to the `Exeprem` sub-module; `Consexec` (execution of the consequent(s)) corresponds to `Execond`. `Reexec` is the same in the two versions.

Note that many of the transformation rules used in NKRL are characterized by the very simply format of Table 5 implying only one ‘consequent’ schema. An example of ‘multi-consequent’ transformation is given by this specific MoD rule: “In a context of ransom kidnapping, the certification that a given character is wealthy or has a professional role can be substituted by the certification that i) this character has a tight kinship link with another person (first consequent schema, *`conseq1`*), and ii) this second person is a wealthy person or a professional people (second consequent schema, *conseq2*)”. Let us suppose that, during the search for all the possible information linked with the Robustiniano Hablo's kidnapping, see occurrence `mod3.c5` in Table 1b above, we ask to the system whether Robustiano Hablo is wealthy. In the absence of a direct answer, the system will automatically ‘transform’ the original query using the above ‘kinship’ rule. The result is given in Figure 3: we do not know if Robustiano Hablo is wealthy, but we can say that his father is a wealthy businessperson.

Table 5. A simple example of ‘transformation’ rule

<u>‘economic/financial transfer’ transformation</u> :

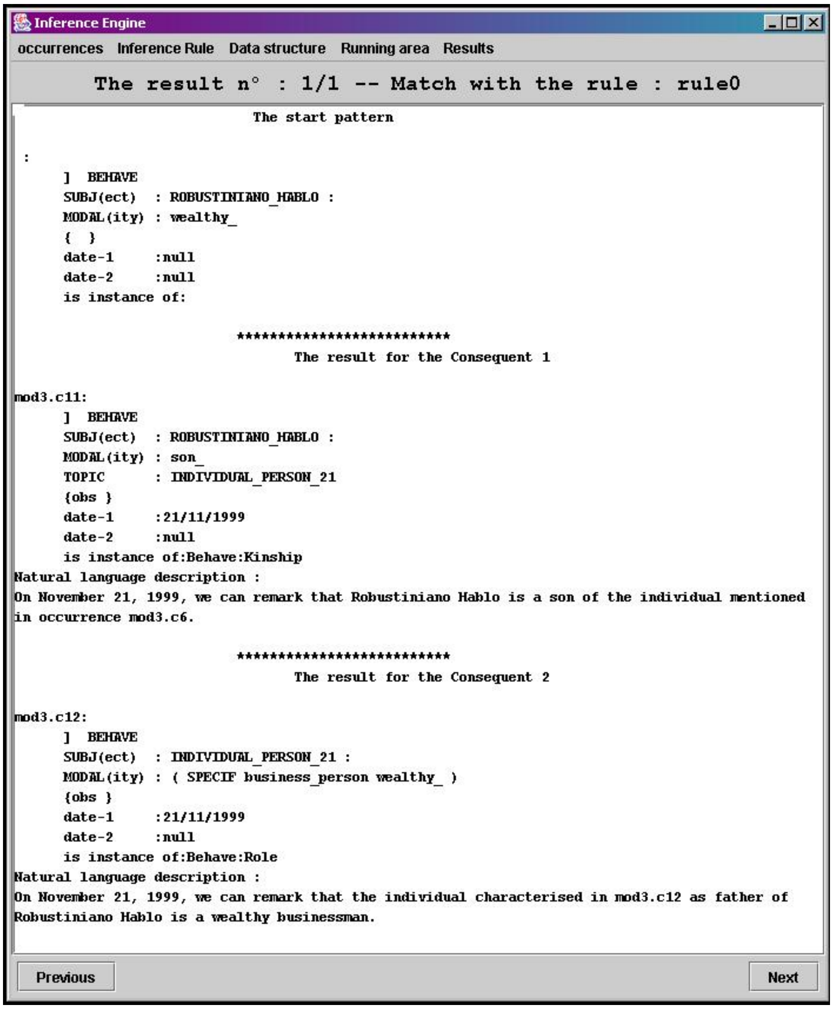
```
t1) BEHAVE  SUBJ    (COORD1 var1 var2)   ⇒   RECEIVE  SUBJ    var2
            OBJ     (COORD1 var1 var2)                OBJ     var4
            MODAL   var3                              SOURCE  var1

      var1  =  human_being_or_social_body
      var2  =  human_being_or_social_body
      var3  =  business_agreement, mutual_relationship
      var4  =  economic/financial_entity
```

To verify the existence of a relationship or of a business agreement between two (or more) persons, try to verify if one of these persons has received a ‘financial entity’ (e.g., money) from the other.

3 Integrating the Two Inferencing Modes of NKRL

As it appears from Table 3, a hypothesis rule corresponds to a *fixed scenario* formed by a given number of reasoning steps. Since these steps are represented as partially instantiated (standard) templates, and can then make use of variables and constraints, their formulation is relatively flexible and easy to generalize. For example, thanks to the variable *var2* introduced in the premise of hypothesis *h1* in Table 3 – and characterized by the two constraints `human_being` and `company_` – hypothesis *h1* is valid in the context of money received either from a *company* or from *private individuals*; adding further variables and constraints, generalization can be even improved. The reasoning steps to be executed are, however, *still rigidly predefined*.

To introduce a certain degree of *fuzziness* in the execution of hypotheses, and to increase the probability of discovering *implicit information* during this execution, we can make use of the concept of ‘transformation’. To be concretely executed, the reasoning steps of a hypothesis must, in fact, be reduced to search patterns and, at least in principle, any NKRL search pattern can be automatically (and semi-randomly) converted into a new search pattern by means of transformation rules. For this, it is sufficient that the original pattern unifies the antecedent part of one of the transformation rules present in the rule base of a given NKRL application. We know the importance of exploiting in depth, in a terrorism context, all the possible knowledge bases of known facts, see again [1]. Carrying out an exhaustive experimentation of the NKRL inference tools in the framework of the MoD application of the Parmenides project has then required the modification of the standard `InferenceEngine` to implement the possibility of making use of transformations in a hypothesis context; we will illustrate the characteristics of this work in the following Sections.


Inference Engine
occurrences Inference Rule Data structure Running area Results
Previous
Next

```
        The result n° : 1/1 -- Match with the rule : rule0
                          The start pattern

 :
      ]  BEHAVE
      SUBJ(ect)   : ROBUSTINIANO_HABLO :
      MODAL(ity)  : wealthy_
      {  }
      date-1      :null
      date-2      :null
      is instance of:

                        ***************************
                               The result for the Consequent 1

mod3.c11:
      ]  BEHAVE
      SUBJ(ect)   : ROBUSTINIANO_HABLO :
      MODAL(ity)  : son_
      TOPIC       : INDIVIDUAL_PERSON_21
      {obs }
      date-1      :21/11/1999
      date-2      :null
      is instance of:Behave:Kinship
Natural language description :
On November 21, 1999, we can remark that Robustiniano Hablo is a son of the individual mentioned
in occurrence mod3.c6.

                        ***************************
                               The result for the Consequent 2

mod3.c12:
      ]  BEHAVE
      SUBJ(ect)   : INDIVIDUAL_PERSON_21 :
      MODAL(ity)  : ( SPECIF business_person wealthy_ )
      {obs }
      date-1      :21/11/1999
      date-2      :null
      is instance of:Behave:Role
Natural language description :
On November 21, 1999, we can remark that the individual characterised in mod3.c12 as father of
Robustiniano Hablo is a wealthy businessman.
```

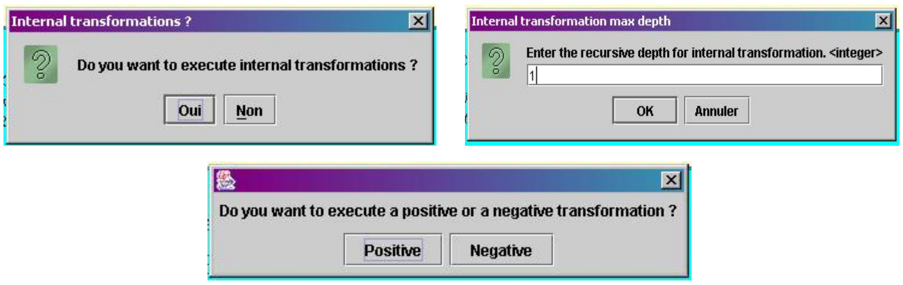
Fig. 3. `InferenceEngine` results corresponding to the application of the 'kinship' transformation to the query about Robustiniano Hablo's status

3.1 Strategies for the Execution of Transformations in a Hypothesis Context

Executing transformations in a hypothesis context means then taking a search pattern built up by `Execond` during the processing of a hypothesis rule and 'transforming' it by executing the following operations:

- after having created a list of all the transformation rules stored in the rule base of the system whose left hand side (antecedent) is generally congruent with the general format of the current search pattern, find in this list *one or more transformation rules, if any, whose antecedents can actually unify the original pattern*;
- in this case, *execute the transformation* by substituting to the original search pattern the new one formed using the right hand side (consequent) of the transformation, see Section 2.3 above;
- if the new search pattern is successful (i.e., if it can unify some occurrences of the knowledge base), store in VALAFF (in the hypothesis environment) some *new values for the variables* var_i *of the condition schema under execution.*

Several strategies can be adopted with respect to i) the 'number' of transformations to be executed and ii) the 'circumstances' of their execution, see [17] for the details. In the present version of the NKRL software, the user is allowed to select a particular transforming strategy, see Figure 4. After having answered 'Yes' to the question about the execution of 'internal' transformations (i.e., the execution of transformation processes in a hypothesis context), on the left, she/he is asked whether she/he wants to carry out *'positive' or 'negative' transformations*. Conventionally, executing 'negative' transformations means that the transformation process is activated *only after the failure of a search pattern derived directly from a hypothesis condition schema* (i.e., a search pattern able to find an unification in the knowledge base cannot be transformed); executing 'positive' transformations means that *all the search patterns* derived directly from a hypothesis condition schema *are transformed*, independently from the fact they have been successful or not. The last query on the right of Figure 4 asks for the *'depth' of the transformation*, i.e., it asks if a pattern that is the result of an internal transformation can be transformed in turn, and how many 'transformation steps' are allowed.


Internal transformations ?

Do you want to execute internal transformations ?

Internal transformation max depth

Enter the recursive depth for internal transformation. <integer>

1

Do you want to execute a positive or a negative transformation ?

Fig. 4. Parameters for internal transformations

3.2 The Correspondence Among Variables

The main 'theoretical' problem concerning the integration hypotheses/transformations concerns the possibility of finding a *correspondence* between i) the variables, var_h, originally present in the (hypothesis) condition schema from which the search pattern to transform has been derived and ii) those, totally disjoint, var_t, that appear in the transformation rules to be used. We recall in fact that, see 2.3 above, the values found by Fum for the variables var_h of a given condition schema, and stored in VALAFF and DESVAR, will be utilized to build up new search patterns from the subsequent condition schemata in order to continue with the processing of the hypothesis. Using a transformation must allow *finding new sets of values for* var_h; the problem concerns the fact that these new values *will not refer directly to* var_h *but to* var_t, the set of variables proper to a transformation rule that, applied to one of the search patterns derived from the original condition schemata, was successful. A correspondence among var_h and var_t must then, in general, be found.

We must now operate a distinction between '*global*' and '*local*' variables. In the context of the 'integration' problem, a 'local' variable is a variable that is used in *only one* condition schema, $cond_i$, without appearing explicitly in any of the subsequent condition schemata $cond_{i+1}$ ... $cond_n$. A global variable, on the contrary, is a variable that, after having been introduced in $cond_i$, is *utilized again* in at least one of the following $n-i$ condition schemata. Note that the variables introduced by the premise are not interested by this distinction, given that the search patterns derived from the premise are never submitted to the transformation operations: if these search patterns cannot unify the starting occurrence, this means that the hypothesis is not suitable for explaining this occurrence. To give an example, variables var3 and var4 introduced the cond1 scheme of Table 3 are *global*, because they are re-utilized in the cond2 scheme; var5 in cond2 is *local*, given that cond2 is the last condition scheme of the hypothesis. We can now say that:

- If, independently from the fact that the search patterns derived from a condition schema $cond_i$ were successful or not, the *new variables* introduced by $cond_i$ are *all local* – i.e., none of them appear in the subsequent $n-i$ condition schemata – this means that, when transforming the search patterns derived from $cond_i$, *the new values for the variables* var_h *of* $cond_i$ *obtained through the transformation process are not explicitly required in the processing of the subsequent schemata of the hypothesis*. In this case, if the original $cond_i$ search patterns failed, a successful transformation (a 'negative transformation', see the previous Section) is nevertheless necessary to allow the hypothesis to continue. If the $cond_i$ search patterns were successful, the success of the transformation (positive transformation) is only useful i) to guarantee, in an indirect way, that $cond_i$ is congruent with the data of the knowledge base and, more importantly, ii) to introduce in the hypothesis' answer some new information (occurrences) to confirm/expand the 'normal' results.
- If some of the *new variables* introduced by $cond_i$ are *global* – i.e., they appear in at least one of the subsequent $n-i$ condition schemata – associating new values with these variables through a successful transformation process will be *absolutely mandatory* to allow the hypothesis to continue *in case of failure of the original* $cond_i$ *search patterns* (negative transformation). In case of success of these search patterns, the success of the transformation (positive transformation) will increase the possibility of constructing *new search patterns* from the subsequent condition schemata making use of the new values for var_h, enlarging considerably the search space and augmenting then the probability of obtain-

ing new interesting results. In both these cases, a *correspondence* between var_h (the variables of the original condition schema) and var_t (the variables used in the transformation) *must necessarily be found.*

To find the correspondence, the following guidelines must be used:

- To build up a generic search pattern sp_h from a condition schema, the variables var_h of this schema must be *replaced* (see Section 2.3 above) by some values val_h, concepts or individuals, corresponding to values obtained from the previous steps of the hypothesis or to constraints originally associated with var_h (these constraints are stored and managed in $\mathrm{RESTRICT}_{hypo}$). *The association* $var_h \equiv val_h$ *is stored in* $VALAFF_{hypo}$.
- When trying to use a transformation rule to modify sp_h, the transformation variables var_t included in the antecedent must be replaced in turn by some of the constraints stored in $\mathrm{RESTRICT}_{transfo}$ to create a search pattern sp_t; the values val_t provisionally associated with var_t are stored in $\mathrm{VALAFF}_{transfo}$. Fum can now be called to (try to) unify sp_t, the search pattern derived from the antecedent, with sp_h, the search pattern (derived from a condition schema) to transform. *The unification is oriented from* sp_t *towards* sp_h, i.e., sp_h plays the role normally associated to a 'predicative occurrence' in the 'standard' Fum unification, see Section 2.2.
- If the unification succeeds, *new values* val_t are associated with var_t in the transformation environment, $var_t \equiv val_t$, and stored in $\mathrm{VALAFF}_{transfo}$. Since the unification is successful (the data structures of sp_h and sp_t are fully congruent), the values val_t, retrieved on sp_h, correspond *necessarily* to the values val_h substituted to var_h to create sp_h: these values have been stored in VALAFF_{hypo}. *The equality* $val_t = val_h$ *determined through the comparison of the values stored in* $VALAFF_{transfo}$ *and* $VALAFF_{hypo}$ *implies then the correspondence* $var_t = var_h$ that we wanted to retrieve. It will then be possible i) to transfer back to the original hypothesis environment the values bound to var_t through the unification of the search patterns derived from the consequent of the transformation with the occurrences of the base, and ii) to associate these values with var_h to continue with the processing of the original hypothesis.

3.3 Some Examples

In Table 6b, we reproduce part of a typical 'terrorism' hypothesis used in the Parmenides project, namely, the hypothesis that allows explaining, among other things, the kidnapping of Robustiniano Hablo (Table 6a and Table 1 above) essentially in terms of:

- The victim has been kidnapped by a terrorist or separatist group (e.g., the Abu Sayyaf Group).
- This group carries out a particular form of kidnapping, *kidnapping for ransom.*
- This form of kidnapping implies that the kidnapped person is either wealthy or a VIP.
- It is then necessary to check then whether the victim is wealthy or is a VIP.

Table 6. The 'kidnapping for ransom' hypothesis rule

a)

```
mod3.c5) PRODUCE  SUBJ     (SPECIF INDIVIDUAL_PERSON_20 weapon_wearing
                           (SPECIF cardinality_ several_)):(VILLAGE_1)
                  OBJ      kidnapping_
                  BENF     ROBUSTINIANO_HABLO
                  CONTEXT  #mod3.c6
                  date-1:  20/11/1999
                  date-2:
```

b)
HYPOTHESIS *h2*

<u>premise</u> :

```
PRODUCE  SUBJ   var1
         OBJ    kidnapping_
         BENF   var2

var1 = human_being_or_social_body
var2 = individual_person
```

A human being has been kidnapped.

<u>first condition schema (*cond1*)</u> :

```
BEHAVE   SUBJ   var1
         MODAL  member_of
         TOPIC  var3

var1 = human_being_or_social_body
var3 = separatist_mouvement, terrorist_organization
```

The kidnappers are member of a separatist movement or of a terrorist organization.

<u>second condition schema (*cond2*)</u> :

```
PRODUCE  SUBJ   var3
         OBJ    var4

var3 = separatist_mouvement, terrorist_organization
var4 = ransom_kidnapping
```

This organization performs ransom kidnapping.

…

Note that, to perform the last reasoning step when trying to explain the Robustiniano Hablo's kidnapping, we must use, in a hypothesis context, the 'two steps' (multi-consequent) 'kinship' transformation mentioned in Section 2.4 above, see also Figure 3. This fact is a further confirmation of the interest of coupling transformations and hypotheses to broaden the explanatory power of hypotheses.

If we examine now `cond2` in Table 6b – which translates the second reasoning step mentioned above, i.e., "this group carries out kidnapping for ransom" – we can see that the unique new variable introduced by this condition schema is *var4* (*var3* has been introduced in *cond1*); *var4* is *'local'*, in the sense that it will not be used in the subsequent condition schemata. We can now suppose to have, among the transformations of the system, the transformation rule reproduced in Table 7b. Using the 'normal' version of `InferenceEngine` for transformations, the sp_t pattern corresponding to the antecedent part of this rule will unify the sp_h pattern derived from *cond2* in Table 6b (see Table 7a and the upper part of Table 8), where *var3* has been replaced by `ABU_SAYYAF_GROUP` – we can suppose that this value has been obtained during the processing of *cond1* – and *var4* by the constraint `ransom_kidnapping`. The execution of the transformation will eventually produce, from the consequent of the transformation, the search pattern reproduced in the lower part of Table 8: we will call this type of pattern sp_f, 'final pattern', given that it will be used to search for unifications within the knowledge base of predicative occurrences as in the 'normal' processing of hypotheses.

Table 7. A possible transformation rule for *cond2* of *h2*

a)
search pattern sp_h derived from *cond2* of Table 6b :

```
PRODUCE
SUBJ : ABU_SAYYAF_GROUP :
OBJ : ransom_kidnapping :
{}
date1 :
date2 :
```

The Abu Sayyaf group performs ransom kidnapping.

b)
'ransom kidnapping' transformation :

```
t2) PRODUCE  SUBJ  var1  ⇒  RECEIVE  SUBJ    var1
             OBJ   var2              OBJ     money_
                                     SOURCE  var3
                                     TOPIC   (SPECIF
                                              captivity_freeing var4)

    var1  =  separatist_mouvement, terrorist_organization
    var2  =  ransom_kidnapping
    var3  =  human_being
    var4  =  human_being
```

To verify if a given organization performs ransom kidnapping, try to see this organization has received some money for freeing from captivity one or more human being(s).

If this sp_f pattern can unify some information in the knowledge base, *the success of the transformation will only give rise to 'variants' of the 'normal' results, if any*: the evidence of the fact that the Abu Sayyaf Group carries out ransom kidnapping will be reinforced by a specific information telling us that the Abu Sayyaf Group has received some money for freeing people from captivity. In this case, no new values will be introduced in the `VALAFF` and `DESVAR` variables of the *h2* hypothesis, that will then continue unchanged with the processing of *cond3*.

Table 8. The original search pattern sp_h derived from *cond2* and the final sp_f obtained from the consequent of *t2*

```
PRODUCE
SUBJ : ABU_SAYYAF_GROUP :
OBJ : ransom_kidnapping :
{}
date1 :
date2 :

RECEIVE
SUBJ : ABU_SAYYAF_GROUP :
OBJ : money_ :
SOURCE : human_being :
TOPIC : (SPECIF captivity_freeing human_being)
{}
date1 :
date2 :
```

If we consider now the *cond1* condition schema of hypothesis *h2*, see Table 6b, we can see that the (unique) new variable introduced in this schema is *var3*: this one is now '*global*' given that it will be used in *cond2* and the following condition schemata. The transformations operating on the search patterns sp_h derived from *cond1* will then be able, in principle, to produce 'new' values for *var3* to be stored in VALAFF/DESVAR and *to be used as they had been obtained through the 'normal' hypothesis operations* – 'new' means values that can be different with respect to those obtained via the usual procedures. At the difference then of the previous case ('*local*' variables) – where the execution of successful transformations in a hypothesis context could only lead to produce 'locally' (i.e., for the condition schema actually transformed) some 'variants' of the standard outcomes – the success of the 'internal' transformations *could* now lead to the addition of totally new branches to the choice tree, likely to produce new results for each of the condition schemata included between the transformation point and the end of the hypothesis.

A problem of '*variable correspondence*' occurs now, to be solved according to the 'guidelines' illustrated at the end of the previous Section. Let us suppose to make use of the transformation represented in Table 9: i.e., the membership in an organization/political group/party can be verified checking, among other things, if the 'member' receives some form of permanent or occasional 'salary' from the organization.

In this case, after having entered the specific transformation environment and having activated the sub-module ANTEXEC (see Section 2.4 above) to execute, using Fum, the unification between the search pattern sp_t derived from the antecedent of *t3* in Table 9 and the sp_h pattern corresponding to *cond1* of *h2*, we are confronted with the situation globally represented in Table 10. The top search pattern (a, sp_h) is the pattern derived from *cond1*. Returning, in fact, to Table 6 above, we can see that – because of the unification, in the original hypothesis environment and using EXEPREM, of the premise of *h2* with the starting occurrence mod3.c5 – the variables $var1_{hypo}$ and $var2_{hypo}$ introduced by the premise have taken, respectively, the values INDIVIDUAL_PERSON_20 and ROBUSTINIANO_HABLO, as reflected by the state of the environment variable $\mathtt{VALAFF}_{\text{hypo}}$ in Table 10. The value `separatist_movement` associated with $\mathtt{var3}_{hypo}$ in $\mathtt{VALAFF}_{\text{hypo}}$, see Table 10, derives from the operations performed by `EXEPREM` (in the hypothesis environment) to build up the search pattern ($\mathtt{a}$, sp_h) from *cond1*. The pattern ($\mathtt{b}$, sp_t) has been built up by `ANTEXEC` (in the transformation environment) from the antecedent part of the transformation *t3* of Table 9; after the unification of ($\mathtt{b}$, sp_t) and ($\mathtt{a}$, sp_h) – this unification, executed by `Fum` is, as already stated, 'oriented', in the sense that ($\mathtt{a}$, sp_h) has a 'predicative occurrence' role – the variables $\mathtt{var1}_{transfo}$ and $\mathtt{var2}_{transfo}$ have, respectively, the values `INDIVIDUAL_PERSON_20` and `separatist_movement`, see the status of `VALAFF` for the transformation environment in Table 10.

Table 9. A possible transformation rule for *cond1* of *h2*

'permanent or occasional salary' transformation :

```
t3)   BEHAVE   SUBJ    var1         ⇒     RECEIVE    SUBJ     var1
               MODAL   member_of                     OBJ      var3
               TOPIC   var2                          SOURCE   var2

      var1   =   human_being
      var2   =   political_group/party
      var3   =   irregular_payment, salary_
```

To verify if a person is a member of a given organization, try to see if, among other things, this person receives a permanent or occasional salary from this organization.

According to the guidelines, from the comparison of the values affected to the variables in the two versions of `VALAFF`, for hypotheses ($\mathtt{VALAFF}_{\text{hypo}}$) and transformations ($\mathtt{VALAFF}_{\mathtt{transfo}}$) – this comparison is entrusted to `Fum` – we can deduce that there *is a correspondence between* $\mathtt{var3}_{hypo}$ *and* $\mathtt{var2}_{transfo}$. This fact will be then registered into a `CORRESP(ondence)` table, see Table 11: only $\mathtt{var3}_{hypo}$ is of interest for the continuation of the hypothesis. After the (successful) unification of ($\mathtt{b}$, sp_t)/($\mathtt{a}$, sp_h), *the processing of the transformation in its proper environment will continue*, and the `CONSEXEC` sub-module will then build up the 'final' search pattern ($\mathtt{c}$, sp_f) of Table 10 from the consequent of *t3* of Table 9.

Let us suppose now that ($\mathtt{c}$, sp_f) of Table 10 is able to unify (at least) a predicative occurrence in the NKRL knowledge base. In this case, a value will be bound to $\mathtt{var2}_{transfo}$; according to the information stored in `CORRESP`, see Table 11, this value will also be bound to $\mathtt{var3}_{hypo}$ and inserted into the `VALAFF/DESVAR` variables of the original hypothesis (*h2*) environment. If we assume now for generality's sake, see Section 3.1 above, that the strategy chosen for the execution of the 'internal' transformations is a 'positive' one – i.e., all the search pattern built up by `EXECOND` from a condition can be transformed independently from the fact that these patterns have been successful or not – four different cases can be envisaged.

Table 10. Data structures and environment variables after the unification of the patterns derived from `cond1` and `t3`

```
(a, sph) BEHAVE
SUBJ : INDIVIDUAL_PERSON_20 :
MODAL : member_of :
TOPIC : separatist_movement
{}
date1 :
date2 :

VALAFFhypo (h2)
var1hypo = INDIVIDUAL_PERSON_20
var2hypo = ROBUSTINIANO_HABLO
var3hypo = separatist_movement

(b, spt) BEHAVE
SUBJ : human_being :
MODAL : member_of :
TOPIC : political_group/party
{}
date1 :
date2 :

VALAFFtransfo (t3)
var1transfo = INDIVIDUAL_PERSON_20
var2transfo = separatist_movement

(c, spf) RECEIVE
SUBJ : INDIVIDUAL_PERSON_20 :
OBJ : irregular_payment :
SOURCE : separatist_movement :
{}
date1 :
date2 :
```

Table 11. The *CORRESP* table for the example

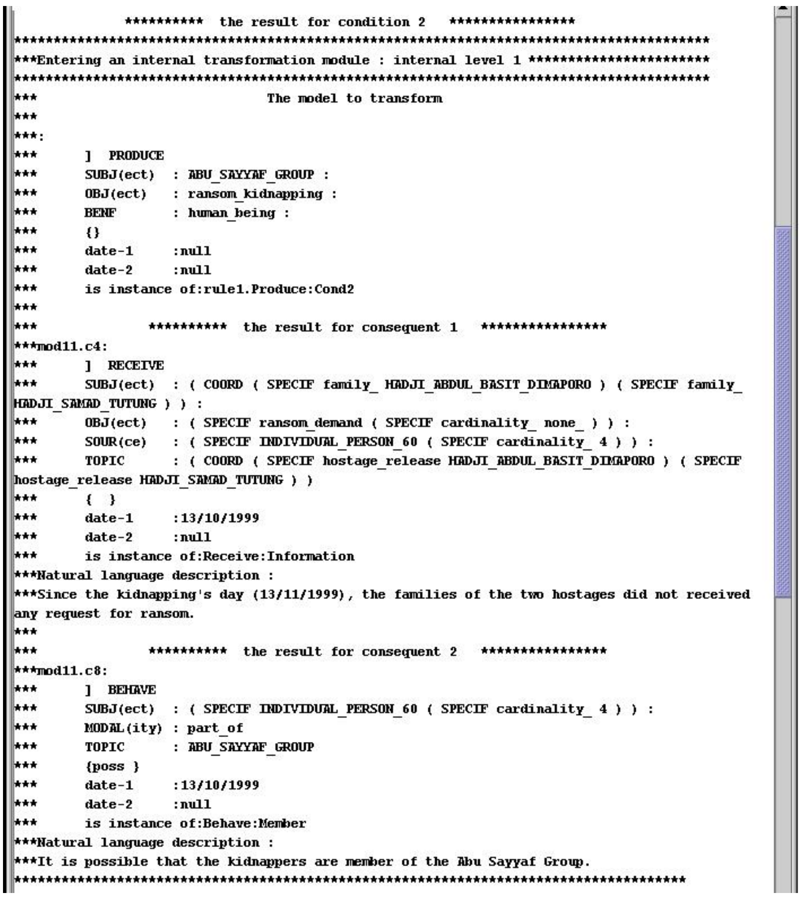
original condition schema: variable name	*value*	*internal transformation: variable name*
var3	separa-tist_movement	*var2*

- In the first one, we suppose that $var3_{hypo}$, before the execution of the transformation, was already linked with ABU_SAYYAF_GROUP – i.e., during the ‘normal’ execution of the hypothesis *h2*, a search pattern directly derived from the condition schema *cond1* has been able to retrieve the information that INDIVIDUAL_PERSON_20 (representing collectively the group of person that has realized the kidnapping) was part of the Abu Sayyaf group. We suppose now that the final transformed search pattern $(c,\ sp_f)$ of Table 10 retrieves the information telling that INDIVIDUAL_PERSON_20 receives some form of occasional salary from the Abu Sayyaf group: $var2_{transfo}$ is also bound to ABU_SAYYAF_GROUP, and passing this value to $var3_{hypo}$ will add nothing from the point of view of an 'augmented' development of the hypothesis. As in the previous case of 'local' variables, the only real benefit linked with the execution of the transformation will be a confirmation of the links between INDIVIDUAL_PERSON_20 and the Abu Sayyaf Group through the discovery that the kidnappers receive also some money from this group.

- If we suppose now that, in the 'normal' execution of the *h2*, all the search patterns directly derived from the condition schema *cond1* failed, the possibility of obtaining the value ABU_SAYYAF_GROUP for $var3_{hypo}$ via the transformation and the passage through $var2_{transfo}$ permits, on the contrary, to continue with the processing of the hypothesis *h2* otherwise irremediably destined to fail.
- We can suppose now that pattern (c, sp_f) of Table 10 is able to find an unification within the knowledge base telling us that INDIVIDUAL_PERSON_20 receives some form of occasional salary *from another group*, e.g., from the Moro Islamic Liberation Front (another Muslim separatist group in the Southern Philippines) – MORO_ISLAMIC_LIBERATION_FRONT will be then, in this case, the final value bound to $var2_{transfo}$ after the unification with the occurrences of the base. If, as in the previous case, $var3_{hypo}$ was 'empty' before the execution of the transformation, activating the process of internal transformation will allow again continuing with the processing of a hypothesis *h2* otherwise destined to fail.
- Eventually, let us suppose that $var2_{transfo}$ takes the value MORO_ISLAMIC_LIBERATION_FRONT *while* $var3_{hypo}$ *is already bound to ABU_SAYYAF_GROUP via the 'normal' hypothesis processing*. This means, in practice, that the group of kidnappers is linked in some way to both the Abu Sayyaf and Moro Islamic Liberation Front groups. According to the correspondence between $var2_{transfo}$ and $var3_{hypo}$ registered in CORRESP, MORO_ISLAMIC_LIBERATION_FRONT must also be bound to $var3_{hypo}$ and stored, accordingly, into the VALAFF/DESVAR variables of the *h2* environment. This new value will be used in the further processing of the hypothesis in parallel with the original one (ABU_SAYYAF_GROUP), leading then (possibly) to a totally new and particularly interesting set of results.

3.4 Additional Examples

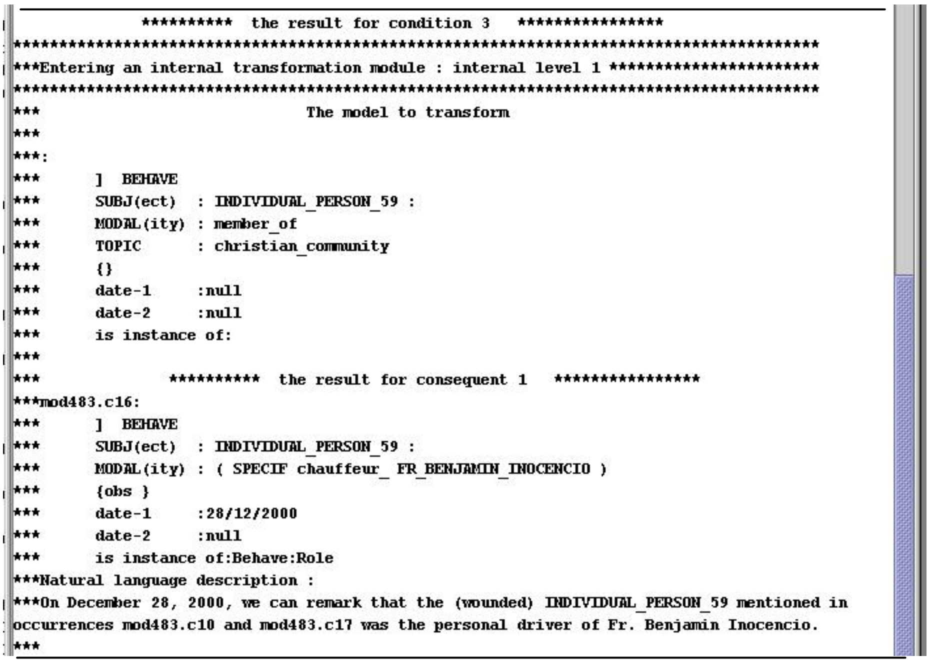
From a practical point of view, the 'positive' strategy evoked in the previous Section is not often employed, because it can be particularly exciting in terms of results, but can also be very computationally expensive. To give only an example, let us suppose to systematically adopt a 'positive' strategy for hypothesis *h2* of Table 6 above: Figure 5 shows what happens at the level of the condition schema *cond2* of *h2* when, after having directly found in a hypothesis context that a given terrorist group/separatist movement practices ransom kidnapping, we ask to the system to transform anyway the corresponding (successful) search patterns to retrieve the same notion in an indirect way. The transformation shown in Figure 5 is – on the contrary of transformation *t2* used in Table 7 for the same condition schema *cond2* of *h2* – a 'two steps' (multi-consequent) transformation, which is part of a (numerous) family of transformations that can be all reduced to the same principle: to demonstrate that a given terrorist/separatist group practices ransom kidnapping, we will try to find evidence that members of this group are involved in actions related to ransom kidnapping. In the case of Figure 5, we can then found, thanks to the transformation, an additional complex indirect result saying that: i) the families of two hostages, Hadji Abdul Basit Dimaporo and Hadji Samad Tutung, did not yet receive a request for ransom from the four individual that have kidnapped their relatives, and ii) the four kidnappers are known as members of the Abu Sayyaf group.



```
            **********  the result for condition 2   ****************
*****************************************************************************************
***Entering an internal transformation module : internal level 1 ***********************
*****************************************************************************************
***                              The model to transform
***
***:
***       ]  PRODUCE
***       SUBJ(ect)   : ABU_SAYYAF_GROUP :
***       OBJ(ect)    : ransom_kidnapping :
***       BENF        : human_being :
***       {}
***       date-1      :null
***       date-2      :null
***       is instance of:rule1.Produce:Cond2
***
***               **********  the result for consequent 1   ****************
***mod11.c4:
***       ]  RECEIVE
***       SUBJ(ect)   : ( COORD ( SPECIF family_ HADJI_ABDUL_BASIT_DIMAPORO ) ( SPECIF family_
HADJI_SAMAD_TUTUNG ) ) :
***       OBJ(ect)    : ( SPECIF ransom_demand ( SPECIF cardinality_ none_ ) ) :
***       SOUR(ce)    : ( SPECIF INDIVIDUAL_PERSON_60 ( SPECIF cardinality_ 4 ) ) :
***       TOPIC       : ( COORD ( SPECIF hostage_release HADJI_ABDUL_BASIT_DIMAPORO ) ( SPECIF
hostage_release HADJI_SAMAD_TUTUNG ) )
***       { }
***       date-1      :13/10/1999
***       date-2      :null
***       is instance of:Receive:Information
***Natural language description :
***Since the kidnapping's day (13/11/1999), the families of the two hostages did not received
any request for ransom.
***
***               **********  the result for consequent 2   ****************
***mod11.c8:
***       ]  BEHAVE
***       SUBJ(ect)   : ( SPECIF INDIVIDUAL_PERSON_60 ( SPECIF cardinality_ 4 ) ) :
***       MODAL(ity)  : part_of
***       TOPIC       : ABU_SAYYAF_GROUP
***       {poss }
***       date-1      :13/10/1999
***       date-2      :null
***       is instance of:Behave:Member
***Natural language description :
***It is possible that the kidnappers are member of the Abu Sayyaf Group.
*************************************************************************************
```

Fig. 5. Members of the Abu Sayyaf's group practice ransom kidnapping

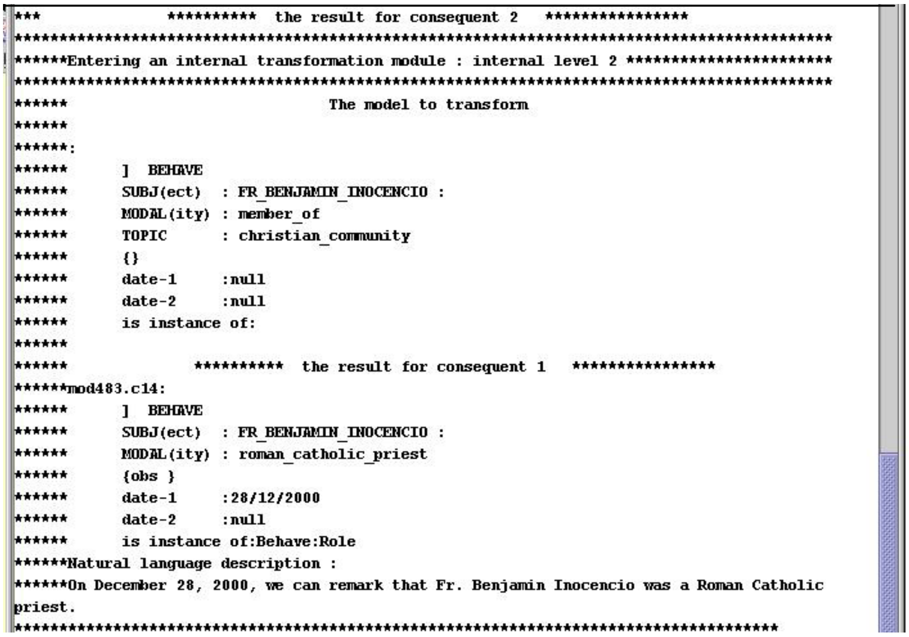
As a last example of use of transformations in a hypothesis context, let us suppose that, after having clicked the 'negative' button, see the lower level of Figure 4, we introduce '2' as an answer to the request about the transformation's depth, Figure 4 right: we then accept that, after having obtained a new search pattern by the transformation process, this last *can pass in turn* through the transformation procedures. By using a specific rule of the MoD application (very easy indeed to generalize), we want now to retrieve the reasons that can have led a given `INDIVIDUAL_PERSON_59` to be injured by members of the Abu Sayyaf group. The rule suggests checking whether the injured/killed person was a member of the Christian community in Southern Philippines: members of this community are, in fact, often the targets of Muslim separatist's attacks. This rule is a very simple one and implies three reasoning steps where the last, *cond3*, consists in verifying the Christian community membership of the offended person: this fact cannot be demonstrated directly for `INDIVIDUAL_PERSON_59`. A first transformation to be used corresponds to the following common sense argument: "A given person can be considered as a 'member



```
              **********  the result for condition 3   ****************
*****************************************************************************************
***Entering an internal transformation module : internal level 1 ***********************
*****************************************************************************************
***                          The model to transform
***
***:
***       ] BEHAVE
***       SUBJ(ect)   : INDIVIDUAL_PERSON_59 :
***       MODAL(ity)  : member_of
***       TOPIC       : christian_community
***       {}
***       date-1      :null
***       date-2      :null
***       is instance of:
***
***                **********  the result for consequent 1   ****************
***mod483.c16:
***       ] BEHAVE
***       SUBJ(ect)   : INDIVIDUAL_PERSON_59 :
***       MODAL(ity)  : ( SPECIF chauffeur_ FR_BENJAMIN_INOCENCIO )
***       {obs }
***       date-1      :28/12/2000
***       date-2      :null
***       is instance of:Behave:Role
***Natural language description :
***On December 28, 2000, we can remark that the (wounded) INDIVIDUAL_PERSON_59 mentioned in
occurrences mod483.c10 and mod483.c17 was the personal driver of Fr. Benjamin Inocencio.
***
```

Fig. 6. First level of transformation for the wounding of `INDIVIDUAL_PERSON_59`

at large' of a given (e.g., Christian) community whether i) it can be proved that he has a very strict employment relationship (e.g., a `domestic_role`) with a second person (first consequent schema), and ii) this second person is known to be part of this community (second consequent schema)". The first step of this transformation can be directly satisfied by retrieving that `INDIVIDUAL_PERSON_59` is the chauffeur of a catholic priest, Fr. Benjamin Inocencio, Figure 6; the second step requires, however, to find out an *explicit proof* of the fact that this second person is a member of the Christian community. Proving the membership is obtained by passing through a further one step transformation: this will specify that being a roman catholic priest is equivalent to being part of the (larger) Christian community, see Figure 7.



```
***              **********  the result for consequent 2   ****************
*************************************************************************************************
******Entering an internal transformation module : internal level 2 **********************
*************************************************************************************************
******                              The model to transform
******
******:
******       ]  BEHAVE
******       SUBJ(ect)   : FR_BENJAMIN_INOCENCIO :
******       MODAL(ity) : member_of
******       TOPIC       : christian_community
******       {}
******       date-1      :null
******       date-2      :null
******       is instance of:
******
******               **********  the result for consequent 1   ****************
******mod483.c14:
******       ]  BEHAVE
******       SUBJ(ect)   : FR_BENJAMIN_INOCENCIO :
******       MODAL(ity) : roman_catholic_priest
******       {obs }
******       date-1      :28/12/2000
******       date-2      :null
******       is instance of:Behave:Role
******Natural language description :
******On December 28, 2000, we can remark that Fr. Benjamin Inocencio was a Roman Catholic
priest.
*******************************************************************************************
```

Fig. 7. Second level of transformation for the wounding of `INDIVIDUAL_PERSON_59`

4 Some Remarks About the Software Solutions

Integrating the two versions, for hypotheses and transformations, of `InferenceEngine`, corresponds to solve a complex 'coroutine' problem, where the main difficulty is generated, as usual in NKRL, by the existence of complex data structures to be managed, stored and reloaded. The integration can be implemented according to two approaches:

- The first one is the classical 'coroutine' solution, where the 'transformation' version of the engine has the same priority of the 'hypothesis' version, and then it starts its execution, as the latter one, from the main method of `InferenceEngine`. The most important difficulty linked with this solution concerns the fact that it should imply the complete execution of the transformation version (the 'internal' version) until a result had been obtained, requiring then an efficient way – Java's threads could be used in this context – to return to the hypothesis version (the 'external' one). On the other hand, this solution should also imply the fact of having a separate display (text output) for the results of the 'internal' execution, with some difficulties then in co-ordinating them with the displaying of the 'external' (hypothesis) results.
- The second approach, more manageable and simpler to implement, consists in just integrating the `InferenceEngine` *Java object* corresponding to the 'internal' (transformation) version in the execution of the 'external' (hypothesis) version. This allows the external hypothesis version of `InferenceEngine` – which works now as the 'main' program – to run the internal version until it has

a result and to get back this result simply as a function execution return. This approach allows implementing a 'transparent' running of the internal module, and implies also the advantage of having the results of the 'internal' execution naturally integrated with those displayed by the interface of the main (hypothesis) version.

We have then chosen the second approach. Giving that, i) during the functioning of `InferenceEngine`, `Reexec` is the only sub-module invoked in order to reconstruct the environment proper to a previous choice point and to allow then `Execond` to build up a new search pattern, and that ii) executing transformation operations – in our case, within a hypothesis context – amounts exactly to build up new search patterns, we can conclude that only `Reexec` must be modified to implement the 'integration' operations. In practice, during the execution of an internal transformation, `Reexec` will run the 'internal' (transformation) Java object until a result has been found and returned: the internal object is then stopped, and it will wait for a new `Reexec` call, producing other results if these are possible for the current transformation. For each call to `Reexec`, there will be then one internal module pending and waiting for further results – each main program execution level will potentially have an internal module object reference pointing to such object. From a Java programming point of view, `InferenceEngine` includes now *three different objects*: `Hypothesis`, `Transformation` and `InternalTransformation`. The first two are practically unchanged. The `InternalTransformation` object is modelled on the `Transformation` one: as already stated, it will run trying to find a result and then, if successful, it will return this result; if this is not possible, it will throw a `NoMoreResultsException()`. `Reexec` now executes first the `InternalTransformation` code to get the next result and, if this is not possible, catches the exception and continues its execution trying to build a new model for this level. See [17] for more information and a detailed example.

5 Future Work

NKRL is a fully implemented language/environment. The software exists in two Java-2 versions, an ORACLE-supported and a file-oriented one. The reasons that justify the existence of a file-oriented version are mainly the following:

- The possibility of running a quite-complete version of the NKRL software on machines unable to support a full-fledged version of ORACLE, e.g., low-range portable computers.
- The possibility of accelerating the processing of the inference procedures – in particular, the most complex inferencing operations involving a co-ordinated running of 'hypothesis' and 'transformation' rules.

With reference to the last point, we can note that a certain 'sluggishness' of the inference procedures in the standard ORACLE version – up to a few minutes to get a result on both an AMD Athlon XP 2100+ 1.73 GHz with 512 MB of RAM and a Pentium M 735 1.7 GHz with 1.0 GB of RAM when transformations and hypotheses are running together; the answer is, on the contrary, almost immediate when the two classes of rules are processed separately – is not a default in itself, given that this integrated processing must be conceived more as a powerful tool for discovering all the possible implicit relationships among the data in the knowledge base than as the support of a standard question-answering system. However, there are situations – e.g., demos – where an immediate answer can be of interest: few seconds are needed to get a result *in the file-oriented environment* even when the most complex examples of hypothesis/transformation combinations are running.

With respect now to the possible improvements, some of them are mainly of a ‘cosmetic’ nature. For example, many of the visualization features (including the visualization of the results of the inference rules, see the Figures in the previous Sections) are inherited from ‘old’ software developed in previous European projects: they are somewhat ‘ugly’ and do not do justice to the complexity and interest of the results. More substantial improvements will concern firstly:

- The addition of features that will allow querying the system in Natural Language, i.e., features that will implement an automatic ‘translation’ of NL queries into ‘external’ search patterns like that of Table 1c. Very encouraging experimental results have already been obtained in this context thanks to the use of shallow parsing techniques (like the AGFL grammar and lexicon, see [18]), coupled with simple ‘production-like rules’ used to generate the templates and with the help of the standard NKRL inference capabilities. With respect to the more general and difficult problem of automatically producing occurrences and annotations from NL descriptions of the original information, see the comments at the end of Section 2.1.
- The introduction of optimisation techniques for the (basic) chronological backtracking of the NKRL `InferenceEngine`, in the style of the well-known techniques developed in a Logic Programming context see, e.g., [19]. This should allow, among other things, to align the processing time of the inference rules in the ORACLE version with that of the file-oriented version of the software, see above.

6 Comparison with Similar Approaches

Comparison of what expounded in the previous Sections with work accomplished in a Semantic Web framework – Semantic Web (SW) is today a very popular paradigm in the Knowledge Representation and Reasoning domain – is not very easy because of a fundamental, ‘epistemological’ difference between the NKRL and the SW approaches.

Semantic Web languages like RDF and OWL are, in fact, inherently ‘binary’, in the sense that, for these languages, a property can only be a binary relationship, linking two individuals or an individual and a value. The (very scarce) proposals intended to extend the SW languages to deal with *n*-ary relationships like those dealt with by NKRL are not very convincing. For example, a recent working paper from the W3C Semantic Web Best Practices and Deployment Working Group (SWBPD WG) about defining *n*-ary relations for the SW languages, see [20], proposes some extensions to the binary paradigm to allow the correct representation of ‘narratives’ like: “Christine has breast tumor with high probability”, “Steve has temperature, which is high, but failing" or "John buys a 'Lenny the Lion' book from books.Example.com for $15 as a birthday gift". The solutions proposed, really questionable, range from the introduction of fictitious 'individuals' to represent the *n*-ary relations to the rediscovery of some semantic networks solutions of the seventies. The SW languages seem then to be relegated, fundamentally, to the set up and management of *static* 'ontologies' of concepts and individuals.

Unfortunately, this corresponds to leave aside a big amount of important, 'economically relevant' information, which is buried into unstructured 'narrative' information resources (or 'narratives'): most of the corporate knowledge documents (memos, policy statements, reports, minutes etc.), the news stories, the normative and legal texts, the medical records, many intelligence messages, etc., as well as, in general, a huge fraction of the information stored on the Web deal in fact with narratives. In these 'narrative documents', or 'narratives', the main part of the information content consists in the *dynamic* description of 'events' that relate the real or intended behaviour of some 'actors' (characters, personages, etc.) – as already stated, see Section 2.1, the term 'event' is taken here in its more general meaning, covering also strictly related notions like fact, action, state, situation etc. These actors try to attain a specific result, experience particular situations, manipulate some (concrete or abstract) materials, send or receive messages, buy, sell, deliver etc. Because of the ubiquity of these 'narrative' resources, being able to represent in a general, accurate, and effective way their semantic content – i.e., their key 'meaning' – is then both conceptually relevant and economically important, as demonstrated indirectly by the choice of *narrative examples* to illustrate the SWBPD WG document mentioned before.

As we have seen in Section 2.1 above, templates and predicative occurrences in NKRL are *n*-ary structures, and then suitable to take rationally into account narrative information. Moreover, NKRL is also endowed with a limited set of second order tools – mainly, binding occurrences and completive constructions, see again [2, 3, 4] – for dealing with those 'connectivity phenomena' that arise when several elementary events are connected through causality, goal, condition, indirect speech etc.

Given that the SW languages are 'binary' and NKRL is '*n*-ary', also the tools (rules) used to manage the data structures must be deeply different in the two approaches. Note that rules have not been included in the standard descriptions of SW languages like RDF and OWL, and the whole SW rule domain seems to be still in a very early state of development. Languages like RuleML [21], TRIPLE [22], and SWRL [23, 24] – all based, roughly, on extensions of the inferential properties of Horn clauses and Datalog to deal with OWL-like data structures, see also [25] in this context – appear to be, for the time being, more limited experimental proposals than implemented languages. With respect now to the existing, OWL-compatible reasoning tools like RACER [26], we must acknowledge that they are characterized by the use of sound and complete inferencing algorithms supported by the description logics (DL) theory [27]. Unfortunately, the reduced expressiveness of the DL main inferencing component, the automatic classification mechanism, linked with the reduced expressiveness of the underpinning data structures, confine their use to the execution of low-level reasoning operations like concept consistency, concept subsumption, instance testing etc., far away from the power of the unification-based, reasoning procedures in the NKRL style. Nearly a printed page is needed to McGuinness and her colleagues, see [28], to prove that, using the DAML+OIL definitions (DAML+OIL is the ancestor of OWL), it is possible to infer that ‘Red’ can be considered as a sort of ‘WineColor’. Under these conditions, it is not really surprising that, when confronted with the need of developing real-time extensive applications in a Semantic Web context, see [29], the developers feel compelled to use Jess as their rule-processing vehicle, i.e., the re-implementation in Java terms of tools like CLIPS that go back to OPS5, the seventies and the expert systems era.

For an approach to ‘inferencing’ similar to that described in this paper, it is then better to look into the ‘semantic network’ domain – in the widest meaning of these words, see [30] – given that systems dealing with *n*-ary data structures are relatively frequent in this field.

Conceptual Graphs (CGs), see [31], are based on a powerful graph-based representation scheme that can be used to represent *n*-ary relationships between complex objects in a system. A conceptual graph is a finite, connected, bipartite graph that makes use of two kinds of nodes, i.e., ‘concepts’ and ‘conceptual relations’; every arc of a graph must link a conceptual relation to a concept. ‘Bipartite’ means that every arc of a conceptual graph associates necessarily one concept with one conceptual relation: it is not possible to have arcs that link concepts with concepts or relations with relations. Very interesting work concerning indexing and querying of knowledge bases of CGs making use of unification techniques can be found, e.g., in [32, 33]; Corbett [34] adds to the standard CGs unification procedures the possibility of dealing with constraints, leading then to the implementation of unification algorithms very similar to those used in Fum. We can make two remarks about CGs.

The first concerns the feeling that, in spite of the advanced unification-oriented work mentioned in the previous paragraph, the existing CGs tools are still relatively limited with respect to their inferencing capabilities. Some of them, like CharGer [35], CGWorld [36] and DNAT [37], are essentially user interfaces and editing platforms able to ‘produce’ CGs and CGs-based annotations – DNAT annotations and graphs can be afterwards processed using standard tools like RDF and OWL. Others, like the CoGITaNT [38] and Notio API libraries [39], allow the execution of some standard canonical operations for creating and modifying graphs. CoGITaNT includes also the possibility of implementing ‘if-then’ rules where antecedent and consequent are in the form of conceptual graphs, and to make use of CGs-implemented constraints. The third version of the Amine Platform [40] allows, among other things, the execution of some advanced information retrieval operations on a knowledge base in the form of CGs. With respect now to Corese [41], this corresponds to an RDF engine implemented in Conceptual Graphs terms that enables the processing of RDF and RDFS statements within the CG formalism: in this case then, the original *n*-ary properties of CGs are downgraded to the usual binary, W3C-like ones.

The second remark is of a more general nature, and concerns what seems to be an important difference between the NKRL- and CGs-based approaches to the set up and managing of *n*-ary structures. Even if CGs have obviously the possibility of defining general conceptual structures, similar to the NKRL templates, for describing narrative-like phenomena, an exhaustive and authoritative list of these structures under the form of ‘canonical graphs’ does not exist, and its construction seems have never been planned. The practical consequence of this state of affairs seems to be the need, whenever a concrete application of CGs must be implemented, of defining anew a specific list of canonical graphs for this particular application. On the contrary, a fundamental characteristic of NKRL concerns the fact that its *catalogue* of 'basic templates' (coinciding with HTemp and including 150 templates, very easy to extend and customize) is in practice *part and parcel* of the definition of the language – as already stated at length in this paper, all the different sorts of low-level or high-level inferential rules used in NKRL are then obtained via the partial instantiation of the templates of the catalogue. This approach could be very important for practical applications, and it implies, in particular, that: i) a system-builder does not have to create himself the structural and inferential knowledge needed to describe and exploit the events proper to a large class of 'narratives' (in the most general meaning of this word); ii) it becomes easier to secure the reproduction or the sharing of previous results.

We will conclude this Section with the mention of CYC, see [42]: CYC concerns one of the most controversial endeavours in the history of Artificial Intelligence. Started in the early '80 as a MCC (Microelectronics and Computer Technology Corporation, Texas, USA) project, it ended about 15 years later with the set up of an enormous knowledge base containing about a million of hand-entered 'logical assertions' including both simple statements of facts and rules about what conclusions can be inferred if certain statements of facts are satisfied. The 'upper level' of the ontology that structures the CYC knowledge base is now freely accessible on the Web, see http://www.cyc.com/cyc/opencyc. A detailed analysis of the origins, developments and motivations of CYC can be found in [43: 275-316]. The knowledge representation language of CYC, CycL, is an *n*-ary language see, e.g., [44].

It is evident that CYC is an amazing achievement from which the entire field of knowledge representation has learned a great deal. However, it has also been severely criticized: e.g., one of the main 'technical' criticisms addressed to CycL concerns its uniform use of the same representation framework (substantially, a frame system rewritten in logical form) to represent entities that are conceptually very different (the 'uniqueness syndrome'). In NKRL, on the contrary, concepts are represented in the (usual) binary way, elementary events (and general classes of events) like *n*-ary structures, connectivity phenomena as labelled lists with reified arguments, etc. An important aspect to take into consideration about CYC concerns the fact that this system defines itself as a 'business' product, commercialised by a 'real' company called Cycorp: however, the real value of at least some of its 'commercial' results is still fundamentally unclear. See, in this context, the comments of the evaluators about the results of the recent HALO project, http://www.projecthalo.com/ halotempl. asp?cid=2133, where it appears that CYC has 'beaten' organisms like SRI or Ontoprise, but also that its real 'understanding' of the problems dealt with appear to be extremely low. HALO concerns "… the development of a 'Digital Aristotle' – a staged, long-term research and development initiative that aims to develop an application capable of answering novel questions and solving advanced problems in a broad range of scientific disciplines".

7 Conclusion

In this paper, we have first mentioned the advantages that, for an in-depth exploitation of 'terrorism' material, could derive from an integration of the two main inferencing modes in NKRL, 'hypotheses' and 'transformations'. We recall here that the 'hypothesis rules' allow to retrieve automatically from an NKRL knowledge base the information that can supply a context or a causal explanation for some known event, and that the 'transformation rules' facilitate in general the recovery of information from the knowledge base by 'adapting', from a semantic point of view, query/queries that failed to the real contents of this knowledge base. Allowing the use of transformations to modify the reasoning steps of hypotheses lets to 'break' the predefined scenarios proper to the hypothesis rules and to augment then the possibility of discovering 'implicit' and 'unexpected' information within the knowledge base. We have then introduced the conceptual problems linked with this integration and the solutions adopted, and then we have shown briefly how these solutions could be implemented from a software point of view. To emphasise the originality of the NKRL approach to the implementation and use of high-level inferencing procedures, the paper ends up with some comparisons with work done in a Semantic Web, Conceptual Graph and CYC context.

References

1. Popp, R., Armour, T., Senator, T., and Numrych, K.: Countering Terrorism Through Information Technology. Communications of the ACM (2004) 47(3): 36-43
2. Zarri, G.P.: NKRL, a Knowledge Representation Tool for Encoding the 'Meaning' of Complex Narrative Texts. Natural Language Engineering – Special Issue on Knowledge Representation for Natural Language Processing in Implemented System*s* (1997) 3: 231-253
3. Zarri, G.P.: Representation of Temporal Knowledge in Events: The Formalism, and Its Potential for Legal Narratives. Information & Communications Technology Law – Special Issue on Models of Time, Action, and Situations (1998) **7**: 213-241
4. Zarri, G.P.: A Conceptual Model for Representing Narratives. In: Innovations in Knowledge Engineering. Advanced Knowledge International, Adelaide (2003)
5. Rinaldi, F., Dowdall, J., Hess, M., Ellman, J., Zarri, G.P., Persidis, A., Bernard, L., and Karanikas, H.: Multilayer Annotations in PARMENIDES. In: Proceedings of the K-CAP (International Conference on Knowledge Capture) 2003 Workshop on Knowledge Markup and Semantic Annotation (October 25-26, 2003, Sanibel Island, Florida, USA)
6. Noy, F.N., Fergerson, R.W., and Musen, M.A.: The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In: Knowledge Acquisition, Modeling, and Management – Proceedings of the European Knowledge Acquisition Conference, EKAW'2000. Springer-Verlag, Berlin Heidelberg New York (2000)
7. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., and Stein, L.A. (eds.): OWL Web Ontology Language Reference – W3C Recommendation 10 February 2004. W3C (2004) (http://www.w3.org/TR/owl-ref/)
8. Horridge, M.: A Practical Guide to Building OWL Ontologies with the Protégé-OWL Plugin (Edition 1.0). The University of Manchester, Manchester (2004)
9. Zarri, G.P.: NKRL Manual, Part II – The HClass and HTemp Hierarchie*s* (Parmenides IST Report). University of Paris IV/Sorbonne, Paris (2003)
10. Zarri, G.P., and Bernard, L.: NKRL Manual, Part III – The NKRL Software (Parmenides IST Report). University of Paris IV/Sorbonne, Paris (2004)
11. Zarri, G.P.: Automatic Representation of the Semantic Relationships Corresponding to a French Surface Expression. In: Proceedings of the First International Conference on Applied Natural Language Processing. Association for Computational Linguistics (ACL), East Stroudsburg (PA) (1983)

12. Zarri, G.P. : Semantic Modeling of the Content of (Normative) Natural Language Documents. In: Actes des douzièmes journées internationales d'Avignon 'Les systèmes experts et leurs applications' - Conférence spécialisée sur le traitement du langage naturel. EC2, Nanterre (1992)
13. Black, W.J., Jowett, S., Mavroudakis, T., McNaught, J., Theodoulidis, B., Vasilakopoulos, A., Zarri, G.P., and Zervanou, K.: Ontology-Enablement of a System for Semantic Annotation of Digital Documents. In: Proceedings of the 4th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2004) – 3rd International Semantic Web Conference (November 8, 2004, Hiroshima, Japan).
14. Fikes, R., Hayes, P., and Horrocks, I.: OWL-QL – A Language for Deductive Query Answering on the Semantic Web. Web Semantics: Science, Services and Agents on the World Wide Web (2004) 2: 19-29.
15. Bertino, E., Ferrari, E., Perego, A., and Zarri, G.P.: An Integrated Approach to Rating and Filtering Web Content". In: Innovations in Applied Artificial Intelligence: Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2005. Springer-Verlag, Berlin Heidelberg New York (2005)
16. Clocksin, W.F., and Mellish, C.S.: Programming in PROLOG. Springer-Verlag, Berlin Heidelberg New York (1981)
17. Zarri, G.P., and Bernard, L.: Using NKRL Inference Techniques To Deal With MoD 'Terrorism' Information (Parmenides IST Report). University of Paris IV/Sorbonne, Paris (2004)
18. Koster, C.H.A.: Head/Modifier Frames for Information Retrieval. In: Computational Linguistics and Intelligent Text Processing: Proceedings of the 5th International Conference, CICLing 2004. Springer-Verlag, Berlin Heidelberg New York (2004)
19. Clark, K.L., and Tärnlund, S.-A. (eds.): Logic Programming. Academic Press, London (1982)
20. Noy, N., and Rector, A. (eds.): Defining N-ary Relations on the Semantic Web–W3C Working Draft 24 May 2005. W3C (2005) (http://smi-web.stanford.edu/people/noy/nAryRelations/n-aryRelations-2nd-WD.html).
21. Boley, H., Tabet, S., and Wagner, G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In: Proceedings of SWWS'01, The First Semantic Web Working Symposium. Stanford University, Stanford (2001)
22. Sintek, M., and Decker, S.: TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web. In: Proceedings of the First International Semantic Web Conference – ISWC 2002. Springer-Verlag, Berlin Heidelberg New York (2002)
23. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., and Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML – W3C Member Submission 21 May 2004. W3C (2004) (http://www.w3.org/Submission/SWRL/)
24. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., and Tsarkov, D.: OWL Rules: A Proposal and Prototype Implementation. Web Semantics: Science, Services and Agents on the World Wide Web (2005) 3: 23-40
25. Rosati, R.: On the Decidability and Complexity of Integrating Ontologies and Rules: Web Semantics: Science, Services and Agents on the World Wide Web (2005) 3: 61-73
26. Haarslev, V., and Möller, R.: Racer: A Core Inference Engine for the Semantic Web. In: Proceedings of the 2nd International Workshop on Evaluation of Ontology Tools (EON2003), Sanibel Island (October 20, 2003, Florida, USA)
27. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P.F. (eds.): The Description Logic Handbook. University Press, Cambridge (2003)

28. McGuinness, D.L., Fikes, R., Hendler, J., and Stein, L.A.: DAML+OIL: An Ontology Language for the Semantic Web. IEEE Intelligent Systems (2002) 17(5): 72-80
29. Hatala, M., Wakkary, R., and Kalantari, L.: Rules and Ontologies in Support of Real-Time Ubiquitous Application. Web Semantics: Science, Services and Agents on the World Wide Web (2005) 3: 5-22
30. Lehmann, F. (ed.): Semantic Networks in Artificial Intelligence. Pergamon Press, Oxford (1992)
31. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundation*s*. Brooks Cole Publishing Co., Pacific Grove (CA) (1999)
32. Ellis, G.: Compiling Conceptual Graph. IEEE Transactions on Knowledge and Data Engineering (1995) 7: 68-81
33. Willems, M.: Projection and Unification for Conceptual Graphs. In: Proceedings of the Third International Conference on Conceptual Structure*s*. Springer-Verlag, Berlin Heidelberg New York (1995)
34. Corbett, D.: Reasoning and Unification over Conceptual Graphs, ICCS'95. Kluwer Academic/Plenum Publishers, New York (2003)
35. Delugach, H.S.: CharGer: A Graphical Conceptual Graph Editor. In: Proceedings of the ICCS 2001 Workshop for Conceptual Graphs Tools (July 30, 2001, Stanford University, USA, http://www.cs.nmsu.edu/~hdp/CGTools/proceedings/papers/CharGer.pdf)
36. Dobrev, P., Strupchaska, A., and Toutanova, K.: CGWorld-2001 – New Features and New Directions. In: Proceedings of the ICCS 2001 Workshop for Conceptual Graphs Tools (July 30, 2001, Stanford University, USA, http://www.cs.nmsu.edu/~hdp/CGTools/proceedings/papers/CGWorld.pdf)
37. Uhlir, J., Kremen, P., and Kral, L.: DNAT – User's Manual (Cipher IST Deliverable D26/2). Czech Technical University, Prague (2004)
38. Genest, D., and Salvat: A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT. In: Proceedings of the Sixth International Conference on Conceptual Structures, ICCS'98. Springer-Verlag, Berlin Heidelberg New York (1998)
39. Southey, F., and Linders, J. G.: Notio – A Java API for Conceptual Graphs. In: Proceedings of the Seventh International Conference on Conceptual Structures, ICCS'99. Springer-Verlag, Berlin Heidelberg New York (1999)
40. Kabbaj, A., Moulin, B., Gancet, J., Nadeau, D., and Rouleau, O.: Uses, Improvements and Extensions of Prolog+CG: Case Studies. In: Proceedings of the Ninth International Conference on Conceptual Structures, ICCS'01. Springer-Verlag, Berlin Heidelberg New York (2001)
41. Corby, O., Dieng-Kuntz, R., and Faron-Zucker, C.: Querying the Semantic Web with the CORESE Search Engine. In: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'2004*)*. IOS Press, Amsterdam (2004)
42. Lenat, D.B., Guha, R.V., Pittman, K., Pratt, D., and Shepherd, M.: CYC: Toward Programs With Common Sense. Communications of the ACM (1990) 33(8): 30-49
43. Bertino, E., Catania, B., and Zarri, G.P.: Intelligent Database System*s*. Addison-Wesley and ACM Press, London (2001)
44. Ramachandran, D., Reagan, P., and Goolsbey, K.: First-Orderized ResearchCyc: Expressivity and Efficiency in a Common-Sense Ontology. In: Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Application*s* (July 2005, Pittsburgh, USA, http://www.cyc.com/doc/white_papers/folification.pdf).

Author Index