Lecture Notes in Computer Science 2405

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Barry Eaglestone Siobhán North
Alexandra Poulovassilis (Eds.)

Advances in Databases

19th British National Conference on Databases, BNCOD 19
Sheffield, UK, July 17-19, 2002
Proceedings

Springer

Volume Editors

Barry Eaglestone
University of Sheffield, Department of Information Studies
Sheffield S10 2TN, United Kingdom
E-mail: b.eaglestone@sheffield.ac.uk

Siobhán North
University of Sheffield, Department of Computer Science
Sheffield S10 2TN, United Kingdom
E-mail: s.north@dcs.shef.ac.uk

Alexandra Poulovassilis
Birkbeck College, Computer Science and Information Systems
Malet Street, London WC1E 7HX, United Kingdom
E-mail: a.poulovassilis@dcs.bbk.ac.uk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in databases : proceedings / 19th British National Conference on Databases, BNCOD 19, Sheffield, UK, July 17 - 19, 2002. Barry Eaglestone ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2405)
ISBN 3-540-43905-6

CR Subject Classification (1998): H.2-4

ISSN 0302-9743
ISBN 3-540-43905-6 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Christian Grosche, Hamburg
Printed on acid-free paper SPIN 10873659 06/3142 5 4 3 2 1 0

Foreword

The British National Conference on Databases (BNCOD) was established in 1980 as a forum for research into the theory and practice of databases. This aim remains highly relevant today as database technology must meet the challenges of global applications that need to access and analyse data stored in multiple heterogeneous, autonomous information sources on the Web.

This volume contains the proceedings of the 19^{th} BNCOD, held at the University of Sheffield in July 2002. The conference theme this year is "Exploring the roles of database technology within the global computing infrastructure". There are 10 full-length papers, 9 poster papers, and two invited talks. As in previous BNCOD conferences, there are a significant number of papers from abroad, including authors from Brazil, France, Italy, The Netherlands, Spain, Sweden, and the USA.

Our first invited speaker is Jiawei Han who is currently at the University of Illinois at Urbana-Champaign. Jiawei Han is a distinguished researcher in the area of data mining, and his invited talk addresses in particular data mining techniques for stream data. The motivation for this research is the vast amounts of data produced at high speed by many real-time systems, which for practical reasons cannot be stored on disk for off-line analysis. Thus, techniques are needed for real-time mining of such stream data. Jiawei Han's talk addresses recent developments in this area, and the major research challenges arising.

Our second invited speaker is Richard Hull who is Director of the Network Data and Services Research Department at Bell Laboratories. Richard Hull's early research was in the fundamentals of database systems, with many seminal papers and results. More recently he has been working on information integration, workflow, and e-services, and his invited talk addresses personalisation of network-hosted services. His talk motivates the need for personalised services provided by "intelligent collective networks", and then focuses in particular on the research challenges arising in data management and policy management for such networks.

The refereed full-length papers are presented in four sessions. The first session is on Query Optimisation. Here, the paper by Gounaris et al. is motivated by the need for adaptive query processing in wide-area distributed applications, where it may be difficult to predict query performance statically and where techniques are needed that adapt query execution plans during query execution. The paper gives a survey of adaptive query processing, comparing and contrasting several techniques with respect to a number of factors, and identifying specific research challenges. The paper by Gianella et al. discusses the use of approximate functional dependencies in query optimisation. They propose decomposing relations according to the functional dependencies which hold in parts of the relation, and describe two query rewriting techniques utilising the decomposed relations, one which always preserves correctness and one which preserves correctness for a certain class of queries. Experimental results are presented and discussed for both techniques.

The second session of full papers is on Data Warehousing and Mining. Voloso et al. focus on the problem of how to choose between the various proposed association rule mining algorithms for a specific data set. They present a comparison and evaluation of

several algorithms with respect to some real and synthetic data sets, and propose a new adaptive algorithm designed for better general performance on real data sets. Engström et al. describe a test bed for measuring the performance of different data warehouse maintenance policies with respect to data sources of different capabilities. They present and discuss some experimental results for relational and XML data sources, concluding that maintenance policy performance can be significantly influenced by source capability.

The third full paper session is on Quality and Integrity. Burgess et al. present a domain-independent taxonomy of quality, with the aim of reducing the problem of information overload when querying distributed heterogeneous environments by limiting the information retrieved to the subset satisfying specific quality requirements. Couchot addresses the problem of analysing the termination behaviour of triggers, or active rules, which are useful for detection and flexible enforcement of integrity constraints. The termination problem for active rules is generally undecidable and research has focussed on developing conservative tests. Couchot presents an improvement of a previous method which is able to detect more cases of rule termination than previous approaches. Finally, the paper by Fu et al. addresses the problem of analysing integrity constraints extracted from legacy systems. A polynomial-time algorithm is presented for determining related integrity constraints, where related constraints are ones that contain a specified syntactic sub-structure.

The fourth full paper session is on Web and Distributed Databases. Zwol and Apers consider techniques for better information retrieval from the Web if the search focus is on a smaller, domain-specific document collection. They present a method which integrates the semantic content of such a document collection into a schema which can then be used to formulate queries over the collection. Some experimental results are presented and analysed which show improved performance over standard search engines. Theodoratos describes a hypergraph-based query language for integration of heterogeneous data sources via view definitions. Two view transformations can be used to incrementally construct or evolve views. Brisaboa et al. present an architecture for generating natural language interfaces to document databases. An ontology is used to describe the concepts of the document database, and a User Interface Generator automatically generates a user interface for that database from the ontology and a set of "skeleton sentences".

Finally, there is a session consisting of 9 poster papers, which were also refereed by the Programme Committee. The first three poster papers, by Boyd et al., Jasper, and Fan, describe recent work on the AutoMed project at Birkbeck and Imperial Colleges, which is investigating heterogeneous database integration via schema transformation rules. The next three poster papers, by Gupta et al., Ram et al., and Shou and North focus on techniques for semantic integration of distributed heterogeneous information sources. The last three poster papers, by Medina et al., Lepinioti and McKearney, and Lodi concern data warehousing and data mining.

Acknowledgements

We would like to thank all the members of the Programme Committee for their excellent work in reviewing the submitted papers and posters. Their commitment and enthusiasm have resulted in an exciting programme for this 19th BNCOD, in keeping with the long and distinguished tradition of previous conferences. Many thanks go also to Alex Gray, Chair of the BNCOD Steering Committee, for his invitation to organise BNCOD 2002 and for his continued support and advice. Last but not least, we would like to thank Monika Kus for her invaluable secretarial support.

May 2002 Barry Eaglestone, Siobhán North, Alex Poulovassilis

Conference Committees

Programme Committee

Alexandra Poulovassilis (Chair)	Birkbeck College, University of London
David Bell	University of Ulster
Peter Buneman	University of Pennsylvania
Richard Connor	University of Strathclyde
Barry Eaglestone	University of Sheffield
Suzanne Embury	University of Manchester
Enrico Franconi	University of Manchester
Carole Goble	University of Manchester
Alex Gray	University of Wales, Cardiff
Peter Gray	University of Aberdeen
Mike Jackson	University of Wolverhampton
Anne James	Coventry University
Keith Jeffery	CLRC Rutherford Appleton
Graham Kemp	University of Aberdeen
Jessie Kennedy	Napier University
Brian Lings	University of Exeter
Nigel Martin	Birkbeck College, University of London
Peter McBrien	Imperial College
Ken Moody	University of Cambridge
Siobhn North	University of Sheffield
Werner Nutt	Heriot-Watt University
Norman Paton	University of Manchester
Brian Read	London Guildhall University
Howard Williams	Heriot-Watt University
Peter Wood	Birkbeck College, University of London

Organising Committee

Barry Eaglestone (Chair)	University of Sheffield
Alex Gray	University of Wales, Cardiff
Anne James	Coventry University
Siobhn North	Sheffield University
Miguel Nunes	Sheffield University

Steering Committee

Alex Gray (Chair)	University of Wales, Cardiff
Nick Fiddian	University of Wales, Cardiff
Carole Goble	University of Manchester
Peter Gray	University of Aberdeen
Keith Jeffery	CLRC Rutherford Appleton
Roger Johnson	Birkbeck College, University of London
Jessie Kennedy	Napier University
Brian Lings	University of Exeter

Table of Contents

Invited Paper

Query Processing

Refereed Poster Papers

Data Warehousing and Mining

Quality and Integrity

Web and Distributed Databases

Have It Your Way: Personalization of Network-Hosted Services

Richard Hull, Bharat Kumar, Arnaud Sahuguet, and Ming Xiong

Network Data and Services Research
Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974
{hull,bharat,sahuguet,mxiong}@lucent.com

Abstract This paper surveys recent trends in network-hosted end-user services, with an emphasis on the increasing complexity of those services due to the convergence of the conventional telephony, wireless, and data networks. In order to take full advantage of these services, private and corporate users will need personalized ways of accessing them. Support for this personalization will involve advances in both data and policy management.

1 Introduction: The Challenge of too Many Services

With the emergence of the World Wide Web the realm of end-user accessible, network-resident data services has become extremely rich and varied. Voice services have lagged behind the web in terms of their complexity, due to their heritage in monolithic circuit-based networks. But voice services are catching up, as packet-based voice becomes more common and gateways are being made between the telephony network and the internet. For example, voice access to web-resident information (e.g., weather, email, personal calendar) is now becoming available through several service providers. And through standards such as Parlay [Par], internet-based services will have broad access to a variety of telephony network information and services (e.g., presence and location awareness, billing, call control), enabling third-party application developers to create a multitude of services that freely mix web-style and voice-based functionalities. Add to this the variety of mobile networks (2G, 3G, WAP, 802.11, ...), the increasing use of instant messaging, and the emergence of SIP [SIP] and intelligent terminal devices, and we see that most end-users will soon be deluged with a broad variety of services and ways to combine them, along with a multitude of details about all the different ways to contact their friends and associates.

Due to the wide-spread use of mobile devices the "collective network", or more precisely, the family of circuit-based, wireless, and data networks, is truly becoming a *ubiquitous* feature in our daily lives. Through a combination of mobile and fixed devices, a substantial portion of the world's population is reachable in an instant, and they can access the network whenever they want. Further, when a mobile device is on, both its status (idle, on-call, etc.) and approximate location can be tracked. And many of us spend one or more hours working with a network-connected computer. As a result, the collective network has a tremendous wealth of up-to-date information about where people are and whether they are available for communication.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 1–10, 2002.

So what will communication-oriented services look like in the future, and why do they need to be personalized? We mention here four key trends.

Making It Easier to Request Services: An important trend is to make the collective network to be more "intelligent" with regards to providing typical services. A simple example of this is to support so-called "call by name" as opposed to "call by number". With call by name a user identifies who they want to call by that person's name, and lets the network determine which number or numbers to actually call, in order to make the connection. More generally, an "intelligent" network could keep track of end-user address books, preferences, appointment books, and habits, and permit the user to request services through high-level directives (e.g., "call Jerome" or "arrange a flight to Chicago after my last appointment on this Wednesday"), rather than the explicit style of service requests used today. The network should also be aware of the device or devices that the end-user is currently using or has available, and tailor information delivery to those devices appropriately. It should be noted that as technology surrounding the web services paradigm matures (e.g., to incorporate automated reasoning for service discovery and composition), then both end-user requests and automated notifications will be able to trigger increasingly sophisticated combinations of services, which implies the increasing complexity of taking individual profile data and preferences into account.

Automated Notifications: We are already seeing automated notification services in limited areas (e.g., flight information, financial market events, and in e-commerce notification if your order gets shipped). These will proliferate as it becomes easier for service providers and businesses to provide them, and for end-users to launch and customize them. In the future we should be able to register for a broad variety of event notifications, specifying also under what conditions a notification should be issued, and how and when the notifications should be delivered (e.g., urgent ones to a pager or cell phone, less urgent ones to voice mail or email). (As an example of notifications gone awry, a well-known bank offered the service of automatically paging clients when their personal monthly 401K statements had been emailed. In general, those emails were processed in bulk during slow periods in the bank's schedule, i.e., at around 2 in the morning!)

Support for Collaborative Activities: Appropriate use of the collective network can substantially enhance the effectiveness of long-term (e.g., multi-day or multi-month) collaborations. This can range from enhancing simple conference calls into flexible multimedia/multi-device conferencing, to supporting persistent "chat sessions" that provide multiple modes of interaction through time, including the abilities to share data files easily, to register for event notifications relevant to the collaboration, to track whether participants are available, and to contact participants immediately when this is deemed appropriate. This notion of collaboration extends also to relationships between enterprises and customers, e.g., between a bank and someone applying for a mortgage loan, where a variety of information must be passed, sometimes under time constraints. More generally, network-hosted services can become as rich as the collaborations that people and/or enterprises participate in.

Keeping Control in the Hands of the End-User: As noted earlier, the collective network has access to a vast amount of information about individual users. This information should be revealed to others only by permission. Further, the individual user will typically be willing to share certain information with certain other individuals, based on a variety of circumstances (e.g., business colleagues might be given access during business hours while family might be given access at essentially any time). It is important that users be able to express their preferences accurately, and that the collective network honor their right to privacy.

Achieving this vision of a more intelligent collective network involves advances in a number of diverse technologies, including data management; preference and policy management; workflow and collaborative systems; distributed systems; standards for sharing information and control across networks and devices; new forms of call models and session management; and continued work on natural language interfaces. The current paper is focused on challenges arising in data and policy management.

2 Managing User Profile Information

Network-hosted services revolve around providing information to end-users, providing connections between end-users with each other or with automated systems working on behalf of enterprises, and enabling end-users to invoke commercial, financial or other services outside of the network proper. Simply providing these services entails the storage and use by the collective network of a broad array of information about end-users, including, e.g., the correspondence between users and their devices, awareness that mobile devices are on data and/or circuit paths for establishing connections to them, and awareness of the billing model (e.g., pre-paid vs. post-paid, and the billing plan). A key element of simplifying and personalizing these services is for the collective network to store and use information specific to individual end-users and relevant to the services being provided, and to flexibly share this information across the components within networks, and between different networks.

Currently, end-user information is scattered across the networks, and typically organized by device rather than end-user, with primary emphasis given to the performance of individual services rather than enabling sharing of information between services and networks. More specifically, in the circuit telephony realm information about subscribers and how to bill them is maintained in so-called “central offices”, which are generally opaque to internet-based applications. In the wireless domain a network of “Home Locator Resources” (HLRs) hold real-time data about device location and status, and have ready access to pre-paid billing information (so that service can be terminated if the account becomes empty). Only recently is HLR data becoming more freely available, through standards such as Parlay. In the internet user data is managed in an extremely loose and distributed manner. The end-user relevant data managed by the various networks is largely governed by standards bodies (e.g., ITU-T [ITU] and more recently Parlay for the wireline telephony network, 3GPP [3GPa] and 3GPP2 [3GPb] for the wireless networks, and IETF [IET] for the internet).

What data needs to be maintained and shared for simplifying and personalizing network-hosted services? Web portals such as MyYahoo! provide a simple form of per-

sonalization, allowing users to specify what locales they would like weather information for, or what kinds of news stories should be presented. Services involving realtime communication can involve more intricate personalization. For example, for "call by name" the network needs to hold a mapping from end-user abbreviations (e.g., "my boss") to unique person identifiers, and from these to the family of devices used by the identified person. Realtime information about presence, availability, and location should be factored into the "call by name" service, to help gain access to the callee. And this realtime information must be filtered against the callee's preferences, e.g., based on her schedule (e.g., working hours vs. personal hours), the caller, and the current situation (e.g., working against an urgent deadline, on vacation, in transit, open to interruption). Over time, the preferences that can be stored will become more and more intricate.

Convenient sharing of profile and preference information between networks, portals, and services is not currently supported. An important pre-cursor to this sharing is to provide support for secure and controlled sharing of such information across networks and between enterprises. The Liberty Alliance consortium [Lib] is working to develop standards in this area. A key paradigm is to support the notion of "network identity", enabling a single sign-on to the network, which can be used transitively to authenticate user-specific access to services across the collective network. The development of this paradigm must take into account issues around enterprise firewalls, since a large population is based within enterprises and use the network for conducting enterprise-directed activities. Importantly, a single sign-on mechanism can provide the vehicle for passing user-specific profile and preference information to services, although care must be taken to share with a given service only information that the user has permitted for that service.

What mechanisms and paradigms should be used to share user profile and preference information across the various networks, and across a multitude of services? The basic answer is to generalize the federated database architecture [HM85,SL90] that was developed in the 70's and 80's. That architecture, developed primarily for data sharing within an enterprise context, acknowledges that multiple relatively autonomous sub-organizations will maintain their own databases, and proposes essentially that a *virtual* global schema be created for sharing data between those organizations. The global schema can be viewed as providing a conceptual single-point-of-access for all of the data, without requiring the different sub-organizations to store their data in a specific format or technology.

Two generalizations of the federated model are needed to effectively support sharing of user profile information. The first arises in the context of a single service provider, where data is often replicated in network components to achieve satisfactory performance. Unlike the usual federated paradigm, these network components are embedded devices that do not provide data management support typical of modern DBMSs. Further, updates may happen in both directions – a change to call-forwarding information might be received from a cell phone through an embedded copy of data, while a change to a billing plan will typically come from a database-resident copy of data and be passed to relevant embedded components. As user profile information becomes more sophisticated, the inherent distribution and complexity of the storage architecture will make it increasingly difficult and expensive for applications and services to access it. For this

reason, following the spirit of the federated architecture, an abstraction layer should be introduced, to support a conceptual single-point-of-access for (realtime and static) user profile information. Services and applications can then access user profile information in a uniform and consistent manner. Because performance is critical in network-hosted applications, new techniques will be needed to provide adequate performance underneath this abstraction layer.

The second generalization of the federated model involves allowing for the graceful interaction between multiple federations. Each host must be able to access and use data that is coming from "outside", i.e., from other hosts and entities in the collective network. One approach to support such sharing is illustrated by the HLR. Sharing of data between HLRs is crucial for support of cell-phone "roaming", and standards have been developed to support this data sharing. But standards will be more elusive in other domains. A wireless service provider (WSP) may want to support the ability of an end-user to maintain an address book in the network, and to synchronize it with the address book on their cell phone. While some users will be satisfied with having the WSP hold the primary copy of their address book, other users will prefer that an independent third party, e.g., a web portal, hold the address book. In this case, the portal and the WSP may both hold federated schemas for their internal data, but also need to share data between those federated schemas. Unlike the original federated architecture, there is no over-arching organization in place to maintain a federated schema between these two entities. Richer examples arise when data relevant to providing a single service is spread across multiple hosting organizations, e.g., when the wireless network holds presence information, a web portal holds preference information, an enterprise is providing a particular service for a fee, and another enterprise (or a network operator) is providing billing services.

What about the data models that are and will be used? While most network-hosted data continues to reside in relational DBMSs, there is increasing usage of an LDAP representation, motivated by the fact that LDAP is rich enough and efficient for many current user profile applications, and provides flexible support for physical distribution. As user profile information becomes richer and as XML becomes more pervasive, we expect a gradual transition to the use of XML for passing profile information between network hosts. Indeed, adopting XML offers the advantages of a unified data model and unified query language, that easily encompass the relational and LDAP models and languages. Further, a standard (XACML) is being developed for access control against XML data.

Schema management will emerge as a key research area in profile management. Historically, schemas for profile databases have developed in an *ad hoc* way by different providers. More recently, standards groups (e.g., DMTF [DMT] DEN User Information Model, 3GPP Generic User Profile) have proposed or are developing standardized schemas for holding profile information. Looking forward we expect network operators to take a hybrid approach – starting essentially with a standardized schema for profile information, but adding to it as new services arise that require the incorporation of new profile information. In many cases, the new services will be developed by third-party application developers. How does the network operator ensure that the growth and evolution of the profile schema is reasonably coherent? More specifically, what tools can be

developed, so that developers that need to extend the profile schema hosted in a network can (a) easily learn what is already in the profile schema, (b) avoid adding redundant elements (unless there is good cause), and (c) add new elements in a manner consistent with the rest of the schema. If the data is represented in XML format, then a possible direction is to annotate the profile schema (perhaps represented in XML Schema) with semantic descriptions of different nodes and subtrees.

3 Managing Preferences and Policies

Maintaining profile and preference information in a (distributed, federated) profile database can provide a solid foundation for extensive personalization of network-hosted services. In this section we argue that incorporating rules-based preferences and policy management into the network can provide substantially increased flexibility and ease of maintenance for supporting this personalization. We also describe several research issues raised by this approach.

The notions of policy and policy-enabled have taken many meanings. In broad terms, *policy* in connection with a computerized system refers to the logic that governs choices that the system can make; these choices may be based on a variety of factors including personalization, optimization, quality of service, etc. A system is *policy-enabled* if it provides a mechanism to change the policy in a dynamic manner, without recoding or stopping the system. There are a range of approaches to policy-enablement in current systems. One approach, called here *data-structure-driven* policy enablement, is to provide a (presumably rich) data structure for holding preference information, and provide application code that interprets the data structure. The data in the data structure can be updated at will, thereby providing dynamically changeable behavior. For example, to support a "call by name" application in this manner there would be a data structure provided so that users can specify when and how they should be reachable, and when they should not be reachable. Another approach, called here *rules-based*, is to permit the specification of families of rules to define the policy, and to provide a rules engine for executing on those rules. Within this approach there is considerable variety, because there are several paradigms for rules-based systems, ranging from simple condition-action languages that do not permit chaining (e.g., as found in the IETF standards), to production system style systems (e.g., OPS5 [For81], CLIPS [CLI], ILOG [ILO]), to various styles of logic programming (e.g., Prolog, stratified, etc.) [Apt91] and other paradigms (e.g., PDL [LBN99], Ponder [DDLS01], Vortex [HLS+99]). The rules-based approach, especially if chaining is supported, is typically more flexible than the data-structure-driven approach, because in practical applications, a rules language has the potential to express more than a fixed program that is interpreting values of a fixed data structure. At the extremely expressive end of policy enablement would be the use of more advanced logical systems (e.g., description logics [BBMR89] or first-order logic) to specify policy. For the remainder of this paper we focus primarily on rules-based policy enablement.

With regards to personalization of network-hosted services, it is useful to keep four distinct architectural aspects of policy enablement in mind. One aspect concerns the presentation of the policy-enablement to end-users. In most cases, because end-users are

coming from all walks of life, they will be presented with a table-driven GUI for entering, viewing, and modifying their preference information. In the case of data-structure-driven policy enablement, the user-entered information will be used to populate the underlying data structure; indeed, there is typically a close relationship between that data structure and the choices presented to the user. In the case of rules-based policy enablement, the user-specified preferences will be translated into a collection of rules, which are held in a *policy database* (following here the terminology used by Parlay and other standards). A *policy execution point* is a location in the system where a collection of rules can be executed, in order to render a decision based on stored policy. And a *policy enforcement point* is a location in the system where the result of policy decisions can are used to dictate actual system behavior.

It is natural to ask at this point: if end-users are typically presented with a table-driven GUI for entering preferences, then why in practice is the increased flexibility of the rules-based approach really needed? One answer is that over time, if a rules-based approach is used then extensions to the kinds of preferences users can specify can typically be made by modifying the GUI and translation into rules, but without modifying the underlying infrastructure. Another answer is that a rules-based approach permits a common underlying infrastructure to support many different classes of users (e.g., road warriors, corporate executives, students, retirees, home-makers, emergency workers), with their respective GUIs that are based on substantially different kinds of preference information.

An important issue in policy management for personalization concerns the choice of rules (or richer logic) paradigm for specifying policies. The DMTF and IETF policy bodies, which focus on managing network and system components, have favored rules systems with no chaining: if the condition of a rule is true, then a specific action is taken at the appropriate policy enforcement point. In connection with personalization, however, the assignment of intermediate variables and the use of rule chaining can be beneficial. As a simple example, consider the "call by name" application (or any kind of calling application), and the preferences that the receiving party might specify to guard their privacy. It may be useful for the receiver to use rules to assign values to intermediate values such as "time category" (e.g., working hours, family hours, sleep hours) and "caller category" (e.g., colleague, family member, high priority caller). These variables might be used in rules that determine whether callers are routed to voice mail instead of making one or more phones ring. Set-valued intermediate variables can also be useful, so that the rules can consider a set of possibilities simultaneously (e.g., devices for contacting a person). The use of intermediate values can enable rule sets to be much more succinct than the case where intermediate values are not permitted. Finally, while intermediate variables and chaining seem useful, it is unclear how important recursive rules are for personalization applications. Restricting attention to acyclic rule sets has benefits in terms of performance, and simplifies other issues mentioned below.

We turn now to key research issues in policy management in support of personalizing network-hosted services. One key issue, of course, is performance. For many services in the telephony network users will expect response times of well under a second, which can be a challenge for typical rules engines operating in, e.g., typical production system applications. In the case of some network hosted services, at least in the near

term this may be mitigated because the size of the rule set used for a particular decision is limited (e.g., for "call by name", only the preferences of the caller and callee need be examined). However, a basic issue in policy management for personalization concerns the interaction of rules with databases. For example, in connection with "call by name" a user might have rules stating that a call from any employee of the same company will be received during working hours – this might mean that an incoming call will have to be checked against a corporate database holding information about 50K or more fellow employees. Mechanisms need to be developed for executing rule sets in a way that avoids expensive database dips whenever possible. >From an architectural point of view, it is important to understand a variety of tradeoffs in this area, including whether to place policy execution points "near" associated databases, and if that cannot be done, how much data to pass to the policy execution point when requesting a decision vs. having the execution point query a database for information on an as-needed basis.

Another research area concerns dynamic combinations of rules. For example, in policy languages such as Ponder and standards from IETF and Parlay, rules can have associated roles, and the roles can be used at runtime to gather together a set of rules to be used for a particular decision. In connection with "call by name" there might be separate rule sets for the caller and the callee, which are combined at runtime. And there might be additional rules that the caller's or callee's employee wish to be enforced, or that a network operator will insist upon. In cases where there is no rule chaining, it is relatively straightforward to interpret the meaning of an essentially arbitrary collection of rules. If there is rule chaining, and hence potentially complex interdependencies between rules, it may be difficult to successfully combine rules dynamically, since unexpected interactions between the combined rule set might arise. A promising direction here, at least in the restricted case of acyclic rule sets, might be to support the construction of targeted rule sets, that essentially map from selected "input" variables to selected "output" variables. We might permit combining of two rule sets if the "output" variables of one set correspond to the "input" variables of the other set. Generalizations of this are easily imagined, to permit multiple rule sets to be combined in a structured manner.

A final challenge concerns providing the ability to specify global policies in essentially one place, and to somehow manage to map them to appropriate network components to achieve maximum performance while remaining true to the semantics of the global policy. In more detail, when end-users specify preferences, they should be able to conceptualize the network as one unified system. But the rules that are created from their preferences will typically reside on, or provide decision-making for, multiple components in the network. As one example, in the Parlay standards several different components can be policy-enabled, including call control, presence awareness, location, and charging; a "call by name" capability might entail several or all of these. More generally, user preferences might be mapped to policy enforcement points at different levels of the network, e.g., to presence awareness at the level of the Parlay standards, to a softswitch for executing policy around call control, and to edge routers for enhancing the performance of streaming data associated with a multimedia call. The emergence of SIP and intelligent endpoints raises another form of distribution, e.g., the possibility

that my SIP-enabled personal phone and my SIP-enabled business phone could both execute policies on my behalf – but how can these policies be coordinated? Managing policy in these kinds of distributed environments is a relatively new topic for policy research. Recent work includes [PFW+02,AMN02], which develop approaches to provide forms of hierarchical structuring on rule sets, with a form of inheritance of rules from a super-entity to a sub-entity. Adapting these approaches to support personalization of network-hosted services will by challenging, in part because of the heterogeneity of the policy paradigms used by different network components, and the peer-to-peer nature of SIP endpoints.

4 Conclusions

We have seen that supporting personalization in network-hosted applications raises several research issues in both data and policy management. In data management this includes developing extensions of the federated model, new tools to support schema management and evolution, and of course, performance. In policy management this includes performance of rules engines in a new context, mechanisms for dynamic combining of rule sets, and mechanisms for working with distributed rule sets.

More generally, the area of personalization leads to a blurring of the traditional roles of databases and rules engines. In the context of personalization, a query such as "is Joe available for a call right now?" can not be viewed as a database access. Rather, it is a context- and time-specific request that will be answered by an interaction of rules engine and database, working with preferences stored in a combination of rules and data. As such, policies are used to control access to personal data according to user preferences. The converse also holds – that smart policies often require access to a variety of data. This confluence of data and policy management gives rise to a broad range of new questions in optimization, in models for distribution, and in providing structure for combining preference information from different sources.

Acknowledgments

The authors thank many of their colleagues at Bell Labs for numerous conversations that have formed the basis for much of the perspective presented here. Special acknowledgment is due to Peter Patel-Schneider, Prasan Roy, Jerome Simeon, and the late Javier Pinto. The Network Data and Services Research Department at Bell Labs is pursuing a number of research projects related to the themes described in this paper.

References

3GPa. 3GPP. The 3rd Generation Partnership Project. `http://www.3gpp.org`.

3GPb. 3GPP2. The 3rd Generation Partnership Project 2. `http://www.3gpp.org`.

AMN02. X. Ao, N. Minsky, and T.D. Nguyen. A hierarchical policy specification language, and enforcement mechanism, for governing digitual enterprises. In *Proc. of IEEE 3rd Intl. Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, Monterey, California, 2002.

Apt91. Krzysztof R. Apt. Logic programming. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 493–574. Elsevier, 1991.

BBMR89. A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 59–67, 1989.

CLI. CLIPS. CLIPS: A tool for building expert systems. `http://www.ghg.net/glips/CLIPS.html`.

DDLS01. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder specification language. In *Proc. of IEEE 2rd Intl. Workshop on Policies for Distributed Systems and Networks (Policy2001)*, HP Labs Brisol, UK, 2001.

DMT. DMTF. The Distributed Management Task Force. `http://www.dmtf.org`.

For81. C. L. Forgy. OPS5 user's manual. Technical Report CMU-CS-81-135, Carnegie Mellon University, 1981.

HLS+99. R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. of Intl. Joint Conf. on Work Activities Coordination and Collaboration (WACC)*, pages 69–78, February 1999.

HM85. D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Trans. on Office Information Systems*, 3(3):253–278, July 1985.

IET. IETF. The Internet Engineering Task Force. `http://www.ietf.org`.

ILO. ILOG. ILOG business rules. `http://www.ilog.com/products/rules`.

ITU. ITU-T. International Telecommunication Union – Telecommunications Standardization Sector. `http://www.itu.int.ITU-T/`.

LBN99. J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In *AAAI*, 1999.

Lib. Liberty Alliance. Liberty Alliance project. `http://www.projectliberty.org`.

Par. Parlay. The Parlay group. `http://www.parlay.org/`.

PFW+02. L. Pearlman, I. Foster, V. Welch, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proc. of IEEE 3rd Intl. Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, Monterey, California, 2002.

SIP. The SIP Forum. Session Initiation Protocol. `http://www.sipforum.org`.

SL90. Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

Adaptive Query Processing: A Survey

Anastasios Gounaris, Norman W. Paton,
Alvaro A.A. Fernandes, and Rizos Sakellariou

Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
{gounaris,norm,alvaro,rizos}@cs.man.ac.uk

Abstract In wide-area database systems, which may be running on unpredictable and volatile environments (such as computational grids), it is difficult to produce efficient database query plans based on information available solely at compile time. A solution to this problem is to exploit information that becomes available at query runtime and adapt the query plan to changing conditions during execution. This paper presents a survey on adaptive query processing techniques, examining the opportunities they offer to modify a plan dynamically and classifying them into categories according to the problem they focus on, their objectives, the nature of feedback they collect from the environment, the frequency at which they can adapt, their implementation environment and which component is responsible for taking the adaptation decisions.

1 Introduction

In a distributed environment, statistical information about the available data sources may be minimal, and the availability or load of physical resources is prone to changes. Consequently, traditional query optimisation can lead to poor performance, especially in long running query evaluations, as the query optimiser may not, at compile time, have the necessary statistics, good selectivity estimates, knowledge of the runtime bindings in the query, or knowledge of the available system resources required to produce an optimal query plan (QP). In addition, traditional optimisers cannot predict the future availability of resources. *Adaptive* (or dynamic) *query processing (AQP)* addresses this, by adapting the QP to changing environmental conditions at runtime.

Current research on adaptive query processing follows two main directions. The first approach responds to changes in the evaluation environment by modifying the execution plan at runtime (e.g. by changing the operators used or the order in which they are evaluated). The other approach involves the development of operators that deal more flexibly with unpredictable conditions and adapt their behaviour by collecting and taking into consideration information that becomes available at query runtime about how query evaluation is proceeding and about changes in the wider execution environment.

A query processing system is defined to be adaptive in [11] if it receives information from its environment and determines its behaviour according to that information in an iterative manner, i.e. there is a feedback loop between the environment and the behaviour of the query processing system. In this survey, we assume a narrower definition, in the sense that we restrict our attention to cases where the feedback loop produces effects during the execution of the query. If the effects of the feedback loop are deferred,

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 11–25, 2002.

then only subsequent queries, rather than the one that is running, can benefit from the adaptive technique employed. This is less appealing for novel data management tasks where systems are processing potentially very long running queries over multiple, semi-autonomous sources on wide-area networks with unpredictable data transfer rates. Broadly speaking, the need for adaptive query processing is emerging as query engines are scaled and federated, so that they must cope with highly unpredictable and volatile environments. If databases are to be integrated into the Grid [8], this need will become even greater.

The remainder of the paper is structured as follows. Section 2 describes the classification criteria and the potential attributes of AQP. Sections 3 and 4 present existing adaptive operators and algorithms respectively, followed by a discussion of general observations in Section 5. Section 6 concludes the paper.

Related Work: [11] presents the most relevant work to this survey. As mentioned above, it examines systems that may produce different query plans at any time in the future, in response to changes in the environment, while the present survey focuses on query processing techniques that can adapt to changing conditions during execution time. Also, [11] uses a very limited set of classification criteria. [14] extends this set by proposing some dimensions over which AQP systems may differ. Other previous surveys on query optimisation, like [10,12], include techniques that have a dynamic flavour, in the sense that the QP they produce at compile time is not completely fixed, but such techniques are not covered by the definition of adaptivity given earlier.

2 The Scope of the Survey

The query evaluator of a database management system (DBMS) comprises two main sub-components: the query optimiser (QO) and the execution engine. The query optimiser generates the input for the execution engine. After a query is submitted, the optimiser chooses one from many, logically equivalent, transformations of the query in the form of algebraic expressions. These logical query plans are often represented as logical operator trees. The most interesting characteristics of logical plans, with regard to query optimisation, are the location of project and select operators, the operator order and the tree shape.

A physical operator plan is derived from the logical one through the selection of algorithms to implement each of the operators of the logical plan. In order to produce the physical plan, decisions on whether to use indices or not, and whether to employ pipelining instead of blocking operators, are also taken. In general, each logical plan corresponds to many physical plans. Either cost models or heuristics are used to predict the cheapest plan for execution.

In parallel and distributed DBMSs, partitioning and scheduling issues arise during optimisation. A query plan can be divided into different sets of operators, called partitions, in order to harness, to the greatest possible extent, the benefits of parallelism. The query optimiser is responsible for deciding the number, the location, the kind, and how partitions will be scheduled.

Existing AQP techniques differ over many dimensions, as they have impacts of different kinds on the QP, they do not always share common goals and focus, and they

are applicable to different environments. Moreover, they may collect different feedback and adapt with different frequency. These dimensions are discussed in the following sections and provide the basis for the survey, which is summarised in Table 1.

2.1 Modifications in the Query Plan

Modifications to the query plan may occur at two levels: logical and physical. Modifications at the logical level (*LL* field in Table 1) may fall into one of the following categories:

Reformulation of the remainder of the query plan, denoted in Table 1 by *rem*, when the AQP techniques can produce a totally different logical subplan for the remainder of the query, introducing new logical operators, changing the operator order or altering the tree shape.

Operator Reordering, denoted in Table 1 by *op_or*, when techniques can only change the order of pre-defined logical operators. The two categories, *rem* and *op_or*, are not mutually disjoint, as the extent of the effects of the techniques in the first category is a superset of those in the second.

No effects, denoted in Table 1 by *no*, when the AQP methods, although they can modify a query during runtime, they do not modify their logical plan.

In this survey, three aspects of modifications to the query plan at the physical level, which may be brought about by AQP techniques, are considered: effects on the physical query plan, re-partitioning and re-scheduling. The impact of the dynamic techniques on the physical query plan (*PP* field in Table 1) is of three kinds:

Usage of adaptive operators, in order to drive the adaptivity. In many cases, extended variants of sort (*sort*), hash (*hash*), pipelining joins (*PJ*) and parallel operators (*parl*) are used in order to enhance the query evaluator with the capability to adapt dynamically to certain changes in the environment. For example, special physical operators could be used to adapt to changes in memory availability, when these changes happen during operator execution.

Operator Replacement, takes place when a particular physical operator can be replaced with a logically equivalent one at runtime in order to produce an optimised QP. For example, a hash join may be replaced by an index join, if an index on the joining attribute becomes available during runtime. As there are no restrictions on the choices that can be made, the relevant category is marked as *any* in the relevant field of Table 1.

No effects, if an AQP technique does not affect the physical plan at all. This is denoted by *no*.

Although most of the techniques could be applied to parallel and distributed environments, a few of them deal with query re-partitioning (*Par*) and re-scheduling of these partitions (*Sch*). An example of effects on the partitioning is when AQP allows partitions to be joined together or split into many subpartitions at runtime. If AQP is designed in such a way that it can explicitly modify the partitioning of a QP running in a parallel or distributed environment, then the relevant value in Table 1 is *yes*. Moreover, some techniques invoke the query optimiser during query execution and, in that

way, they may modify the partitioning, provided that they are applied to a parallel or distributed setting (denoted as *maybe* in Table 1).

The scheduling of the query plan is modified when an operation in execution is moved to another node, the number of execution nodes changes during execution, or the relative input rates for these nodes is adjusted in order to achieve optimal performance. In wide-area systems, it is also desirable to have the capability to replace a source if it fails, and to postpone the decisions on which node to execute some operations until runtime. Finally the amount of data allocated to each participating node, can be controlled in order to achieve load balance. In all the above cases the scheduling is affected. The semantics of the relevant field in Table 1 is the same as for *partitioning*.

2.2 Characteristics of Adaptive Query Processing Techniques

AQP methods vary significantly in what they attempt to adapt to. There are six main different areas of focus, as shown in the *Focus* field:

Fluctuations in memory (*mem_fl*). Systems in this category try to adapt to memory shortages and to the availability of excess memory. Memory shortages can happen due to competition from higher-priority transactions, for example. In that case, running query plans may be forced to release some or all of the resources they hold. On the other hand, executing transactions may be given additional resources as they become available, when, e.g., other queries complete and free their buffers.

User preferences (*user_pr*). Adapting to user preferences includes cases where users are interested in obtaining some partial results of the query quickly. In order to meet such needs, the system produces results incrementally, as they become available, and the user can tradeoff the time between successive updates of the running queries and the amount by which the partial and the actual results differ at each update. The user can also classify the elements of the output in terms of importance, and in that case the query evaluator adapts its behaviour in order to produce more important results earlier.

Data arrival rates (*dar*). Techniques that adapt to data arrival rates apply to parallel and distributed systems, where the response times of remote data sources are less predictable.

Actual statistics (*ac_stats*) about the data sources which are not available initially. In some cases, it is not possible at compile time to gather accurate statistics about the data sources. A solution to this problem is to collect such statistical information at runtime, thereby ensuring that they are valid for the current circumstances, and to adapt query execution based on it. Techniques that adopt this policy may change the query plan when the actual statistics have become available.

Fluctuations in performance (*per_fl*). Problems in performance fluctuation arise more often in parallel systems. In these systems, performance can degrade when even a single node experiences poor data layout on disks, high memory and CPU load, or competing data streams.

Any changes in the environment (*any*) combine elements of all the other categories. Some techniques are comprehensive, as they can adapt to many kinds of changes

in their environment, i.e. to computer resources, like memory and processor availability, and to data characteristics, like operator costs, selectivities and data arrival rates.

The aim of adaptive techniques may also differ. More specifically, AQP may aim at:

Minimising the total response time (*trt*) for a single evaluation. It is important to note that many AQP techniques use specific operators in order to achieve adaptivity. Adaptive operators perform better than static ones when there are changes in the environment. However, if all the parameters of the query are known at compile time and do not change during execution, a static QP may result in better response times.

Minimising the initial response time (*irt*). Some systems are mostly interested in minimising the time until useful information is returned to the user, according to his or her preferences. These systems are optimised to produce more, or more important, results during the early stages of the execution.

Maximising the throughput (*thr*) of the system. Some AQP techniques aim at minimising the total response time for the evaluation of a set of independent operations. Apparently, there is an interplay between the techniques of the last two categories. E.g., if there is only one operation to be evaluated, they share exactly the same objective. This is only one of the many possible ways that techniques described in the survey can interfere with each other.

Another characteristic of AQP techniques is the nature of the feedback (*FN*) they collect from the environment in order to examine whether there is scope for adaptation of the QP. Different kinds of feedback include:

- memory availability (*mem_av*),
- potential input from the user (*user_in*), such as priority ratings for different parts of the result or for the rate of updates of partial results,
- input availability (*in_av*), which is relevant to operators that may have one or more of their inputs blocked and need to check whether delayed inputs have become available,
- workload (*l*),
- data rates (*dr*), i.e. the rates at which tuples are produced, and
- statistical information (*stats*), other than those covered by the above categories, e.g. the size of each relation, the approximate number and frequency of different values for an attribute, the availability of indices, the layout of the data on disk, lists of replicated databases, etc.

For the frequency of feedback collection (*FFC*), there are two possible values:

- inter-operator (*inter*), and
- intra-operator (*intra*) frequency.

Techniques in the first category collect and may act on feedback between the execution of different physical operators in the QP. Collection can be triggered after certain physical operators have been evaluated or after special events, like the arrival of partial results from local data sources in multidatabase environments. In techniques adapting

with intra-operator frequency, feedback is collected during the evaluation of physical operators. Check-points are added to the operator execution for this purpose. In general, feedback is collected after a block of tuples has been processed. In the limit, this block consists of a single tuple, resulting in a potentially different plan for each tuple.

Techniques differ in whether they have been designed for

- uniprocessor (*U*),
- parallel (*P*), or
- distributed (*D*) environments,

as shown in the *Env* field of Table 1. This does not mean that a technique initially developed on a single processor platform cannot be applied to a parallel or distributed setting, rather it is used to indicate that the developers may not have taken into consideration some of the specific problems that arise in such settings. For example, for a QP to be optimal in a distributed environment, statistical information about network cost, replicated data sources and the available nodes is required, along with commonly available statistics which underpin good models at least for uniprocessor systems. Such statistics typically include selectivities, histograms and indices among others. The main difference between distributed and parallel DBMSs is that the former are constructed by a collection of independent, semi-autonomous processing sites that are connected via a network that could be spread over a large geographic area, whereas the latter are tightly coupled systems controlling multiple processors that are in the same location, usually in the same machine room [12].

The responsibility for adaptivity decisions (*AD*) may also be assigned differently. Such decisions can be taken:

by the physical operators (*O*), when physical operators are implemented in such a way that they can adapt their behaviour during runtime without being invoked by any other DBMS component or optimiser;

locally (*L*), if the query optimiser, or any other DBMS component, takes action during runtime to ensure that the remainder of the query plan will be evaluated in an optimal manner;

globally (*G*), which applies to parallel and distributed systems when a global view of the state of many participating nodes may be required, in order to adapt.

The last criterion used in this survey is whether a technique is realised as a physical operator (*O*), a concrete algorithm (*A*), or a system (*S*). In the first case, the whole technique can be represented in the physical query plan as a single physical operator. A technique is classified as an algorithm, if it implies extension and specific manipulation of the operators of a QP, in order to achieve adaptivity. A system comprises many autonomous techniques in an integrated computational entity.

3 Adaptive Query Operators

This section describes query operators which have the capability to adjust their behaviour according to changing conditions and the information available at runtime.

Adaptive operators continuously collect and evaluate the feedback from the environment. They act autonomously in order to adapt to changes but have limited effects on the executing QP, as they cannot perform radical changes to it. Additionally, as they are basically developed for uni-processor environments, they do not consider partitioning and scheduling issues.

Table 1. Properties of existing adaptive query processing techniques

Technique	**LL**	**PP**	**Par**	**Sch**	**Focus**	**Aim**	**FN**	**FFC**	**Env**	**AD**	**real.**
Memory Adaptive Sorting [19]	no	sort	no	no	mem_fl	trt	mem_av	intra	U	O	O
Memory Adaptive Merge-Sort [27]	no	sort	no	no	mem_fl	thr	mem_av	intra	U	O	A
PPHJ [20]	no	hash	no	no	mem_fl	trt	mem_av	intra	U	O	O
Ripple [9]	no	PJ	no	no	user_pr	irt	user_in	intra	U	O	O
XJoin [24]	no	PJ	no	no	dar/ user_pr	trt - irt	in_av	intra	U	O	O
Dynamic Reordering [23]	no	sort	no	no	user_pr	irt	user_in	intra	U	O	O
Mid-Query Reoptimisation [15]	rem	any	maybe	maybe	ac_stats	trt	stats	inter	U	L	A
Eddies [4,5]	op_or	no	no	no	any	trt	stats	intra	U	L	A
Rivers [3]	no	parl	no	yes	per_fl	trt	l/dr	intra	P	O	A
MIND [18]	op_or	any	no	yes	ac_stats	trt	stats	inter	D	G	A
Query Scrambling[1,2,26]	rem	any	no	no	dar	trt	in_av	inter	D	G	A
Pipeline Scheduler [25]	op_or	PJ	no	no	user_pr	irt	dr/ user_in	intra	D	O	A
Bouganim *et al* [6,7]	rem	PJ	maybe	maybe	dar/ mem_fl	trt	dr/ mem_av	intra	D	O/G	A
Conquest [16,17]	rem	any	yes	yes	any	trt	stats	inter	P	G	A
Tukwila [13,14]	rem	PJ	maybe	yes	any	trt - irt	stats	inter	D	O/G	S
Telegraph [11]	op_or	parl	no	yes	any	trt	stats/ l/dr	intra	D	O/L	S
dQUOB [21,22]	op_or	no	no	no	ac_stats	trt	stats	inter	D	L	A

3.1 Memory Adaptive Sorting and Hash Join

[19] introduces techniques that enable external sorting to adapt to fluctuations in memory availability. External sorting requires many buffers to run efficiently, and memory management significantly affects the overall performance. The memory-change adaptation strategy introduced is *dynamic splitting*, which adjusts the buffer usage of external sorts to reduce the performance penalty resulting from memory shortages and to take advantage of excess memory. It achieves that by splitting the merge step of external sorts into a number of sub-steps in case of memory shortage, and by combining sub-steps into larger steps when sufficient buffers are available. Alternative strategies, like *paging* and *suspension*, are non-adaptive and yield poor results [19]. Adopting memory-adaptive physical operators for sort operations avoids potential under-utilisation of memory resources (e.g. when memory reserved prior to execution based on estimates is in fact more than the memory required) or thrashing (e.g. when memory reserved prior to execution is in fact less than the memory required), and thus reduces the time needed for query evaluation. This technique is complementary to existing optimised strategies for the split phase of external sorts. In the same way, new memory-adaptive variants of merge-sort operators can also be developed.

For join execution, when the amount of memory changes during its lifetime, a family of memory adaptive hash joins, called *partially preemptible hash joins (PPHJs)*, is proposed in [20]. Initially, source relations are split and held in memory. When memory is insufficient, one partition held in memory flushes its hash table to disk and deallocates all but one of its buffer pages. The most efficient variant of PPHJs for utilising additional memory is when partitions of the inner relation are fetched in memory while the outer relation is being scanned and partitioned. This method reduces I/O and, consequently, the total response time of the query. Although it cannot adapt very well to rapid memory fluctuations, it outperforms non-memory-adaptive hybrid hash joins, which employ strategies, such as *paging* and *suspension*.

3.2 Operators for Producing Partial Results Quickly

In many applications, such as online aggregation, it is useful for the user to have the most important results produced earlier. To this end, pipelining algorithms are used for the implementation of join and sort operations. The other kind of operators are block ones, which produce output after they have received and processed the whole of their inputs.

Ripple joins [9] are a family of physical pipelining join operators that maximise the flow of statistical information during processing. They generalise block nested loops (in the sense that the roles of inner and outer relation are continually interchanged during processing) and hash joins. Ripple joins adapt their behaviour during processing according to statistical properties of the data, and user preferences about the accuracy of the partial result and the time between updates of the running aggregate. Given these user preferences, they adaptively set the rate by which they retrieve tuples from each input of the ripple join. The ratio of the two rates is reevaluated after a block of tuples has been processed.

XJoin [24] is a variant of Ripple joins. It is an operator with low memory requirements (due to partitioning of the inputs), so many such operators can be active in parallel. Apart from producing initial results quickly, XJoin is also optimised to hide intermittent delays in data arrival from slow and bursty remote sources by reactively scheduling background processing. XJoin executes in three stages: Initially, it builds two hash tables, one for each source. In the first stage, a tuple, which may reside on disk or memory, is inserted into the hash table for that input upon arrival, and then is immediately used to probe the hash table of the other input. A result tuple will be produced as soon as a match is found. The second stage is activated when the first stage blocks and it is used for producing tuples during delays. Tuples from the disk are then used to produce some part of the result, if the expected amount of tuples generated is above a certain activation threshold. The last stage is a clean-up stage as the first two stages may only partially produce the final result. In order to prevent the creation of duplicates, special lists storing specific time-stamps are used.

To obtain and process the most important data earlier, a sort-like re-ordering operator is proposed in [23]. It is a pipelining dynamic user-controllable reorder operator that takes an unordered set of data and produces a nearly sorted result according to user preferences (which can change during runtime) in an attempt to ensure that interesting items are processed first; it is a best-effort operator. The mechanism tries to allocate as many interesting items as possible in main memory buffers. When consumers request an item, it decides which item to process (i.e. it uses a pull model). The operator uses the two-phase Prefetch & Spool technique: in the first phase tuples are scanned, and uninteresting data are spoooled to an auxiliary space until input is fully consumed. In the second phase the data from the auxiliary space are read.

4 Algorithms and Systems for Adaptive Query Processing

4.1 Extensions to Adaptive Operators

A work on memory-adaptive sorting, which is complementary to [19] as discussed in Section 3.1, is presented in [27]. This method focuses on improving throughput by allowing many sorts to run concurrently, while the former focuses on improving the query response time. Because it depends on the sort size distribution, in general, limiting the number of sorts running concurrently can improve both the throughput and the response time. The method proposed enables sorts to adapt to the actual input size and changes in available memory space. For the purpose of memory adjustment, all the possible stages of a sort operation regarding memory usage are identified and prioritised. The operations may be placed in a wait queue because of insufficient memory in the system. Priority ratings are assigned to the wait queues as well. When memory becomes available, the sorts in the queue with the highest priority are processed first. A detailed memory adjustment policy is responsible for deciding whether a sort should wait or proceed with its current workspace. If it is decided to wait, further decisions on which queue to enter, and whether to stay in the current queue or to move to another, are made. Like [19], [27] has impact only on the physical query plan and adapts with intra-operator frequency.

The method in [25] for quicker delivery of initial results in pipelined query plans extends Ripple joins [9] and XJoin [24] with the capability to re-order join operators of the QP. Instead of scheduling operators, the *Dynamic Pipeline Scheduler* splits them into independent units of execution, called *streams*, and then schedules these streams. This reduces the initial response time of the query but may result in increases in the total execution time.

The Dynamic Pipeline Scheduler consists of two algorithms. In the *rate-based pipeline scheduling algorithm*, all result tuples have the same importance. The algorithm maximises the result output rate by prioritising and scheduling the flow of data across pipelined operators. Characteristics of the operators, such as cost and productivity, can be dynamically changed (due to blocked inputs, finished operators, etc). Productivity differs from selectivity in that selectivity describes how many tuples will eventually be produced, whereas productivity describes how many tuples will be produced at that point in the execution [25]. In the *importance-based tuple regulation algorithm*, the result tuples have different degrees of importance (according to the user) and the algorithm allocates more resources to the more important tuples. For the importance-rated algorithm, dynamic collection of query statistics is necessary. So, this algorithm needs to be combined with another technique that enables the collection of necessary statistics at runtime. The scheduling policy is modified on the fly in light of changes in system behaviour. Ideally, the output rates should be recomputed every time a tuple is processed by a stream, but, in fact, the scheduler is invoked less often in order to avoid large overheads.

4.2 Algorithms that Adapt to Data Arrival Rates

In the light of data arrival delays, a common approach is to minimise idle time by performing other useful operations, thus attenuating the effect of such delays. Query Scrambling [1,2,26] and a generic AQP architecture discussed in [6,7] are two representative examples in this area.

Query scrambling focuses on problems incurred by delays in receiving the first tuples from a remote data source. The system performs other useful work in the hope that the problem will eventually be resolved and the requested data will arrive at or near the expected rate from then on. In the first phase, it changes the execution order to avoid idling (which is always beneficial) following a specific procedure, and, in the second, it introduces new operations in the QP, which is risky. In query scrambling, there is a trade-off between the potential benefit of modifying the QP on the fly and the risk of increasing the total response time instead of reducing it.

[6,7] deal also with the problem of unpredictable data arrival rates, and, additionally, with the problem of memory limitation in the context of data integration systems. A general hierarchical dynamic query processing architecture is proposed. Planning and execution phases are interleaved in order to respond in a timely manner to delays. The query plan is adjusted to the data arrival rate and memory consumption. In a sub-optimal query plan, the operator ordering can be modified or the whole plan can be re-optimised, by introducing new operators and/or altering the tree shape. In general, operators that may incur large CPU idle times are pushed down the tree. Scheduling such operators as soon as possible increases the possibility that some other work is available to schedule

concurrently if they experience delays. Bushy trees are also preferred because they offer the best opportunities to minimise the size of intermediate results. Thus, in case of partial materialisation, the overhead remains low. The query response time is reduced by running concurrently several query fragments (with selection and ordering being based on heuristics) and partial materialisation is used, as mentioned above. Because materialisation may increase the total response time, a *benefit materialisation indicator (bmi)* and a *benefit materialisation threshold (bmt)* are used. A *bmi* gives an approximate indication of the profitability of materialisation and the *bmt* is its minimum acceptable value.

4.3 Algorithms that Defer Some Optimisation Decisions until Runtime

To ensure optimality in query evaluation, algorithms may wait until they have collected statistics derived from the generation of intermediate results thus far.

Kabra and deWitt [15] introduced an algorithm that detects sub-optimality in QPs at runtime, through on-the-fly collection of query statistics, and improves performance by either reallocating resources such as memory or by modifying the QP. This method aims to combat the problem of constructing QPs based on inaccurate initial statistical estimates about the data sources. To this end, a new operator, called the *statistics collector operator*, is used. The re-optimisation algorithm is heuristics-based and relies heavily on intermediate data materialisation. The *statistics collector insertion algorithm* inserts *statistics collector operators*, ensuring that these do not slow down the query by more than a specific fraction, and also assigns a potential inaccuracy level of low, medium, or high to the various estimates. So, overhead is taken into consideration and only statistics that can affect the remainder of the QP are collected.

The MIND system [18] uses a similar approach to address the problem of where to perform combination operations on partial results of a global query returned by local DBMSs in a multidatabase system. Due to the fact that useful statistics about these results are difficult for a static optimiser to estimate, decisions on where to execute operations on data that reside at remote sources are deferred until the arrival of the actual partial results. The decisions are based on both the selectivities and the cost involved.

dQUOB conceptualises streaming data with a relational data model, thus allowing the stream to be manipulated through SQL queries [21,22]. For query execution, runtime components embedded into data streams are employed. Such components are called *quoblets*. Detection of changes in data stream behaviour is accomplished by a statistical sampling algorithm that runs periodically and gathers statistical information about the selectivities of the operators into an equi-depth histogram. Based on this information, the system can reorder the select operators on the fly.

4.4 The Telegraph Project

The Telegraph project [11] combines River [3] with Eddies [4,5] to yield an adaptive dataflow system. Thus, unpredictable dataflows can be routed through computing resources in a network, resulting in a steady manageable stream of useful information. A dataflow engine is an execution environment that moves large amounts of data through

a number of operators running on an arbitrary number of machines. Such an engine is a database query processing system. Eddies can reshape dataflow graphs for each tuple they receive to maximise performance, while rivers are responsible for load balance across multiple nodes.

More specifically, Eddies is a query processing mechanism that continuously (i.e., for each tuple) reorders operators on the fly in order to adapt dynamically to changes in computing resources (e.g., memory) and data characteristics (operator costs, operator selectivities, and rates at which tuples arrive from the inputs), provided that these operators are pipelined [4,5]. Although current work focuses on uniprocessor environments, Eddies can be exploited in large federated databases. In order to insert Eddies in a query plan, joins should allow frequent and efficient reoptimisation. Such joins have frequent moments of symmetry, adaptive or non-existent synchronisation barriers, and minimal ordering constraints. A moment of symmetry is a state in which the orders of the inputs to a binary operator can be changed without modifying any state in this operator. A synchronisation barrier occurs when an input of a binary operator waits until the other input produces a specific tuple. Joins that are appropriate for Eddies include Ripple joins, pipelined hash join, Xjoin, and index joins, with the Ripple join family being the most efficient.

A River [3] is a dataflow programming environment and I/O substrate for clusters of computers to provide maximum performance even in the face of performance heterogeneity. In intra-operator parallelism, data is partitioned to be processed among system nodes. The two main innovations in a River are *distributed queues (DQ)* and *graduated declustering (GD)*, which enable the system to provide data partitioning that adapts to changes in production and consumption nodes. DQs are responsible for naturally balancing load across consumers running at different rates. GD is used for full-bandwidth balanced production, with the effect that all available bandwidth is utilised at all times, and all producers of a given data set complete near-simultaneously. While GD is transparent to programmers resulting in new ready-to-use physical operators, DQs need to be added to the operator interface manually.

4.5 Adaptive Query Processing in High-Performance Systems

AQP engines are an integral part of high-performance data mining and data integration systems, like Conquest [16,17] and Tukwila [13,14], respectively.

The Conquest query processing system enables dynamic query optimisation in a parallel environment for long-running queries following a triggering approach in response to runtime changes [16,17]. Changes relate to system parameters (e.g., new processors become available, while others may be withdrawn), and data characteristics relevant to query statistics (e.g., a buffer queue becomes empty, or selectivity is higher than initially estimated). Such changes make the re-computation of cost estimates necessary. The capabilities of Conquest for modifying a QP are very strong, especially in terms of partition scheduling, but are limited to unary operators, such as scans. Systems that can modify the remainder of a QP use materialisation techniques in order to capture and restore the intermediate state of the operators and executions. One partial exception to this is the Conquest System, which may only require the buffered records to be materialised before reconfiguring the operators.

The Tukwila [13,14] project is a data integration system in which queries are posed across multiple, autonomous and heterogeneous sources. Tukwila attempts to address the challenges of generating and executing plans efficiently with little knowledge and variable network conditions. The adaptivity of the system lies in the fact that it interleaves planning and execution and uses adaptive operators. Coordination is achieved by event-condition-action rules. Possible actions include operator reordering, operator replacement and re-optimisation of the remainder of a QP. The Tukwila system integrates adaptive techniques proposed in [15,26]. Re-optimisation is based on pipelined units of execution. At the boundaries of such units, the pipeline is broken and partial results are materialised. The main difference from [15] is that materialisation points can be dynamically chosen by the optimiser (e.g., when the system runs out of memory). Tukwila's adaptive operators adjust their behaviour to data transfer rates and memory requirements. A *collector operator* is also used, which is activated when a data source fails so as to switch to an alternative data source.

5 Discussion

In general and according to results from evaluation of prototypes, AQP techniques often attain their goal of decreasing the response time of a query in the presence of changes in the parameters they consider, or of delivering useful results as early as possible. However, in AQP, there is always a trade-off between the potential benefits from adapting to current conditions on the fly with very high frequency, and the risk of incurring large overheads.

The techniques that have more extensive capabilities in terms of the modifications they can induce in the running plan, are more expensive and risky than the others. Before applying such techniques, certain steps need to be taken (e.g., validation against a cost model) to ensure that adaptation is likely to be beneficial. These techniques are mostly applicable to parallel and distributed settings, taking the adaptivity decisions at a global level and relying heavily on the materialisation of intermediate results. This happens because in parallel and distributed settings, network costs and resource availability need to be taken into consideration, and not adapting to changes in data transfer rates or resources may have detrimental effects. On the other hand, less complex adaptive strategies (e.g. those that do not affect the logical plan of the query) are generally intended for single processor environments, where it is more common that satisfactorily accurate statistics can be obtained at compile-time.

Surprisingly, little attention has been paid to changes in the pool of available processors and data sources, which have a great impact on long-running queries in wide-area systems. As a result, most of the AQP techniques proposed so far for parallel and distributed DBMSs do not alter their partitioning and scheduling policies dynamically at runtime.

In summary, for memory fluctuations, the main strategy is to dynamically adjust the usage of memory buffers. In the light of data arrival delays, a common approach is to minimise idle time by performing other useful operations, and thus to hide such delays. The problem of inaccurate cost and statistical estimates is addressed by collecting and using the actual statistics on the fly and then comparing them with the estimates in

order to decide whether there is an opportunity for query re-optimisation. Techniques that attempt to adapt to various changes in a wide area environment follow a triggered approach. Event-condition-action rules are used, where events can be raised both by operator execution (e.g. a block has completed its processing or there is insufficient memory) and by environmental changes (e.g. a new processor has become available). In online aggregation queries where the time to provide the user with useful information quickly is more important than the time to query completion, adaptive pipelining operators need to be adopted. Non-traditional operators are also required in all the cases where the query needs to adapt with intra-operator frequency.

6 Conclusions

In this survey, existing adaptive query processing techniques have been classified and compared. The opportunities offered by such techniques to modify a query plan on the fly, have been identified. In addition, adaptive systems have been classified according to the problem they focus on, their objectives, the nature of feedback they collect from the environment and the frequency at which they can adapt. Other areas of interest include the implementation environment and which component is responsible for taking the adaptation decisions. The survey reveals the inadequacy of existing techniques to adapt to environments where the pool of resources is subject to changes. In particular, the paper provides evidence to the need for research into AQP in computing infrastructures where resource availability, allocation and costing are not, by definition, decidable at compile time. Consequently, a promising area for future work is the development of concrete cost models to evaluate whether the expected benefits from the improved query plan and decision quality compensate for the cost of collecting and evaluating feedback from the environment during execution time.

Acknowledgement

This work is supported by the Distributed Information Management Programme of the Engineering and Physical Science Research Council, through Grant GR/R51797/01. We are pleased to acknowledge their support.

References

1. L. Amsaleg, M. Franklin, and A. Tomasic. Dynamic query operator scheduling for wide-area remote access. *Distributed and Parallel Databases*, 6(3):217–246, 1998.
2. L. Amsaleg, M. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *Proc. of the Fourth International Conference on Parallel and Distributed Information Systems*, pages 208–219. IEEE Computer Society, 1996.
3. R. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. Culler, J. Hellerstein, D. Patterson, and K. Yelick. Cluster I/O with River: Making the fast case common. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 10–22, Atlanta, GA, 1999. ACM Press.
4. R. Avnur and J. Hellerstein. Continuous query optimization. Technical Report CSD-99-1078, University of California, Berkeley, 1999.

5. R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of ACM SIGMOD 2000*, pages 261–272, 2000.
6. L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. A dynamic query processing architecture for data integration systems. *IEEE Data Engineering Bulletin*, 23(2):42–48, 2000.
7. L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *Proc. of ICDE 2000*, pages 425–434, 2000.
8. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999.
9. P. Haas and J. Hellerstein. Ripple joins for online aggregation. In *Proc. of ACM SIGMOD 1999*, pages 287–298, 1999.
10. A. Hameurlain and F. Morvan. An overview of parallel query optimization in relational databases. In *11th International Workshop on Database and Expert Systems Application (DEXA 2000)*. IEEE Computer Society, 2000.
11. J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.
12. Y. Ioannidis. Query optimization. *ACM Computing Surveys*, 28(1):121–123, 1996.
13. Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution system for data integration. In *Proc. of ACM SIGMOD*, pages 299–310, 1999.
14. Z. Ives, A. Levy, D. Weld, D. Florescu, and M. Friedman. Adaptive query processing for internet applications. *IEEE Data Engineering Bulletin*, 23(2):19–26, 2000.
15. N. Kabra and D. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proc. of ACM SIGMOD 1998*, pages 106–117, 1998.
16. K. Ng, Z. Wang, and R. Muntz. Dynamic reconfiguration of sub-optimal parallel query execution plans. Technical Report CSD-980033, UCLA, 1998.
17. K. Ng, Z. Wang, R. Muntz, and S. Nittel. Dynamic query re-optimization. In *Proc. of 11th International Conference on Statistical and Scientific Database Management*, pages 264–273. IEEE Computer Society, 1999.
18. F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac. Dynamic query optimization in multidatabases. *Data Engineering Bulletin*, 20(3):38–45, 1997.
19. H. Pang, M. Carey, and M. Livny. Memory-adaptive external sorting. *The VLDB Journal*, pages 618–629, 1993.
20. H. Pang, M. Carey, and M. Livny. Partially preemptible hash joins. In *Proc. of ACM SIGMOD 1993*, pages 59–68, 1993.
21. B. Plale and K. Schwan. dquob: Managing large data flows using dynamic embedded queries. In *IEEE International High Performance Distributed Computing (HPDC)*, pages 263–270, 2000.
22. B. Plale and K. Schwan. Optimizations enabled by a relational data model view to querying data streams. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2001.
23. V. Raman, B. Raman, and J. Hellerstein. Online dynamic reordering. *The VLDB Journal*, pages 247–260, 2000.
24. T. Urhan and M. Franklin. XJoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 23(2):27–33, 2000.
25. T. Urhan and M. Franklin. Dynamic pipeline scheduling for improving interactive query performance. *The VLDB Journal*, pages 501–510, 2001.
26. T. Urhan, M. Franklin, and Laurent Amsaleg. Cost-based query scrambling for initial delays. In *Proc. of ACM SIGMOD 1998*, pages 130–141, 1998.
27. W. Zhang and P. Larson. Dynamic memory adjustment for external mergesort. *The VLDB Journal*, pages 376–385, 1997.

Improving Query Evaluation with Approximate Functional Dependency Based Decompositions

Chris M. Giannella[1], Mehmet M. Dalkilic[2],
Dennis P. Groth[2], and Edward L. Robertson[1⋆]

[1] Department of Computer Science, Indiana University Bloomington, IN 47405 USA
[2] School of Informatics, Indiana Univeristy Bloomington, IN 47405 USA
{cgiannel,dalkilic,dgroth,edrbtsn}@cs.indiana.edu

Abstract We investigate how relational restructuring may be used to improve query performance. Our approach parallels recent research extending semantic query optimization (SQO), which uses knowledge about the instance to achieve more efficient query processing. Our approach differs, however, in that the instance does not govern whether the optimization may be applied; rather, the instance governs whether the optimization yields more efficient query processing. It also differs in that it involves an explicit decomposition of the relation instance. We use approximate functional dependencies as the conceptual basis for this decomposition and develop query rewriting techniques to exploit it. We present experimental results leading to a characterization of a well-defined class of queries for which improved processing time is observed.

1 Introduction

A powerful feature of relational query languages is that identities of relational algebra may be used to transform query expressions to enhance efficiency of evaluation. Some transformations are always valid (*e.g.* reordering joins, pushing selects), but whether they enhance or degrade efficiency depends upon characteristics of the data. Other transformations are valid for instances of a particular schema due to integrity constraints (*e.g.* keys constraints). A third type of transformation holds because of characteristics of a particular instance (patterns that exist in the instance, for example, a functional dependency may hold for the instance but may not be an integrity constraint). This paper takes a characteristic of the third sort, namely functional dependency (FD), recasts it to a characteristic of the first sort, and investigates resulting implications on query evaluation. For a certain set of queries, we characterize certain cases, including some surprising ones, that yield substantial decreases in query execution time. A wider range of cases is covered in [5]. Thus, this work is suggestive rather than definitive, in that it indicates the value of an approach rather than providing a full characterization of applicable results.

As noted, an FD is a characteristic of an instance. FDs used in database design hold because a declared constraint is enforced, while query optimization may also use "discovered" FDs. The difficulty with using discovered functional dependencies is that

⋆ All authors were supported by NSF Grant IIS-0082407.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 26–41, 2002.

they are brittle, so that a minor change to the instance may mean that a dependency no longer holds. Previous work (described below) has addressed this difficulty by trying to handle the situations in which a functional dependency may break. Our work, on the other hand, uses a more supple notion, approximate functional dependency (AFD), which bends but does not break as the instance changes. The notion of AFD applies to any instance, parameterized only by "degree of approximation." To our knowledge, this represents the first use of AFDs in query optimization. AFDs are so ubiquitous in this work that explicit mention of them disappears from discussion, replaced by a relational decomposition (details given later) which, among other things, captures the "degree of approximation" in a simple manner.

Query optimization as considered here involves modifying a query such that semantics is preserved but performance is enhanced. The first step is to replace the name of the decomposed relation by an expression that recovers the table from the decomposition. One would hope that off-the-shelf query evaluators could optimize the rewritten query, but unfortunately our experiments have failed to bear this out. Thus we have defined query rewrite rules that apply specifically to our relational decomposition. The rules are very simple and we envision a preprocessor managing these rules and passing the resulting query on to the query engine (see Figure 6). As with other query rewriting, application of these rules may be blocked in a particular query, just as the standard "push down selects" rules is blocked when the select condition spans both branches of a join.

In order to appraise the decomposition and rewriting rules, we performed a number of experiments using synthetic and real-world data. Synthetic data allowed control of the "degree of approximation" of the AFD being tested. Real-world data was used to validate the synthetic data generation model. These experiments showed that rewriting yielded substantial performance improvements (on queries where they were not blocked, of course). Most surprisingly, these improvements occurred not only in cases where the AFD was quite close to an FD but also in cases that were far from an FD.

We envision our approach as only the beginning of a larger investigation of similar restructurings and rewritings. There are two different features that may vary in this investigation: classes of queries where rewritings may apply or be blocked and characteristics of instances that may suggest different restructurings. For example, multi-valued or approximate multi-valued decompositions are likely candidates.

The remainder of the paper is as follows. The next section provides a brief overview of the work leading up to our approach. Section 3 presents basic definitions and provides the theoretical background and results for the paper. Included in this are examples of two different query rewriting techniques. The results of experiments designed to test the effectiveness of the rewriting techniques are presented and discussed in Section 4. The paper ends with conclusions and future work.

2 Previous Work: SQO and Dependencies

Two separate trains of research led toward the work reported in this paper: a long-running and substantial effort in query optimization and a more recent interest in AFDs.

Semantic query optimization (SQO) began some two decades ago and was discovered independently by King[14,15] and Hammer *et al.*[9]. SQO is the use of "semantic

knowledge for optimizing queries..."[3,12,33]. Some researchers proposed that a query be rewritten prior to being passed to the query engine. The query is rewritten (to an equivalent query) according to a set of rewrite rules with the idea that the rewritten query will execute faster. One such rule allows sub-queries to be merged into the outer block (thereby eliminating the sub-query). The use of rewrite rules in this fashion has been implemented in Starburst and IBM DB2 ([27,28]).

Some researchers used additional information such as integrity constraints (ICs, *e.g.* a primary key)[29] to rewrite queries [24,25,30,32,33,34]. For example, Paulley and Larson [24,25] rewrite to eliminate unnecessary group by and distinct operations.

Fundamentally different from this use of declared constraints is that of discovering information about the *instance* itself that can be used in SQO. For example, several researchers have incorporated rules discovered from the data to rewrite queries [1,10,31,36]. For example, Bell [1] uses discovered FDs to eliminate group by and distinct operations. Recently, work by Godfrey *et al.*[6,7] demonstrates that instance knowledge yields significant, positive results in SQO. They use the concept of a soft constraint (SC), which reflects knowledge about the state of the database. Thus, SCs are weaker than traditional ICs in that they do not impose any conditions upon the database. The role they play, then, is not to ensure integrity, but to "semantically characterize the database"[7]. Godfrey *et al.* introduce two classes of SCs, absolute soft constraints (ASC) and statistical soft constraints[1] (SSC). ASCs hold completely and absolutely during the current state of the database. In contrast, SSCs do not hold completely. An obvious advantage of an ASC is that, when it holds, it can be incorporated in SQO, since, for the time it holds true, it functions essentially like an IC. ASCs can be applied to SQO for the purposes of: 1. query rewriting; 2. query plan parameterization; and 3. cardinality estimation. An advantage of an SSC is that it need not be checked against every update to verify whether it holds–rather, every so often the SSCs must be brought up to date. While Godfrey *et al.* shows how SSCs can be useful for cardinality estimation, they do not show how SSCs can be used for query rewriting.

Godfrey *et al.* then describe how SCs would be incorporated into an RDBMS (since no current system is available): 1. discovery, 2. selection, 3. maintenance. For ASCs, Godfrey *et al.* focus on both checking when an ASC is violated and maintaining ASCs. Because of their tenuous nature *i.e.*, being state-dependent, considerable care must be given to both checking and maintaining ASCs – a difficult task. This is, in fact, the "Achilles heel" of ASCs. A natural question arises from the work of Godfrey *et al.*: how can SSCs be used, if at all, for query rewriting?

While the body of work deriving from SQO is substantial, a second, more recent body of work concerning AFDs was even more significant in the genesis of this paper. The notion of a functional dependency was originally introduced as an IC for use in database design. However, more recently, research has been conducted with the view point that FDs represent interesting patterns existent in the data. In this setting, FDs are not regarded as declared constraints. Researchers have investigated the problem of efficiently discovering FDs that hold in a given instance [11,13,16,18,20,21,23,35]. Researchers have also considered the concept of an FD "approximately holding" in an

[1] The term "statistical" is meant to connote that the soft constraint holds true for some, if not all, of the data and not that any probabilistic techniques are used.

instance and have developed measures to characterize the "degree of approximation" [2,4,8,16,17,19,22,26]. The measure proposed by Kivinen and Mannila [16], g_3, correlates with the idea of "correction" that we use. Huhtala *et al.* [11] develop an algorithm for efficiently discovering all AFDs in a given instance whose g_3 approximation measure is below a user specified threshold.

3 Definitions and Theoretical Results

In this section we describe the theoretical results that form the basis of our work. We describe two query rewriting techniques. The first technique is guaranteed to always preserves correctness. The second technique is guaranteed to preserve correctness only on a special class of queries described later. Then, we describe two hypotheses about how our rewriting techniques affect query evaluation time.

We do not give proofs in this paper. Rather, we describe intuitively why the results work and illustrate with examples. Rigorous proofs can be given, but, in our opinion, do not illuminate the ideas.

3.1 Basic Notation

We assume the reader is familiar with the basic concepts of relational database theory (see [29] for a review). In what follows, we fix a relation symbol R with schema $\{A, B, C\}$. Our results can easily be generalized to apply to schema with any number of attributes; however, for simplicity, we stick with $\{A, B, C\}$. Since tables, in practice, may contain repeats, we develop our theoretical foundations with bags. Whenever we write "relation instance" or "instance" or "relation" we mean a bag and not necessarily a set.

We also make the distinction between those relational algebra (RA) operators that return sets and those that return bags. The ones that return sets (*i.e.* the ones that are duplicate removing) are denoted in the standard way: Π, σ, $\bowtie$, δ, $\cup$, and $-$ (projection, selection, natural join, renaming, union, and minus, respectively). Their bag counterparts are denoted: $\hat{\Pi}$, $\hat{\sigma}$, $\hat{\bowtie}$, $\hat{\delta}$, $\hat{\cup}$, and $\hat{-}$. We call the relational algebra over all operators (bag and set) the *bag relational algebra (bag RA)*.

Let $\mathbf{s}_1, \mathbf{s}_2$ be instances over some schema S. We say that $\mathbf{s}_1$ and $\mathbf{s}_2$ are *set equivalent*, written $\mathbf{s}_1 \equiv \mathbf{s}_2$ if $\Pi_S(\mathbf{s}_1) = \Pi_S(\mathbf{s}_2)$. In other words, $\mathbf{s}_1$ and $\mathbf{s}_2$ are equal once duplicates have been removed. Given, Q, a bag RA expression involving R, and E, another bag RA expression, let $Q[R \leftarrow E]$ be the result of replacing all occurrences of R by E. Note that the result may contain schema conflicts.

Example 1. *Let* $Q := \Pi_{A,B}(\sigma_{C=0}(R))$. *Let* $E := R_1 \hat{\cup} \hat{\Pi}_{A,B}(R_2)$ *where* R_1 *has schema* $\{A, B\}$ *and* R_2 *has schema* $\{A, B, C\}$. *Note that* E *has no schema conflicts. The schema of* E *is* $\{A, B\}$. *By definition* $Q[R \leftarrow E]$ *is* $\Pi_{A,B}(\sigma_{C=0}(E))$. *This expression has a schema conflict since* $\sigma_{C=0}$ *is being applied to* E *which has schema* $\{A, B\}$.

Consider another example. Let $Q' := \Pi_{A,B}(R \cup S)$ *(*S *has the same schema as* R*,* $\{A, B, C\}$*). By definition* $Q'[R \leftarrow E]$ *is* $\Pi_{A,B}(E \cup S)$. *This expression has a schema conflict since* E *has schema* $\{A, B\}$ *while* S *has schema* $\{A, B, C\}$. □

3.2 Horizontal-Vertical Decompositions

Let $\mathbf{r}$ be an instance of R. If the functional dependency $A \rightarrow B$ holds in $\mathbf{r}$, then $\mathbf{r}$ may be decomposed vertically as $\mathbf{r}_{AB} = \Pi_{A,B}(\mathbf{r})$, $\mathbf{r}_{AC} = \hat{\Pi}_{A,C}(\mathbf{r})$. This decomposition enjoys the property of being join lossless: $\mathbf{r} = \mathbf{r}_{AB} \hat{\bowtie} \mathbf{r}_{AC}$.

If $A \rightarrow B$ does not hold in $\mathbf{r}$, then $\mathbf{r}$ may still be decomposed. But we must first horizontally decompose $\mathbf{r}$ into two disjoint, non-empty relation instances whose union is $\mathbf{r}$, such that $A \rightarrow B$ holds in one of these relation instances. This instance is then vertically decomposed. The result is a *horizontal-vertical (HV) decomposition* of $\mathbf{r}$. The next set of definitions makes this concept precise.

$\mathbf{r}^c \subseteq \mathbf{r}$ is said to be an $A \rightarrow B$ *correction* for $\mathbf{r}$ if $A \rightarrow B$ holds in $\mathbf{r} \hat{-} \mathbf{r}^c$. Often we omit mention of $A \rightarrow B$ and $\mathbf{r}$ when clear from context and only say that $\mathbf{r}^c$ is a correction. Given correction $\mathbf{r}^c$, the HV decomposition induced is $\mathbf{r}'_{AB}$, $\mathbf{r}'_{AC}$, and $\mathbf{r}^c$ where $\mathbf{r}'$ is $\mathbf{r} \hat{-} \mathbf{r}^c$, $\mathbf{r}'_{AB}$ is $\Pi_{A,B}(\mathbf{r}')$, and $\mathbf{r}'_{AC}$ is $\hat{\Pi}_{A,C}(\mathbf{r}')$. As noted, this corresponds to the g_3 measure of [16].

Consider the instance, $\mathbf{s}$, in Figure 1. Clearly, $A \rightarrow B$ does not hold. Let $\mathbf{s}^c = \{(1,3,1), (2,1,1)\}$. Since $A \rightarrow B$ holds in $\mathbf{s}' = \{(1,2,1), (2,1,1)\}$, then $\mathbf{s}^c$ is a correction. The HV decomposition induced is depicted in Figure 2. Notice that $\mathbf{s} = (\mathbf{s}'_{AB} \hat{\bowtie} \mathbf{s}'_{AC}) \hat{\cup} \mathbf{s}^c$. Moreover, in this case, $\hat{\Pi}_{A,B}(\mathbf{s}) = \mathbf{s}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{s}^c)$. The first observation points to the lossless property of HV decompositions. The second observation points to another property that will later be shown important: if C is not needed in a query involving $\mathbf{s}$, then the join can be eliminated from the decomposition.

A	B	C
1	2	1
1	3	1
2	1	1
2	1	1

Figure 1. An instance, $\mathbf{s}$, over schema $\{A, B, C\}$

$\mathbf{s}^c$			$\mathbf{s}'_{AB}$		$\mathbf{s}'_{AC}$	
A	B	C	A	B	A	C
1	3	1	1	2	1	1
2	1	1	2	1	2	1

Figure 2. Induced HV non-minimal decomposition, $\mathbf{s}^c$, $\mathbf{s}'_{AB}$, $\mathbf{s}'_{AC}$

In the previous example, $\mathbf{s}^c$ is not minimal since a smaller correction can be found. For example, $\mathbf{s}^c = \{(1,3,1)\}$ is also a correction. The HV decomposition induced is depicted in Figure 3. Notice that $\mathbf{s} = (\mathbf{s}'_{AB} \hat{\bowtie} \mathbf{s}'_{AC}) \hat{\cup} \mathbf{s}^c$. Moreover, $\hat{\Pi}_{A,B}(\mathbf{s}) \equiv$

$\mathbf{s}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{s}^c)$ (but equality does not hold). As in the previous example, the first observation points to the lossless property of the decomposition, and the second observation points to the property that if C is not needed in queries involving $\mathbf{s}$, then the join can be eliminated from the decomposition. However, this example shows that the property must be weakened to set equality rather than equality (see Theorem 1).

$\mathbf{s}^c$			$\mathbf{s}'_{AB}$		$\mathbf{s}'_{AC}$	
A	B	C	A	B	A	C
1	3	1	1	2	1	1
			2	1	2	1
					2	1

Figure 3. Induced HV minimal decomposition, $\mathbf{s}^c$, $\mathbf{s}'_{AB}$, $\mathbf{s}'_{AC}$

The point of these last two examples was (i) to illustrate two important properties of HV decompositions, and (ii) to point out that these properties do not depend on the correction being minimal. Our query rewriting techniques rely on these properties. If these properties were dependent on the decomposition being minimal, then maintaining the decomposition in the presence of updates would be difficult. But, these properties are not dependent on the correction being minimal. Hence, we gain much greater maintenance flexibility. Nonetheless, maintenance is still a difficult issue. Due to space constraints, we defer description of a simple method for maintaining the decomposition to an extended version of this paper [5]. The performance of the maintenance method is not analyzed in [5]; this is left as future work.

In short, the fundamental properties of HV decompositions needed for our query rewriting techniques are the following.

Theorem 1.

1. $\mathbf{r} = (\mathbf{r}'_{AB} \hat{\bowtie} \mathbf{r}'_{AC}) \hat{\cup} \mathbf{r}^c$.
2. $\hat{\Pi}_{A,B}(\mathbf{r}) \equiv \mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)$.

Part 1 shows how an instance can be decomposed using AFD $A \rightarrow B$. This result forms the basis of our first rewriting technique. Part 2 shows how the decomposition can be simplified if C is not needed. This result forms the basis of our second rewriting technique. Later, we will discuss the second technique and the setting in which it is applied. In that setting, it will be made clear why we only need set equivalence ($\equiv$) rather than equality ($=$).

3.3 Query Rewriting: Technique I

HV decompositions, in a sense, "expose" structural information about the instance. Our basic idea is to rewrite queries using the decomposition such that structural information is exposed to the DBMS query evaluator. Our thinking is that the optimizer could use this structure to improve query evaluation (pushing selects into the decomposition, for

example, may result in speedup). Theorem 1, part 1 provides the foundation of our first rewriting technique (illustrated by the following example). Consider an example query, Q:

```
Select Distinct R1.A, R1.B
From R as R1
Where R1.A = 0.
```

Expressed in bag RA terms, $Q := \Pi_{A,B}(\sigma_{A=0}(R))$. If the HV decomposition $\mathbf{r}'_{AB}$, $\mathbf{r}'_{AC}$, $\mathbf{r}^c$ is kept in the database, then, by Theorem 1, part 1, we have $Q(\mathbf{r}) = \Pi_{A,B}(\sigma_{A=0}((\mathbf{r}'_{AB} \hat{\bowtie} \mathbf{r}'_{AC}) \hat{\cup} \mathbf{r}^c))$. So, Q can be rewritten as Q_1:

```
Select Distinct R1.A, R1.B
From ((Select RAB.A as A, RAB.B as B, RAC.C as C
       From RAB, RAC
       Where RAB.A=RAC.A)
       Union All
       (Select A,B,C
       From Rc)) as R1
Where R1.A = 0.
```

This technique of query rewriting preserves correctness on any SQL query, *i.e.* the rewritten query is well-formed (no schema conflicts) and, when handed to the DBMS query evaluator, produces exactly the same result as the rewritten query. If R occurs more than once (*e.g.* "R as R2"), then each occurrence of R is replaced as above. We call this *Rewriting Technique I*.

3.4 Query Rewriting: Technique II

In the previous subsection, Q was rewritten as Q_1. However, Q has properties that allow further rewriting. First observe that attribute C does not appear in the output schema of Q and is not used elsewhere in the query. As a result, only the attributes A and B are needed; C can be projected out: $Q(\mathbf{r}) = Q(\hat{\Pi}_{A,B}(\mathbf{r}))$.

Now by Theorem 1 part 2, we have $Q(\mathbf{r}) \equiv \Pi_{A,B}(\sigma_{A=0}(\mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)))$. But since $\Pi_{A,B}$ appears at the top level of Q (hence duplicates are removed from the output), then we may replace set equality by equality: $Q(\mathbf{r}) = \Pi_{A,B}(\sigma_{A=0}(\mathbf{r}'_{AB} \hat{\cup} \hat{\Pi}_{A,B}(\mathbf{r}^c)))$. So, Q may be rewritten as Q_2:

```
Select Distinct R1.A, R1.B
From ((Select RAB.A as A, RAB.B as B
       From RAB)
      Union All
      (Select Rc.A as A, Rc.B as B
       From Rc)) as R1
Where R1.A = 0.
```

The decomposition join has been eliminated and we expect that Q_2, when handed to the DBMS query evaluator, will evaluate faster than Q_1. Moreover, we would expect that Q_2 will evaluate faster than Q if $\mathbf{r}'_{AB}$ and $\mathbf{r}^c$ are not large relative to $\mathbf{r}$. If R occurs more than once in Q, then replace each occurrence as above. We call this *Technique II*.

Application of Technique II. In the previous subsection, Q was rewritten as Q_1 and then further rewritten as Q_2. While rewriting as Q_1 (Technique I) always preserves correctness, rewriting as Q_2 (Technique II) does not. We would like to isolate syntactic properties of Q that guarantee that Technique II preserves correctness.

Since Technique II replaces R by an expression whose schema is $\{A, B\}$, then the original query cannot involve C or else the rewritten query may have schema conflicts. The top bag RA expression in Example 1 illustrates how such a schema conflict can arise. Moreover, if the original query involves union or minus, then schema conflicts can also arise. The bottom RA expression in Example 1 illustrates how schema conflicts can arise in the presence of union. However, even when the original query does not involve C, the rewritten query may not produce the same output (but is set equivalent to the original). Consider the following example query, which is the same as query Q in subsection 3.3 except that the "Distinct" has been removed.

```
Select R1.A, R1.B
From R as R1
Where R1.A = 0.
```

When applying Technique II to this query, the rewritten query may not be equivalent because of duplicates in the output. For a similar reason, aggregates cannot be supported by Technique II.

We give a result (Corollary 1) that defines a general class of queries over which Technique II is guaranteed to preserve correctness. First, though, we state a theorem that defines a general class of bag RA expressions over which Technique II is guaranteed to preserve correctness *up to set equivalence*. Corollary 1 falls out immediately by restricting the class further to those which have Π at their top level.

Let Q be any bag RA expression. Let E be the bag RA expression $R'_{AB} \hat{\cup} \hat{\Pi}_{AB}(R^c)$ where R'_{AB} is a relation symbol over schema $\{A, B\}$ and R^c is a relation symbol over schema $\{A, B, C\}$. Let Q_2 be $Q(R \leftarrow E)$.

Theorem 2. *If Q does not involve union or minus, and the output schema of Q does not contain C, and C does not appear in Q, then Q_2 is well-defined and $Q(\mathbf{r}) \equiv Q_2(\mathbf{r}'_{AB}, \hat{\Pi}_{A,B}(\mathbf{r}^c))$.*

We say that a bag RA expression Q is *top-level distinct* if it is of the form $\Pi_{...}(Q')$. A top-level distinct expression always returns a set.

Definition 1. *We say that a bag RA expression, Q, is* join elimination re-writable (JE-rewritable) *if (i) Q is top-level distinct, (ii) Q does not involve union or minus, (iii) The output schema of Q does not contain C, and (iv) C does not appear in Q (*i.e. *is not part of any projection, selection, or renaming condition, and no join has C among its join attributes).*

Take note that the first example in subsection 3.3 is the SQL of the JE-rewritable, bag RA expression $\Pi_{A,B}(\sigma_{A=0}(\mathbf{r}))$. Theorem 2 implies the following result that defines the class of bag RA expressions over which Technique II preserves correctness.

Corollary 1. *If Q is JE-rewritable, then Q_2 is well-formed and $Q(\mathbf{r}) = Q_2(\mathbf{r}'_{AB}, \hat{\Pi}_{A,B}(\mathbf{r}^c))$.*

The SQL queries equivalent to the JE-rewritable, bag RA expressions are the queries on which Technique II is guaranteed to preserve correctness. We call these *JE-rewritable queries*.

3.5 Hypotheses

We have developed two query rewriting techniques: $Q \rightarrow Q_1$ (Technique I), $Q \rightarrow Q_2$ (Technique II). The first technique is guaranteed to preserve correctness for any query. The second is only guaranteed to preserve correctness for JE-rewritable queries. Q_1 may evaluate faster than Q when handed to the DBMS query evaluator, because Q_1 exposes more of the structure of $\mathbf{r}$, thereby allowing the optimizer to possibly take advantage of this structure. We arrive at our first hypothesis. Let $time(Q)$ denote the time required by the DBMS query evaluator to evaluate Q (likewise, define $time(Q_1)$).

Hypothesis 1 (Rewriting Tech. I). *Given query Q, $time(Q_1) < time(Q)$.*

We expect Q_2 to evaluate faster than Q_1 because of the elimination of the join in the HV decomposition. Moreover, we expect Q_2 to evaluate faster than Q when $\mathbf{r}'_{AB}$ and $\mathbf{r}^c$ are small relative to $\mathbf{r}$. Let $|\mathbf{r}|$, $|\mathbf{r}'_{AB}|$, and $|\mathbf{r}^c|$ denote the number of tuples in $\mathbf{r}$, $\mathbf{r}'_{AB}$, and $\mathbf{r}^c$, respectively. Let $adom(A, \mathbf{r}'_{AB})$ denote the active domain of A in $\mathbf{r}'_{AB}$ (likewise define $adom(A, \mathbf{r})$). By definition of HV decompositions, $|\mathbf{r}'_{AB}| = |adom(A, \mathbf{r}'_{AB})|$. We use $\frac{|\mathbf{r}|}{|adom(A,\mathbf{r}'_{AB})|+|\mathbf{r}^c|}$ to quantify the size of $\mathbf{r}'_{AB}$ and $\mathbf{r}^c$ relative to $\mathbf{r}$. For example, if $|adom(A, \mathbf{r}'_{AB})|$ is two percent of $|\mathbf{r}|$ and $|\mathbf{r}^c|$ is twenty percent of $|\mathbf{r}|$, then $|\mathbf{r}|$ is 4.55 times as large as $|\mathbf{r}'_{AB}| + |\mathbf{r}^c|$.

Hypothesis 2 (Rewriting Tech. II). *Given JE-rewritable query Q, if $\frac{|\mathbf{r}|}{|adom(A,\mathbf{r}'_{AB})|+|\mathbf{r}^c|} > 1$, then $time(Q_2) < time(Q)$. Moreover, as $\frac{|\mathbf{r}|}{|adom(A,\mathbf{r}'_{AB})|+|\mathbf{r}^c|}$ increases, then $time(Q) - time(Q_2)$ also increases.*

4 Experimental Results

Datasets for the experiments were generated randomly, controlling for the size of the relation, the size of the correction, and the size of the active domains for A and B. The order of tuples was permuted to avoid long sequences of tuples in the dataset with the same A and B value. The table used for the join query was generated from $\mathbf{r}$ by projecting out the unique B values and adding a description field, which resulted in the schema $S = \{B, D\}$.

Fixing the size of the active domains provides the benefit of controlling for the selectivity of the queries that select rows based on a constant. The constant used for

these queries was the value representing the median frequency. We fixed the size of $adom(B, \mathbf{r})$ to be 100 for each experiment. We considered three sizes for $adom(A, \mathbf{r})$: 100, 1000, and 10000. The results for each of these sizes were similar, so we report the case where $adom(A, \mathbf{r}) = 1000$. The other sizes, as well as a more thorough description of the synthetic data generation procedure, are reported in [5].

For all experiments, the size of the relation was 500,000 tuples. The size of the correction ranged from 0–90% in 10% increments. Note that the 0% correction represents a case where the functional dependency $A \rightarrow B$ holds. The decompositions generated for the experiments were minimal, so $|adom(A, \mathbf{r})| = |adom(A, \mathbf{r}'_{AB})|$ in each case.

The datasets were stored as tables in an Oracle (Version 8.05) database running on Sun UltraSparc 10 server running Solaris 7 equipped with 256 MB of RAM. No indexes were generated for any of the tables. However, statistics were generated for all tables, allowing for cost-based optimization.

Queries were executed from a Java 1.3 application using the JDBC thin client interface provided by Oracle. Each query was executed 5 times, with the mean completion time to return the final row in the result reported. The standard deviations we observed were small and are omitted from this report. We used the "NOCACHE" optimizer directive on each query to avoid reusing the cache. [2]

4.1 Testing Hypotheses

We tested Hypothesis 1 on the following query:

```
Select Distinct R1.A,R1.B,R1.C
From R as R1
Where R1.A=constant
```

rewritten using Technique I as:

```
Select Distinct R1.A,R1.B,R1.C
From (Select RAB.A,RAB.B,RAC.C
      From RAB,RAC Where RAB.A=RAC.A
       Union All
      Select A,B,C From Rc) as R1
Where R1.A=constant
```

The results show that the rewritten query performs worse than the original query (due to space constraints, detailed results are omitted, see [5]). Consequently, it does not appear that exposing the structure to the optimizer yielded any benefits. Interestingly, the worst performance occurs in the case of a perfect functional dependency. We conclude that our hypothesis is incorrect. We believe that the reason for failure is directly related to the join in the rewritten query. A closer examination of the query plans makes clear why we did not experience an improvement. The plan generated for the

[2] This directive specifies that blocks are placed in the least recently used part of the LRU list in the buffer cache when a full table scan is performed. This was necessary when repeating a query to avoid misleading results.

rewritten query did not push the selects into the query. Instead, the original relation was materialized and then scanned to get the answer. As future work, we plan on augmenting Technique I to include pushing of selections and projections into the decomposition.

Rewriting Technique II, when applicable, does not produce queries requiring the join. As a result, we expect that the performance of queries rewritten with Technique II will be clearly faster than those rewritten with Technique I. Now we test whether queries rewritten with Technique II are faster than the original queries.

To test hypothesis 2, the following four JE-rewritable queries were used.

1. `Select Distinct R1.A,R1.B From R as R1 Where R1.A=constant`
2. `Select Distinct R1.A,R1.B From R as R1 Where R1.B=constant`
3. `Select Distinct R1.A,R1.B,S.D From R as R1,S Where R1.B=S.B`
4. `Select Distinct R1.A,R1.B From R as R1`

Each query was rewritten as follows:

1. ```
   Select Distinct R1.A,R1.B
   From (Select A,B From RAB Union All Select A,B From Rc)
   as R1 Where R1.A=constant
   ```
2. ```
   Select Distinct R1.A,R1.B
   From (Select A,B From RAB Union All Select A,B From Rc)
   as R1 Where R1.B=constant
   ```
3. ```
   Select Distinct R1.A,R1.B,S.D
   From (Select A,B From RAB Union All Select A,B From Rc)
   as R1,S Where R1.B=S.B
   ```
4. ```
   Select Distinct R1.A,R1.B
   From (Select A,B From RAB Union All Select A,B From Rc) as R1
   ```

Figure 4 shows the results for Queries 1 and 2; figure 5 shows the results for Query 3. For Queries 1 and 2, we see that the rewritten queries perform better than the original query when the size of the correction is small. As the size of the correction increases, the performance of the rewritten queries degrades in a linear fashion. Queries 3 and 4 exhibited similar behavior to each other. See the extended version of the paper for details [5].

4.2 Discussion

Hypothesis 1 was incorrect due to the cost of the join introduced by the decomposition and the fact that the optimizer did not take advantage of the decomposition. For example, as shown in the queries below, the optimizer could have pushed selects into the decomposition. The first query is the original query rewritten with Technique I used to test hypothesis 1. The second query exploits the structure of the decomposition by pushing selects as deep as possible, which is one of the most basic query optimization strategies. In these experiments the query optimizer did not take advantage of the decomposition.

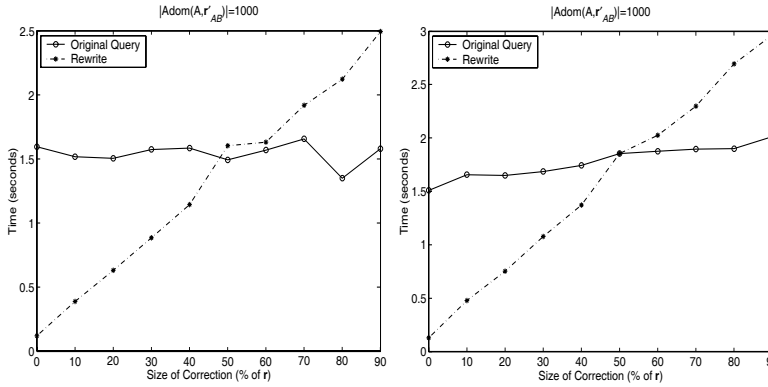


Figure 4. Comparing the mean execution time for Queries 1 and 2 to the size of the correction

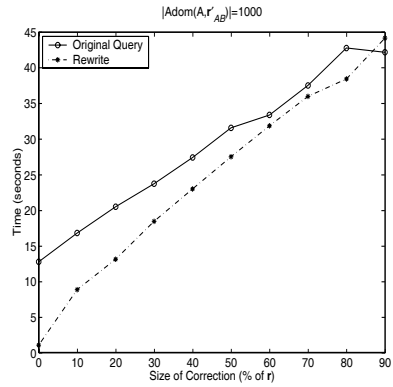


Figure 5. Comparing the mean execution time for Query 3 to the size of the correction

```
1. Select Distinct R1.A,R1.B,R1.C
   From (Select RAB.A,RAB.B,RAC.C From RAB,RAC Where
         RAB.A=RAC.A Union All Select A,B,C From Rc) as R1
         Where R1.A=constant

2. Select Distinct R1.A,R1.B,R1.C
   From (Select RAB.A,RAB.B,RAC.C From RAB,RAC
         Where RAB.A=constant and RAC.A=constant and
         RAB.A=RAC.A Union All Select A,B,C From Rc
         Where A=constant) as R1
```

Queries that were able to use Technique II outperformed the original queries until the size of the correction exceeded 50% - a certainly robust technique. We were

pleasantly surprised that the breaking point was so high. After all, it seems intuitive that the decomposition will perform better when the AFD is close to an FD. It is surprising though, that the AFD $A \rightarrow B$ in the original relation can be significantly far from being an FD and result in better performing queries. Observe that for our experimental datasets, the size of the active domain does not materially affect the relative performance of the queries.

In addition to experiments against synthetic data, we tested Technique II on U.S. Census data (see [5] for details). These results demonstrate that the technique was beneficial when applied to real-world data. In fact, the rewritten queries performed better against the census data than against the synthetic data. However, more experiments against real data are necessary to determine the general applicability of the technique.

5 Conclusion and Future Work

In this paper we investigated an approach (paralleling recent work extending SQO) to improve query evaluation. Our approach is based on decomposing relation instances with respect to AFDs and rewriting queries to take advantage of these decompositions (HV decompositions). The primary idea is that the semantic information contained in an AFD can be exposed by creating an HV decomposition of the relation instance. This information can then be exploited to speed up query evaluation: rewrite the query to use the decomposition instead of the original relation, then, issue the rewritten query to the DBMS query engine instead of the original query. Two things in particular should be pointed out about how our approach fits into the literature. First, it represents (to our knowledge) the first use of AFDs in query evaluation. Second, it addresses the question raised by the work of Godfrey *et al.* described in Section 2. Our original motivation was not to specifically address the question. We discovered after we had obtained our results that they could be used to address the question.

We investigated two rewriting techniques. Technique I replaces all occurrences of the relation symbol by its decomposition. This technique is guaranteed to preserve correctness on all SQL queries. The motivation was that the optimizer can take advantage of the decomposition and produce a more efficient plan. Our experiments, however, point out that this was not the case. The introduction of the decomposition join caused the rewritten queries to run more slowly. Technique II replaces all occurrences of the relation symbol by the decomposition without the join. This technique is only guaranteed to preserve correctness on a special class of queries (JE-rewritable queries). However, our experiments show that queries rewritten with this technique tend to evaluate significantly faster than the original query, provided the correction was not too large.

Our results suggest an architecture designed around AFD-based decompositions. We have observed that Technique II can offer significant speed-up, but, it can only be applied to JE-rewritable queries. So, the original relation should be kept along with its HV decomposition. The preprocessor examines the query to see if it is JE-rewritable. If so, and if the correction is not too large, then the query is rewritten. Otherwise, the query is not rewritten and the original query is handed to the DBMS query engine. See Figure 6. Take note that the original relation and its HV decomposition are kept in this architecture. The extra space required is substantial but we feel that improving query

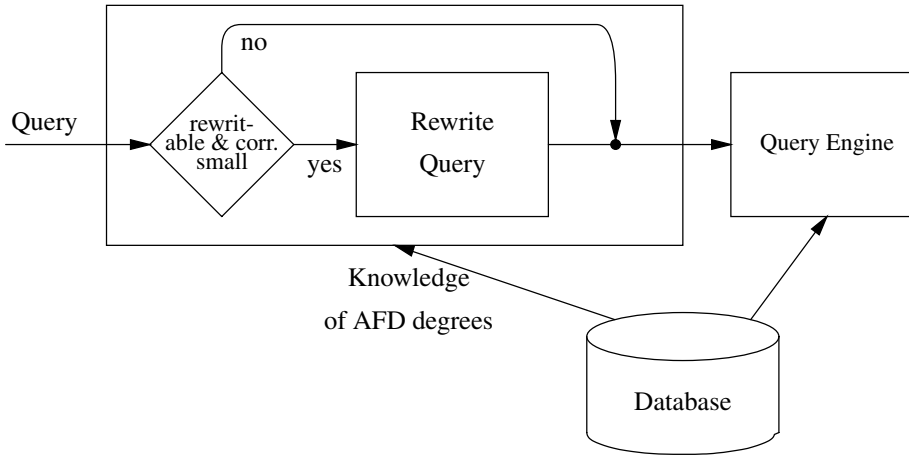


Figure 6. The preprocessor rewrites JE-rewritable queries provided that the correction is not too large

evaluation time at the expense of disk space is a worthwhile trade-off. Maintenance of the decomposition is a significant concern, but we leave this to future work.

We described the techniques on a three column table for ease of exposition. However, the techniques apply in the same way to many columns. We did not investigate how the results scale with the number of columns involved. The crucial issue here is that the size difference between the original instance and the instance projected on the dependency (projected on A, B in our case) diminishes with the number of columns involved in the dependency. Since the speedup obtained by Technique II depends on the size difference between the original instance and the projected one, then we would expect the speedup to decrease as more columns are involved.

There are a number of directions for future work. 1. Address the primary drawback to Technique II: it is guaranteed to preserve correctness only on JE-rewritable queries. In particular, modify our approach to handle aggregate operations like count and sum. This can be achieved by modifying the construction of $\mathbf{r}'_{AB}$ to keep counts. The significant issue is then sharpened: how to rewrite queries to use the counts. 2. Investigate extending Technique I to push selections and projections into the decomposition. 3. Investigate the use of other decompositions. A natural candidate is a decomposition induced by approximate multi-valued dependencies, which creates two correction relations $\mathbf{r}^+, \mathbf{r}^-$. 4. Investigate methods for maintaining an HV decomposition. Does the extra time required for processing inserts and deletes eclipse the gains in query evaluation? Also, when should corrections be reorganized to minimize $\mathbf{r}^c$? The $\mathbf{r}^+$, $\mathbf{r}^-$ decomposition mentioned above is intriguing because deletes are processed via $\mathbf{r}^-$.

In closing we point out that the primary purpose was to introduce the idea of exploiting AFDs in query evaluation via HV decompositions and demonstrate that the idea is fertile grounds for future work. We feel that we have achieved this goal.

References

1. Bell S. Deciding distinctiveness of query results by discovered constraints. *Proc. of the 2nd International Conf. on the Practical Application of Constraint Technology*, pages 399–417, 1996.
2. Cavallo R. and Pittarelli M. The theory of probabilistic databases. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pages 71–81, 1987.
3. Chakravarthy U., Grant J., and Minker J. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.
4. Dalkilic M. and Robertson E. Information dependencies. In *Proc. of the Nineteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 245–253. ACM, 2000.
5. Giannella C., Dalkilic M., Groth D., and Robertson E. Using horizontal-vertical decompositions to improve query evaluation. Technical report, Computer Science, Indiana University, Bloomington, Indiana, USA, 2002.
6. Godfrey P., Grant J., Gryz J., and Minker J. *Logics for Databases and Information Systems*, pages 265–307. Kluwer Academic Publishers, Boston, MA, 1998.
7. Godfrey P., Gryz J., and Zuzarte C. Exploiting constraint-like data characterizations in query optimization. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data*, pages 582–592, May 2001.
8. Goodman L. and Kruskal W. Measures of associations for cross classifications. *Journal of the American Statistical Association*, 49:732–764, 1954.
9. Hammer M. and Zdonik S. Jr. Knowledge-based query processing. In *Proc. of the Sixth Intl. Conf. on Very Large Data Bases*, pages 137–147, Montreal, Canada, Oct. 1980.
10. Hsu C. and Knoblock C. Using inductive learning to generate rules for semantic query optimization. In Fayyad U. and Piatetsky-Shapiro G., editor, *Advances in Knowledge Discovery and Data Mining*, 1996.
11. Huhtala Y., Kärkkäinen J., Porkka P., and Toivonen H. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings 14th International Conference on Data Engineering*, pages 392–401. IEEE Computer Society Press, February 1998.
12. Jarke J., Clifford J., and Vassiliou Y. An optimizing PROLOG front-end to a relational query system. In *Proc. of the ACM SIGMOD Conf.*, pages 296–306, 1984.
13. Kantola M., Mannila H., Räihä K., and Siirtola H. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7:591–607, 1992.
14. King J. QUIST–A system for semantic query optimization in relational databases. In *Proc. of the 7th International Conf. on Very Large Data Bases Cannes*, pages 510–517, Cannes, France, Sept. 1981. IEEE Computer Society Press.
15. King J. Reasoning about access to knowledge. In *Proc. of the Workship on Data Abstraction, Databases, and Conceptual Modelling*, pages 138–140. SIGPLAN Notices, Jan. 1981.
16. Kivinen J. and Mannila H. Approximate dependency inference from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.
17. Lee T. An information-theoretic analysis of relational databases–part I: Data dependencies and information metric. *IEEE Transactions on Software Engineering*, SE-13(10):1049–1061, October 1987.
18. Lopes S., Petit J., and Lakhal L. Efficient discovery of functional dependencies and Armstrong relations. In *Lecture Notes in Computer Science 1777 (first appeared in the Proceedings of the Seventh International Conference on Extending Database Technology (EDBT))*, pages 350–364, 2000.

19. Malvestuto F. Statistical treatment of the information content of a database. *Information Systems*, 11(3):211–223, 1986.
20. Mannila H. and Räihä K. Dependency inference. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pages 155–158, 1987.
21. Mannila H. and Räihä K. Algorithms for inferring functional dependencies. *Data & Knowledge Engineering*, 12:83–99, 1994.
22. Nambiar K. Some analytic tools for the design of relational database systems. In *Proceedings of the 6th International Conference on Very Large Databases (VLDB)*, pages 417–428, 1980.
23. Novelli N., Cicchetti R. FUN: An efficient algorithm for mining functional and embedded dependencies. In *Lecture Notes in Computer Science 1973 (Proceedings of the 8th International Conference on Database Theory (ICDT))*, pages 189–203, 2001.
24. Paulley G. *Exploiting Functional Dependence in Query Optimization*. PhD thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Sept. 2000.
25. Paulley G. and Larson P. Exploiting uniqueness in query optimization. In *Proc. of the 10th ICDE*, pages 68–79, 1994.
26. Piatetsky-Shapiro G. Probabilistic data dependencies. In *Machine Discovery Workshop (Aberdeen, Scotland)*, 1992.
27. Pirahesh H., Hellerstein J., and Hasan W. Extensible/rule based query rewrite optimization in starburst. In *Proc. 1992 ACM-SIGMOD Int. Conf. Management of Data*, pages 39–48, May 1992.
28. Pirahesh H., Leung T.Y., and Hasan W. A rule engine for query transformation in starburst and IBM DB2 C/S DBMS. In *Proc. 13th Int. Conf. on Data Engineering (ICDE)*, pages 391–400, April 1997.
29. Ramakrishanan R. and Gehrke J. *Database Management Systems 2nd Edition*. McGraw-Hill Higher Education, Boston, MA, 2000.
30. Sagiv Y. Quadratic algorithms for minimizing join in restricted relational expressions. *SIAM Journal of Computing*, 12(2):316–328, May 1983.
31. Shekhar S., Hamidzadeh B., Kohli A., and Coyle M. Learning transformation rules for semantic query optimization: a data-driven approach. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):950–964, December 1993.
32. Shenoy S. and Ozsoyglu Z. A system for semantic query optimization. In *Proc. 1987 ACM-SIGMOD Int. Conf. Management of Data*, pages 181–195, May 1987.
33. Shenoy S. and Ozsoyglu Z. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):344–361, Sept. 1989.
34. Sun W. and Yu C. Semantic query optimization for tree and chain queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):136–151, February 1994.
35. Wyss C., Giannella C., and Robertson E. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *Lecture Notes in Computer Science 2114 (Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery)*, pages 101–110, 2001.
36. Yu C. and Sun W. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):362–374, September 1989.

The AutoMed Schema Integration Repository

Michael Boyd, Peter McBrien, and Nerissa Tong

Dept. of Computing, Imperial College, London SW7 2BZ
{mboyd,pjm,nnyt98}@doc.ic.ac.uk

1 Introduction

The AutoMed EPSRC project, jointly run by Birkbeck and Imperial Colleges, has as part of its aims the implementation of some previous theoretical work, which we now term the AutoMed approach to database schema and data integration. In this approach [2] the integration of schemas is specified as a sequence of bidirectional transformation steps, incrementally adding, deleting or renaming constructs, so as to map one schema to another schema. Optionally associated with each transformation step is a query expression, describing how instances of the construct (i.e. the data integration) can be obtained from the other constructs in the schema. One feature of AutoMed is that it is not restricted to use a particular modelling language for the description of database models and their integration. Instead, it works of the principle that data modelling languages such as ER, relational, UML, etc. are graph-based data models, which can be described [3] in terms of constructs in the **hypergraph data model (HDM)** [5]. The implementation of AutoMed provides only direct support for the HDM, and it is a matter of configuration of AutoMed to provide support for a particular variant of a data modelling language.

In this paper we describe the first version of the repository of the AutoMed toolkit (available from `http://www.doc.ic.ac.uk/automed/`). This is a Java API, that uses a RDBMS to provide a persistent storage for data modelling language descriptions in the HDM, database schemas, and transformations between those schemas [1]. The repository also provides some of the shared functionality that tools accessing the repository may require.

The AutoMed repository has two logical components, assessed via one API. The **model definitions repository (MDR)** allows for the description of how a data modelling language is represented as combinations of nodes, edges and constraints in the HDM. It is used by AutoMed 'experts' to configure AutoMed so that it can handle a particular data modelling language. The **schema transformation repository (STR)** allows for schemas to be defined in terms of the the data modelling concepts in the MDR. It also allows for transformations to be specified between those schemas. Most AutoMed tools and users will be concerned with editing this repository, as new databases are added to the AutoMed repository, or those databases evolve [4].

Before describing how the MDR and STR APIs function, we give in Fig. 1 an example of two schemas in a variant of the ER modelling language, together with a sequence of transformations which map between the two schemas.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002 LNCS 2405, pp. 42–45, 2002.

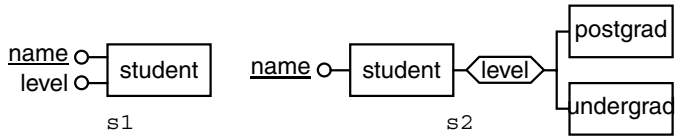


Figure 1. Example of Schema Integration

The two schemas illustrate the well known attribute-generalisation equivalence. Instances of undergrad are those instances of student which have ug as the value of the level attribute, and postgrad are those instances with pg as the level.

transformation s1→s2
① *addEntity* ⟨⟨undergrad⟩⟩ $\{x \mid \langle x, \textit{'ug'}\rangle \in \langle\langle$student,level$\rangle\rangle\}$
② *addEntity* ⟨⟨postgrad⟩⟩ $\{x \mid \langle x, \textit{'pg'}\rangle \in \langle\langle$student,level$\rangle\rangle\}$
③ *addGen* ⟨⟨level,total,student,undergrad,postgrad⟩⟩
④ *delAttribute* ⟨⟨student,level⟩⟩
$\{x, y \mid x \in \langle\langle$undergrad$\rangle\rangle \wedge y = \textit{'ug'} \vee x \in \langle\langle$postgrad$\rangle\rangle \wedge y = \textit{'pg'}\}$
$(\langle _, y\rangle \in \langle\langle$student,level$\rangle\rangle \rightarrow y = \textit{'ug'} \vee y = \textit{'pg'})$

2 Describing a Data Modelling Language in the MDR

In [3] we proposed a general technique for the modelling of any structured data modelling language in the HDM. This has been used as a basis for the design of the MDR. First, to specify a modelling language, we create an instance of the Model class, with an associated identifying name:

```
Model er=Model.createModel("er");
```

Constructs in the modelling language are then defined by selecting one of four variants [3] — nodal, link nodal, link, and constraint — and describing details of the scheme of the construct. For example, entities in an ER modelling language correspond to nodes in the underlying HDM, which we identify by the general scheme template of ⟨⟨entity_name⟩⟩. Thus in the API, we create a new nodal Construct called "entity" in the "er" Model created earlier, and add to its scheme a new HDM node to hold the entity_name.

```
Construct ent=er.createConstruct("entity",Construct.CLASS_NODAL,true);
ent.addNodeNameScheme();
```

An attribute in an ER model must always be attached to an already existing entity. This is an example of a link nodal construct. Its scheme takes the general form ⟨⟨entity_name,attribute_name,cardinality⟩⟩ where entity_name must already exist as the name of the entity (hence the second line below), attribute_name is the name of a new node holding instances of the attribute (hence the third line), and cardinality will be one of key, notnull or null, and acts as a constraint on instances of the attribute (hence the fourth line).

```
Construct att=
      er.createConstruct("attribute",Construct.CLASS_LINK_NODAL,true);
att.addReferenceScheme(ent);
att.addNodeNameScheme();
att.addConstraintScheme(false);
```

Other ER constructs can be defined in a similar manner, which can be found in the full version of the program available on the AutoMed website.

3 Describing Schemas and Transformations in the STR

A database schema is held in the STR, as a set of SchemaObject instances, each of which must be based on Construct instances that have been created in the MDR. The schema is created by building an instance of Schema, and then populating the schema with instances of SchemaObject.

```
Schema s=Schema.createSchema("s1",false);
SchemaObject student=s.createSchemaObject(ent,new Object[]{"student"});
s.createSchemaObject(att,new Object[]{student,"name","key"});
SchemaObject studentlevel=
      s.createSchemaObject(att,new Object[]{student,"level","notnull"});
```

Two things should be noted about the above example. Firstly, the second argument of createSchemaObject is an Object array, the elements of which must match the types specified in the scheme of the construct in the MDR. For example, a runtime error would result if the text "student" (a string) replaced the student (a SchemaObject instance). Secondly, the Schema could have added to it constructs from different modelling languages. Mixing modelling languages in one schema is useful when translating between different modelling languages [3].

Transformations can be defined by being 'applied' to one schema, generating the next schema in the transformation sequence. Steps ① and ② both add an entity based on a query on the level attribute of student, and this is done by specifying the scheme of the new entity as an argument to the method of creating a transformation below. The first argument is the type of Construct being added, the second argument the scheme of SchemaObject, and the third argument is the query to derive instances of that object:

```
Schema s1a=s1.applyAddTransformation(ent,new Object[]{"undergrad"},
      "{x | ⟨x,ug⟩ in ⟪student,level,notnull⟫}",null);
Schema s1b=s1a.applyAddTransformation(ent,new Object[]{"postgrad"},
      "{x | ⟨x,pg⟩ in ⟪student,level,notnull⟫}",null);
```

The result of these two method calls is a schema held in s1b that contains entities undergrad and postgrad in addition to what is shown in s1 in Figure 1. To create ③, we need to obtain references to these entities so that they can appear in the scheme of the generalisation hierarchy level under student. This is done by finding the 'to' object of the transformations (the new object added by the transformation to the schema), before actually creating the transformation:

```
SchemaObject undergrad=Transformation.getTransformationToObject(s1,s1a);
SchemaObject postgrad=Transformation.getTransformationToObject(s1a,s1b);
Schema s1c=s1b.applyAddTransformation(gen,
    new Object[]{"level","total",student,undergrad,postgrad},null,null);
```

Finally, ④ is specified by creating a delete transformation, which does not need to specify the type of Construct being deleted, since that can be determined from the SchemaObject.

```
Schema s2froms1=s1c.applyDeleteTransformation(studentlevel,
    "{x,y | x in ⟨⟨undergrad⟩⟩,y=ug;x in ⟨⟨postgrad⟩⟩,y=pg}",
    "⟨_,y⟩ in ⟨⟨student,level⟩⟩ −⟩ y=ug;y=pg");
```

The result *is not* the schema s2, but a schema that *appears* to be the same as s2, but derived from the information in s1. To associate these two conceptually identical schemas together, a series of ident transformations are specified to associate pairs of identical objects, as follows:

```
Transformation.createIdentTransformations(s2,s2froms1,null,null);
```

Whilst s2 and s2froms1 appear identical, queries on s2 will be executed on its underlying database, whilst queries on s2froms1 will be executed on the database underlying s1. Hence we are able to control which database is used as the source for instances of a particular.

4 Conclusions

The alpha release of the API presented here has been fully tested, and work on a beta release is almost completed. Current development is focused on developing schema integration tools that work over the repository, and on integrating distributed querying processing software into the repository.

References

1. M. Boyd and N. Tong. The automed repositories and api. Technical report, 2001.
2. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — A schema transformation approach. In *Proceedings of ER99*, volume 1728 of *LNCS*, pages 96–113. Springer-Verlag, 1999.
3. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Advanced Information Systems Engineering, 11th International Conference CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer-Verlag, 1999.
4. P.J. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Advanced Information Systems Engineering, 14th International Conference CAiSE 2002*, LNCS. Springer-Verlag, 2002.
5. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.

Global Query Processing in the AutoMed Heterogeneous Database Environment

Edgar Jasper

School of Computer Science and Information Systems
Birkbeck College, University of London
edgar@dcs.bbk.ac.uk

1 Introduction

One of the requirements for implementing a heterogeneous database system is *global query processing*. The global query processor takes as input a query expressed on the constructs of a global schema of the heterogeneous database system. This query must be translated so that it is expressed over the constructs of the local database schemata. Appropriate local sub-queries are then sent to the local data sources for processing there. The global query processor must then take the results it receives from the local data sources and perform any necessary post-processing to obtain the result of the originally submitted global query.

This extended abstract discusses global query processing within the AutoMed Heterogeneous Database Environment (http://www.doc.ic.ac.uk/automed). AutoMed uses a common data model called the hypergraph data model (HDM) [9]. The HDM is a low level model where schemata consist of nodes, edges and constraints. The edges are directed edges that can link multiple nodes and other edges. The advantage of such a low level common data model is that it simplifies the mapping process between the constructs of different high level data models and the common data model. It is possible to translate a high level schema (eg. relational, ER, OO, XML) into an HDM schema [4,5]. Given knowledge of how a given data model's constructs are expressed in the HDM this schema translation can be done automatically.

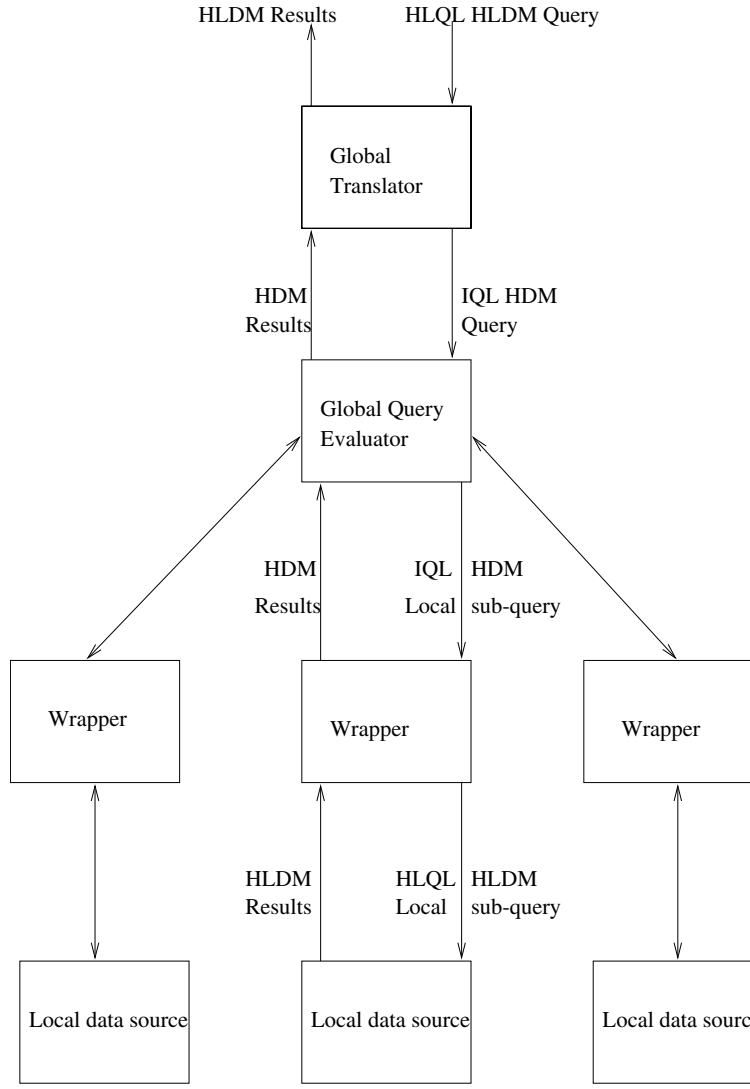
In the AutoMed framework, component HDM schemata are integrated into a global HDM schema by means of applying a list of primitive schema transformations to each component schema that transforms it into the global schema. The primitive transformations add, delete and rename nodes, edges and constraints. These transformations are automatically reversible (see [3]). This is significant for global query processing because it means that transformations used to transform component schemata into a global schema can be used to translate queries expressed on the global schema into queries expressed on the component schemata.

In AutoMed, queries on HDM schemata are written in a functional Intermediate Query Language (IQL) [8]. Thus a global query processor working in this framework will need to translate high level query languages (such as SQL, OQL) into and out of IQL.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 46–49, 2002.

2 Architecture of the AutoMed Global Query Processor

The following figure illustrates the overall architecture of the AutoMed Global Query Processor, which consists of one Global Translator, one Global Query Evaluator and several Wrappers.



The **Global Translator** receives a query written in a high level query language (HLQL) such as SQL employing a high level data model (HLDM) such as the relational model. This HLQL HLDM query is translated into an IQL HDM query in two steps. It is first translated into an IQL HLDM query. Then this IQL HLDM query is translated into an IQL HDM query.

This order is chosen because a data model cannot depend on the use of a particular query language but it is possible that a query language could be specific to a particular data model. Thus an HLQL HDM query might be nonsensical. It is expected that IQL is sufficiently general and extensible to work on any data model and cover the expressivity of any query language required in practice.

The IQL HDM query is then sent to the global query evaluator which returns an HDM result. This HDM result must be converted to a HLDM result before it is returned to the user.

A **Wrapper** takes an IQL HDM query, translates this into an appropriate IQL HLDM query and then into an HLQL HLDM query that is sent to the local data source. The HLDM results that the local data source returns are converted into HDM results before they are returned to the global evaluator.

The **Global Query Evaluator** takes as input an IQL query expressed using the constructs of the HDM representation of the global schema. The query needs to be rewritten to be expressed using the constructs of the component schemata. This is done using the transformation pathways between the component schemata and the global schema. These are retrieved from Automed's Schemas and Transformations Repository (see [1]).

Given the transformation pathways from the global schema to each component schema the input query can be rewritten. In particular, a view definition expressed in component schema constructs can be derived for each global schema construct for each component schema (see [2] for details). These view definitions are composed together and substituted for the global construct they represent in the input query. Thus the input query is rewritten to be expressed in terms of the component schema constructs.

Next, local sub-queries must be identified in the query. These are sub-queries that require only the constructs of one component schema. These sub-queries are then dispatched to the appropriate component translator and the results substituted into the query. The result is a query consisting of data and function applications which will be evaluated and then returned by the global evaluator.

Query optimisation is of course essential for global query processing. In our approach this can take place at several points. In the global translator after the query has been translated there may be redundancy that can be eliminated. There may be similar redundancy and general non-optimal structure in a wrapper after the query it receives has been translated. In the global evaluator there will be general optimisation work to be done — as in any heterogeneous database query processor. For this query optimisation functionality, we expect to draw on previous techniques for optimising functional query languages.

The above architecture has been partially implemented. Six months into the AutoMed project we have produced software that, given an IQL HDM query expressed on global schema constructs and given the transformation pathways, can rewrite this query to be expressed on component schema constructs. The software can then send local sub-queries (consisting only of individual component schema identifiers) to HDM data sources and substitute the results into the query. The resulting query can then be evaluated and the result returned.

3 Conclusions and Future Work

We have discussed the architecture of the AutoMed Global Query Processor and the implementation that has been completed so far. The next step is to implement the global translators and wrappers. This software will initially be language-specific. Ultimately, it is desirable that there be some means of defining arbitrary HLQLs and that the translation software is capable of using this information to provide automatic translation between IQL and such HLQLs.

Query optimisation will be a recurring theme throughout future work. In the short term, the priority will be finding and implementing rewrite rules for maximising the size of local sub-queries sent to the wrappers. In the longer term, further optimisation based on information about the local data sources themselves may well be possible [7].

The AutoMed approach to heterogeneous database systems is a novel one and so in turn our work on global query processing approaches the problem from a novel standpoint. In AutoMed, a low level data model - the hypergraph data model - is used as the common data model. Furthermore, the AutoMed schema integration approach is based on schema transformations rather than view definitions. While some advantages of using the AutoMed automatically reversible schema transformation approach over view definition approaches are documented (see [6]) it is an open question as to what, if any, will be the benefits arising from a global query processing point of view and our longer term aim will be to investigate this question.

References

1. M. Boyd and N. Tong. The AutoMed Repositories and API. Technical report, AutoMed Project, March 2002.
2. Edgar Jasper. Global query processing in the AutoMed heterogeneous database environment. Technical report, AutoMed Project, April 2002.
3. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proc. ER'99*, volume 1728 of *LNCS*, pages 96–113. Springer-Verlag, 1999.
4. 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-Verlag, 1999.
5. P.J. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE 2001*, volume 2068 of *LNCS*, pages 330–345. Springer-Verlag, 2001.
6. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. Technical report, AutoMed Project, February 2002.
7. George A. Mihaila, Louiqa Raschid, and Maria-Esther Vidal. Using quality of data metadata for source selection and ranking. In *WebDB (Informal Proceedings) 2000*, pages 93–98, 2000.
8. A. Poulovassilis. The AutoMed Intermediate Query Language. Technical report, AutoMed Project, June 2001.
9. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.

Tracing Data Lineage Using Automed Schema Transformation Pathways

Hao Fan

School of Computer Science and Information Systems, Birkbeck College
University of London, Malet Street, London WC1E 7HX
hao@dcs.bbk.ac.uk

1 Introduction

Data warehousing is being increasingly used for integrating distributed, heterogeneous data in order to enable sophisticated analysis of this data. **Automed** is a database transformation and integration system supporting both virtual and materialized integration of schemas expressed in a variety of modelling languages [9,5,6]. Automed has as its a common data model a low-level **hypergraph data model (HDM)**, and a set of primitive schema transformations operate on HDM schemas. An HDM schema consists of a set of nodes, edges and constraints. The primitive transformations add, delete, and rename a node, edge or constraint. The $addNode$ and $addEdge$ transformations include a query which defines the extent of the new schema construct in terms of the extents of the existing schema constructs (so adding the construct does not change the information content of the schema). Similarly, the $delNode$ and $delEdge$ transformations include a query which shows how the extent of the deleted construct can be reconstructed from the remaining schema constructs.

For the purposes of tracing data lineage, we assume here that both the source schemas and the integrated schema are expressed in the HDM data model since, as discussed in [6], higher-level schemas and the transformations between can be automatically translated into an equivalent HDM representation. We use a functional *intermediate query language* (IQL) [8] for defining the queries accompanying add and del transformations. For example, this IQL query returns the maximum daily sales total for each store in a relation *StoreSales* (*store_id*, *daily_total*, *date*), where "gc" is a "group-and-compute" operator:

$$gc\ max\ [(s,t)\,|\,(s,t,d) \leftarrow StoreSales]$$

2 Data Lineage Tracing Using Transformation Pathways

The *data lineage problem* concerns tracing how items of warehouse data have been derived from the data sources. Previous works relating to data lineage tracing have defined the notions of *fine-grained* data lineage [11], *derivation pool* [3], and the difference of *why*- and *where- provenance* [2]. We use all of these notions in our approach. We give the definitions of *affect-pool* and *origin-pool* in [4]. What we regard as affect-provenance includes all of the source data that had some influence on the result data.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 50–53, 2002.

Origin-provenance is simpler because here we are only interested in the specific data in the source databases from which the resulting data is extracted.

As in [3], we use *derivation tracing queries* to evaluate the lineage of a tuple t. That is, we apply a query to the source data repository D and the obtained result is the derivation of t in D. We call such a query the *tracing query for t on D*. Let $V = q(D)$ be the bag that results from applying an IQL query q to a source data repository D, consisting of one or more bags. Then, for any tuple $t \in V$, the tracing query $TQ_D^{AP}(t)$ gives the affect-pool of t in D, and the tracing query $TQ_D^{OP}(t)$ give the origin-pool of t in D. For simple IQL queries, TQ_D^{AP} and TQ_D^{OP} are defined as follows, where "++" is bag union and "−" is bag monus:

$$
\begin{aligned}
q &= D_1 + + \ldots + + D_r \quad D = \langle D_1, \ldots, D_r \rangle : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \langle [x | x \leftarrow D_1; x = t], \ldots, [x | x \leftarrow D_r; x = t] \rangle \\
q &= D_1 - D_2 \quad D = \langle D_1, D_2 \rangle : \\
TQ_D^{AP}(t) &= \langle [x | x \leftarrow D_1; x = t], D_2 \rangle \\
TQ_D^{OP}(t) &= \langle [x | x \leftarrow D_1; x = t], [x | x \leftarrow D_2; x = t] \rangle \\
q &= group\ D : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \langle [x | x \leftarrow D; first\ x = first\ t] \rangle \\
q &= sort\ D\ /\ sortDistinct\ D : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \langle [x | x \leftarrow D; x = t] \rangle \\
q &= max\ D\ /\ min\ D : \\
TQ_D^{AP}(t) &= \langle D \rangle \\
TQ_D^{OP}(t) &= \langle [x | x \leftarrow D; x = t] \rangle \\
q &= count\ D\ /\ sum\ D\ /\ avg\ D : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \langle D \rangle \\
q &= gc\ max\ D\ /\ gc\ min\ D : \\
TQ_D^{AP}(t) &= \langle [x | x \leftarrow D; first\ x = first\ t] \rangle \\
TQ_D^{OP}(t) &= \langle [x | x \leftarrow D; x = t] \rangle \\
q &= gc\ count\ D\ /\ gc\ sum\ D\ /\ gc\ avg\ D : \\
TQ_D^{AP}(t) &= TQ_D^{PP}(t) = \langle [x | x \leftarrow D; first\ x = first\ t] \rangle \\
q &= [x | x \leftarrow D_1; member\ D_2\ x] \quad D = \langle D_1, D_2 \rangle : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \langle [x | x \leftarrow D_1; x = t], [x | x \leftarrow D_2; x = t] \rangle \\
q &= [x | x \leftarrow D_1; not\ (member\ D_2\ x)] \quad (D = \langle D_1, D_2 \rangle) : \\
TQ_D^{AP}(t) &= \langle [x | x \leftarrow D_1; x = t], D_2 \rangle \\
TQ_D^{OP}(t) &= \langle [x | x \leftarrow D_1; x = t] \rangle \\
q &= [p | p_1 \leftarrow D_1; \ldots; p_r \leftarrow D_r; c_1; \ldots; c_k] \quad D = \langle D_1, \ldots, D_r \rangle : \\
TQ_D^{AP}(t) &= TQ_D^{OP}(t) = \\
&\quad \langle [p_1 | p_1 \leftarrow D_1; p_1 = t_1; \ldots; p_r \leftarrow D_r; p_r = t_r; c_1; \ldots; c_k], \ldots, \\
&\quad [p_r | p_1 \leftarrow D_1; p_1 = t_1; \ldots; p_r \leftarrow D_r; p_r = t_r; c_1; \ldots; c_k] \rangle
\end{aligned}
$$

In the last query form, each pattern p_i is a sub-pattern of p and all tuples $t \in V$ match p; for any $t \in V$, t_i is the tuple derived by projecting the components of p_i from t. For more complex queries, the above formulae can be recursively applied to the syntactic structure of an IQL query. An alternative (which we are currently exploring) is to decompose a transformation step containing a complex IQL query into a sequence of transformation steps each containing a simple IQL query.

Other ongoing work within the Automed project is investigating simplification techniques for transformation pathways [10]. As a result of such simplification, we assume here that each construct O appearing in an integrated schema has been created from the source schemas in one of three ways:

(i) By an *add*(O, q) transformation, in which case the lineage of data in O is located in the constructs that appear in q.
(ii) By a *rename*(P, O) transformation, in which case the lineage of data in O is located in the source construct P.
(iii) O exists in a source schema and remains in the integrated schema, in which case the lineage of data in O is located in the source construct O.

It is simple to trace data lineage in cases (ii) and (iii): all of data in O is extracted from a source database, and the affect-pool is equal to the origin-pool. For (i), we use the formulae for the tracing queries given earlier to obtain the lineage of the data:

We first use two procedures *affectPoolOfTuple*(t, O) and *originPoolOfTuple*(t, O) to trace the affect pool and origin pool of a tuple, where t is the tracing tuple in the extent of some construct O of the integrated schema (see [4] for these procedures). The result of these procedures, D^*, is a bag which contains t's derivation in the source databases.

Next, two procedures *affectPoolOfSet*(T, O) and *originPoolOfSet*(T, O) compute the derivations of a tuple set T contained in the extent of a construct O (see [4]). These procedures call *affectPoolOfTuple*(t, O) and *originPoolOfTuple*(t, O) above to trace the derivations of each tuple $t \in T$ and incrementally add each time the result to D^*.

Finally, we give below our recursive derivation tracing algorithm for tracing data lineage using entire transformation pathways, *traceAffectPool*(*TL*, *OL*) (the *traceOriginPool*(*TL*, *OL*) algorithm is similar, obtained by replacing "affect" by "origin" everywhere). *TL* = $T_1, \ldots, T_n$ is a list of tuple sets such that each T_i is contained in the extension of some integrated schema construct O_i. *OL* is the list of integrated schema constructs $O_1, \ldots, O_n$. Each schema construct has an attribute *relateTP* that refers to the transformation step that created this construct. Each transformation step has an attribute *transfType* which is *"add"*, *"del"* or *"rename"*, and an attribute *sourceConstruct*; for an "add" transformation, *sourceConstruct* returns the set of schema constructs appearing in the query parameter. If a construct O_i containing tuple set T_i is created by a *rename* transformation or remains from a source schema then the computed data are directly extracted from the source data. If O_i is created by an *add*(O_i, q) transformation, the constructs in query q may have been created by the earlier part of the transformation pathway, and the computed data needs to be extracted from these constructs. Therefore, we call procedure *traceAffectPool* recursively while the *relateTP* of the construct is "*add*':

```
proc traceAffectPool(TL, OL)
begin
    D* ← ∅;
    for i := 1 to n do {
      temp ← affectPoolofSet(T_i, O_i);
      if T_i.relateTP.transfType = "add"
        temp ← traceAffectPool(temp, T_i.relateTP.sourceConstruct);
      D* ← D* ++ [x | x ← temp; not (member D* x)];
    }
    return(D*);
end
```

3 Conclusions and Future Work

We have described how the Automed transformation pathways can be used to trace the derivation of data in an integrated database in a step-wise fashion. More details are given in [4]. We are currently implementing our algorithms over the Automed repository [1]. For future work we plan to extend our algorithms to the more expressive transformation language described in [7], and to explore the relationship between lineage tracing and incremental view maintenance using Automed's schema transformation pathways, in order to determine if an integrated approach can be adopted for both.

References

1. M. Boyd and N. Tong. The Automed repositories and API. Technical report, Automed Project, 2001.
2. P. Buneman, S. Khanna, and W.C. Tan. Why and Where: A characterization of data provenance. In *Proc. ICDT 2001*, pages 316–33, 2001.
3. Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2):179–227, 2000.
4. H. Fan and A. Poulovassilis. Tracing Data Lineage Using Automed Schema Transformation Pathways. Technical report, Automed Project, 2002. BBKCS-02-07.
5. P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications - A schema transformation approach. In *Proc. ER'99*, pages 96–133, 1999.
6. P. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, pages 333–348, 1999.
7. A. Poulovassilis. An enhanced transformation language for the HDM. Technical report, Automed Project, 2001.
8. A. Poulovassilis. The Automed Intermediate Query Language. Technical report, Automed Project, 2001.
9. A. Poulovassilis and P. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
10. N. Tong. Database schema transformation optimisation techniques for the Automed system. Technical report, Automed Project, 2002.
11. A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc ICDE'97*, pages 91–102, 1997.

A System for Managing Alternate Models in Model-Based Mediation

Amarnath Gupta, Bertram Ludäscher, Maryann E. Martone, Xufei Qian,
Edward Ross, Joshua Tran, and Ilya Zaslavsky

San Diego Supercomputer Center, University of California San Diego
{gupta,ludaesch,xqian,eross,trantj,zaslavsk}@sdsc.edu,
mmartone@ncmir.ucsd.edu

1 Introduction

In [1,3], we have described the problem of *model-based mediation* (MBM) as an extension of the global-as-view paradigm of information integration. The need for this extension arises in many application domains where the information sources to be integrated not only differ in their export formats, data models, and query capabilities, but have widely different schema with very little overlap in attributes. In scientific applications, the information sources come from different subdisciplines, and despite their poorly overlapping schema, can be integrated because they capture different aspects of the same scientific objects or phenomena, and can be conceptually integrated due to scientific reasons. In the MBM paradigm, a "mediation engineer" consults with domain experts to explicitly model the "glue knowledge" using a set of facts and rules at the mediator. Integrated views are defined in MBM on top of the exported schemas from the information sources together with the glue knowledge source that ties them together. We have successfully applied the MBM technique to develop the KIND mediator or Neuroscience information sources [1]–[4]. To accomplish this, sources in the MBM framework export their conceptual models (CMs), consisting of the logical schema, domain constraints, and ***object context*s**, i.e., formulas that relate their conceptual schema with the global domain knowledge maintained at the mediator. Thus model-based mediation has a hybrid approach to information integration - on the one hand at the mediator integrated views are defined over source CMs and the Knowledge Map using a global-as-view approach; on the other hand, object-contexts of the source are defined as local-as-view.

In the demonstration, we present the KIND2 system, a further extension to the MBM paradigm - here the mediator may have *alternate* sources of glue knowledge, often developed by consulting different domain experts. We presume there are no contradictions between recorded or derived facts from different sources, and show how integrated views are defined and queries are evaluated in the presence of alternate knowledge sources in an MBM setting.

2 The Demonstration System

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 54–57, 2002.

2.1 Data and Knowledge Sources

The KIND2 system mediates across a number of neuroscience data sources provided to us by different partner institutions. The data consists of relational sources containing image and volume-based measurements, XML sources containing protein information and time-series sources containing physiological recordings from neural responses for specific stimulations. The mediator uses two forms of knowledge sources to integrate this information - a graph structured ontology (stored and displayed to the user as a labeled graph) and a spatial atlas of the brain:

- The ontology is constructed from the Unified Medical Language System [1] ontology from the National Library of Medicine and the Gene Ontology from the Gene Ontology Consortium [2]. The two ontologies together store about 2 million concept names (nodes) and 10 million relationships (edges) represented as relational tables in an Oracle8 database. They are accessed through FLORA, an F-logic engine built on top of XSB Prolog [5]. The F-logic engine allows the definition of (often *highly* recursive) views. In our setting, it also pushes certain operations (e.g., SPJ and some hierarchical queries) to the Oracle system below, and performs deductive computations on the results.
- The spatial atlas consists of a number of layers, where a layer is an orthographic cross-section of the brain, on which the observed structure on the section are outlined and labeled. Obviously, since most structures in the brain are three-dimensional, they appear on multiple layers. Our atlas source is created from commercially available brain atlases, by converting line drawings to polylines and polygons in Oracle Spatial Data Cartridge. Using this system, one can perform two-dimensional topological and metric queries on atlas objects. We have developed additional algorithms to simulate some three-dimensional operations as stored PL-SQL procedures on top of the system's native two-dimensional query capabilities. The demonstration system will show how object contexts are defined from both of these knowledge sources, by illustrating how a user can navigate the knowledge sources themselves "looking for" data sources that are reachable from subgraphs of the first source, or subregions of the second source.

2.2 The KIND2 Mediator

The primary mediator module in the KIND2 system is built using F-Logic. Data sources register with the mediator by wrapping their native schema into F-Logic. The query capabilities of data sources are modeled by source-specific special predicates. For example, the volume analysis data source for morphometry supports an operation for spine density distribution which, given a user-specified interval along the length of a dendrite, returns an XML document containing frequency histogram of spine density in that interval. The mediator views this operation as a built-in predicate with binding patterns for the input and output parameters.

[1] `http://www.nlm.nih.gov/research/umls/index.html`.
[2] `http://www.geneontology.org`.

Integrated views in KIND2 are defined as a set of F-logic rules. We illustrate the use of alternate knowledge sources by defining intensional predicates both in terms of logical and spatial operations. For example, the predicate **contains_tc**(object1, object2) can be defined as a transitive closure on the predicate **contains**(object1, object2) maintained by the ontology. Given that the UMLS ontology knows contains(thalamus, 'fasiculi thalami') and contains('fasiculi thalami', 'anterior peduncle') are true, the system infers **contains_tc**(thalamus, 'anterior peduncle'). Alternately, using the spatial atlas (Fig. 1), one can use a spatial operator named **inside**: $polygon \rightarrow$ **setof**$(polygon)$. The atlas will look for the polygon labeled as thalamus in each slice, and geometrically locate all the other labeled polygons inside it, thus finding the 'anterior peduncle'. Using either method, one may define an integrated view on the data sources. The view may be a selection on all proteins *P* involved in some activity *A* of some neurons *N* and that can be localized in region *R* of the brain. Here, the activity *A* is defined in terms of the time-series recording of neurons, the protein properties are retrieved from a protein database[3], and the protein localization information is available both from whole-brain experiments and neuron-level experiments. The role of the ontology and the atlas is to provide two missing pieces of information to construct the view: (a) the subregional architecture of the brain, and (b) situating specific neurons in specific brain regions. Given a query against this view the mediator needs to rewrite it using the predicate **contains_tc**, the operation **inside** or both. The decision is based on several factors including:

- whether the two knowledge models have equal granularity of information for the query region
- whether there are any data sources that refer to only one model for the query region
- whether other predicates in the query necessitate visiting one source over the other
- the estimated cost of the two operations for the query region
- Often a good solution is to partially execute the query using one source, obtain subregion names, and pass them to the other source to complete the query.

In the demonstration system we will show the system's query evaluation functionality with a *plan-viewer tool*. The plan starts from the user's query and first selects all matching views that support the binding pattern of the query. Once the views are identified, the user of the demo system may choose any view to unfold in order to continue query processing. We will demonstrate the case of the alternate view definitions described earlier. The tool provides a simple graphical interface to demonstrate a trace of the query planning process. Given an arbitrary query against a given view, the demo user may choose to see all generated plans and the plan chosen by the mediator. The demo user may also select a plan from the initial set of plans, and trace when it gets eliminated. In this case, the system will show how the chosen plan is selected over other plans or is pruned by a competing plan. This will be shown using a trace of the query processing rules that were applied to prune one plan over another.

[3] For example, consider the web-accessible database for calcium-binding proteins located at `http://structbio.vanderbilt.edu/cabp_database/cabp.html`.

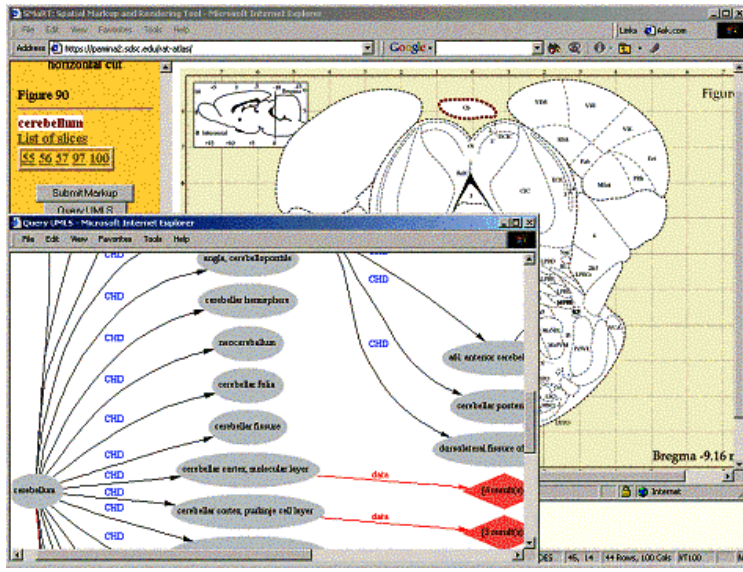


Figure 1. A query that use the UMLS ontology (in front) and the spatial atlas (behind)

References

1. A. Gupta, B. Ludäscher, M. Martone, "Knowledge-based Integration of Neuroscience Data Sources", 12th *Int. Conf. Scientific and Statistical Database Management Systems*, Berlin, 39-52, 2000.
2. B. Ludäscher, A. Gupta, M. Martone, "A Mediator System for Model-based Information Integration", *Int. Conf. VLDB*, Cairo, 639-42, 2000.
3. B. Ludäscher, A. Gupta, M.E. Martone, "Model-Based Mediation with Domain Maps", *17th Intl. Conference on Data Engineering* (ICDE), Heidelberg, Germany, IEEE Computer Society, 81-90, 2001.
4. Xufei Qian, Bertram Ludäscher, Maryann E. Martone, Amarnath Gupta: "Navigating Virtual Information Sources with Know-ME", EDBT 2002: 739-741
5. G. Yang, M. Kifer, "FLORA: Implementing an Efficient DOOD System Using a Tabling Logic Engine", *Computational Logic*, 1078-1093, 2000.

CREAM: A Mediator Based Environment for Modeling and Accessing Distributed Information on the Web*

Sudha Ram[1], Jinsoo Park[2], and Yousub Hwang[1]

[1] Department of Management Information Systems
University of Arizona, Tucson, AZ 85721
{ram,yhwang}@bpa.arizona.edu
[2] Department of Information Decision Sciences
University of Minnesota, Minneapolis, MN 55455
park@umn.edu

1 Introduction

The proliferation of the Internet has led to increasing semantic heterogeneity on the World Wide Web. Networking technology provides physical connectivity, but it does not ensure meaningful data exchange. Therefore, semantic interoperability among heterogeneous information sources continues to pose enormous challenges to the database and other communities. Consequently, cooperation of metadata, ontologies, and mediators is becoming even more important in achieving semantic interoperability; mediators collect information from heterogeneous and autonomous sources and resolve semantic conflicts by referring to metadata and ontologies. Based on the mediator approach, we have implemented a **C**onflict Resolution **E**nvironment for **A**utonomous **M**ediation (CREAM) system. The system provides various user groups (e.g., end-users and information integrators) with an integrated and collaborative facility for achieving semantic interoperability among the participating heterogeneous information sources. The detailed description of a theoretical framework for semantic conflict detection and resolution methodology can be found in [1]. Our major contribution is that our system automatically *understands* and *resolves* a large number of semantic and syntactic conflicts among heterogeneous databases. We have tried to minimize the burden on the end users and database administrators while trying to automate the heterogeneous conflict resolution process.

2 Conflict Resolution Environment for Autonomous Mediation

CREAM can be used to access underlying heterogeneous structured databases as well as Web published semi-structured information. As shown in Fig. 1, the CREAM architecture consists of four different layers, each of which contains several software modules: the metadata layer, the semantic mediation layer, the data exchange layer, and the data access layer.

* This research is supported in part by the HyDiS and RESAC research grants from NASA.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 58–61, 2002.

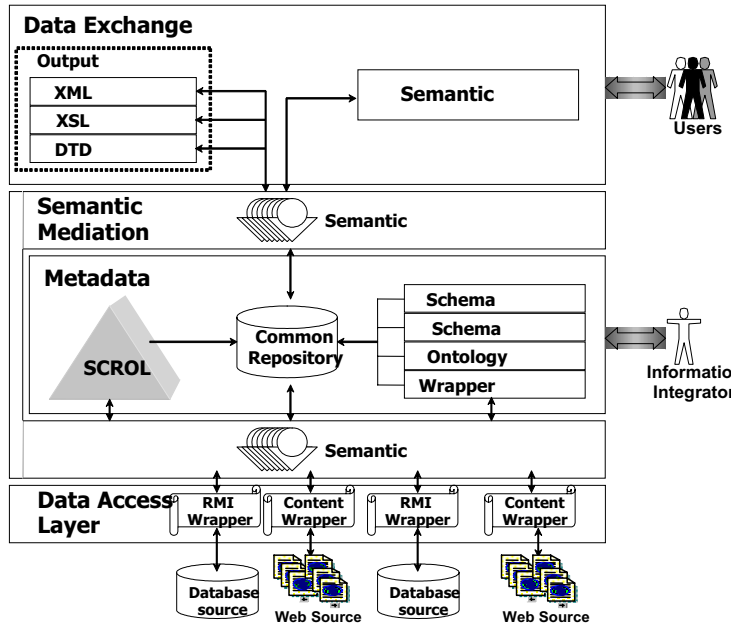


Figure 1. Overview of CREAM Architecture

2.1 Metadata Layer

The metadata layer provides the ability to capture semantic metadata into a common repository and construct wrappers for Web sources. Semantic metadata that needs to be captured includes: (i) federated and local schemata for heterogeneous databases, (ii) semantic mapping knowledge to capture schema and ontology mappings, and (iii) semantic extraction rules to construct wrappers for Web sources. These tasks are performed by a human information integrator aided by a set of metadata acquisition and representation tools, such as conceptual schema designer, schema mapper, ontology mapper, and wrapper generator.

The *conceptual schema designer* allows the information integrator to construct a federated schema using a point-and-click interface. It is also used to define and/or automatically generate local schemata from underlying databases. These conceptual schemata provide a way to capture the metadata associated with information sources.

The *schema mapper* allows the integrator to make logical links between semantically similar constructs in heterogeneous local schemata to a federated schema. The integrator simply clicks on a component in a federated schema and points to the corresponding local schema component. In addition, users can browse and update the schema mappings.

The *ontology mapper* allows users to establish mappings between a concept in SCROL (**S**emantic **C**onflict **R**esolution **O**nto**l**ogy) to a conceptually matching schema component. These ontology mappings help capture the contextual knowledge of multiple information sources. Since SCROL is used to capture various semantic conflicts at both data and schema levels, they also help automate the conflict detection and resolution process [2]. This knowledge is used by a query generator (a type of semantic mediator) to generate proper local queries from a single global query requested by a user.

The *wrapper generator* assists the information integrator in easily constructing wrappers for Web sources. In order to extract content from Web sources, a wrapper needs two important types of rules: (i) URL generation rules for physically identifying Web sources, and (ii) semantic content extraction rules for describing the syntactic structure of the Web source. Using a URL generator interface, the information integrator captures URL generation rules for a given Web source. In addition, the information integrator needs to provide parameter values for the dynamic portion of a URL string. The wrapper generator encapsulates the parameter values into an object that is used during the URL generation process. Another important piece of knowledge required to access Web data is the semantic extraction rules for specific Web pages. The wrapper can identify and extract semantic content from the Web source by analyzing the syntactic structure and content of the Web sources. Because the semantic extraction rule generator provides an interactive mechanism for analyzing the syntactical structure and content of the web pages with a few mouse clicks, it is not necessary for the information integrator to understand and use a high-level declarative language to generate semantic extraction rules.

2.2 Semantic Mediation Layer

The semantic mediation layer is intended to resolve various semantic conflicts and coordinate the information gathering process from heterogeneous and autonomous information sources. We define seven distinct types of semantic mediators based on different functionalities and tasks. They are: query generator, coordinator, conflict detector, conflict resolver, selector, message generator, and data collector.

2.3 Data Access Layer

This layer is responsible for gathering query results from different types of information sources. This layer provides an interface between the semantic mediation layer and the underlying heterogeneous information sources. This layer consists of wrappers that are responsible for accessing the corresponding information sources. These wrappers are classified into two types based on the nature of the information sources, viz. structured and semi-structured. The first type of wrapper is responsible for accessing information sources in databases. They simply wrap RMI (Remote Method Invocation) calls, while the other type of wrapper is intended to extract semantic content from Web sources. Wrappers of this kind are generated using our wrapper generator.

2.4 Data Exchange Layer

The data exchange layer provides tools for users to initiate information requests and export the semantically integrated information into XML. It consists of two software components. The *semantic filter* is a query interface that allows users to create global queries. Since the semantic filter provides a homogeneous view of heterogeneous information sources, which is independent of the structure and the location of the actual data sources, the user can transparently access data from difference sources without the prior knowledge of their information content. Users can also export the retrieved results into an XML-based document with complete semantic metadata information using the *output generator* that creates XML tags and inserts corresponding values from the data passed by semantic mediators. It also coordinates the DTD and XSL generation. Since XML is an emerging standard data format for B2B transactions, our architecture leverages the power of XML to support interoperability between loosely coupled virtual partners in an electronic marketplace [3].

3 Demonstration

Our CREAM prototype has been developed using Java 2 SDK and Oracle 8i. Our examples for the demonstration will illustrate how to facilitate semantic interoperability among heterogeneous and autonomous information sources. Specifically, we will demonstrate the following aspects of the CREAM system:

- The ability to identify relevant information sources and provide a unified view of data.
- The incorporation of a semantic data model and ontology to provide an automatic way of identifying and resolving semantic conflicts at both data and schema levels.
- The ability to semi-automatically generate wrappers for semi-structured information sources.

References

1. Park, J.: Facilitating Interoperability among Heterogeneous Geographic Database Systems: A Theoretical Framework, A Prototype System, and Evaluation. Doctoral Thesis, Department of Management Information Systems, The University of Arizona, Tucson, Arizona, 1999.
2. Ram, S., Park, J., Kim, K., Hwang, Y.: A Comprehensive Framework for Classifying Data- and Schema-Level Semantic Conflicts in Geographic and Non-Geographic Databases. In: Sarkar, S., Narasimhan, S. (eds.): Proceedings the Ninth Workshop on Information Technologies and Systems. Charlotte, North Carolina, December 11-12, 1999. 185-190.
3. Ram, S., Hwang, Y., Khatri, V.: Infomediation for E-business enabled Supply Chains: A Semantics Based Approach. In: Proceedings of DS-9-Semantic Issues in Ecommerce Systems. Hong Kong, April 23-28, 2001. 217-231.

An Integrated Approach to Handling Collaborative Diagram Databases on the WWW

Xiao Mang Shou and Siobhán North

Department of Computer Science, University of Sheffield, Sheffield S1 4DP
X.M.Shou@shef.ac.uk, S.North@dcs.shef.ac.uk

1 Introduction

The capability of the Web in handling text and multimedia information is well known but diagram handling on the Web remains an insufficiently explored field. The motivation for this research was the Cytokine WebFacts project, a database of information about Cytokines to be created and updated by cytokine experts. The data to be stored on the Cytokine WebFacts included text, images and cytokine signal pathway diagrams. The pathway diagram database, the most challenging part, was formed of diagrams submitted by users. Having established the diagram database, functions were developed to allow users to index and search diagrams, embed links within them, publish alternative views of diagrams, modify other people's work and so on. To enable these functions, a new set of tools were developed and are described here. Very little of this work is domain specific and thus could be of use in other application domains.

2 Editing, Saving, and Searching Diagrams on the Web

Although there are no widely used tools for diagram handling on the web, there are some conventional approaches. The most common method is to treat diagrams as images; this is simple but the diagrams cannot be edited and it takes up too much space. Another common approach is to let users create and edit diagrams using any package and have a webmaster subsequently compile and display on the Web. This approach is only practical in application domains where the quantity of the diagrams is limited, the diagrams are simple and modifications to the diagrams are not required on a frequent basis, effectively a diagram store rather than a true database. Though limited, there are some diagram applications available [2,4,5]. However none of them have any concept of a collaborative diagram database updated by its users. The development of WIDE (the WWW Interactive Diagram Editor) [6] came from the requirement to have an interactive diagram editor on the Web for users to enter, update and save diagrams directly on the Web server. WIDE was implemented using Java applets and Java servlets. It is a GUI drawing panel providing basic, intuitive editing functions. The most innovative part of WIDE was the Diagram Transfer mechanism. Two functions, Save and Load, are provided. Both functions are implemented in a way that ensures users need to know nothing about the data format of the diagrams so users play an active role in building and maintaining the diagram database. In addition to conventional drawing elements WIDE diagrams can contain URL links to other Web pages or other diagrams. These

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 62–65, 2002.

links, like everything else in the diagram, can be added, removed or updated directly by users. Given the relatively narrow domain that links are likely to belong to our approach was to provide a picklist of possible internal (to the database) URL links automatically so that users need not be aware of the internal server structure [7]. However users are still allowed to enter any URL address freely should they need to.

The diagram search engine was designed to search local diagram files only. It is specific to WIDE diagrams in that it searches only text within diagrams and not for particular shapes or line types as objects in cytokine signal pathway diagrams have no semantic meaning and are left entirely up to the user. It does, however, used meta data associated with every diagram. The meta data includes the title, keywords, a description, the contributor's name and email address. The meta data has to be searchable as well as text in diagram body. They were assigned different weights according to their importance in identifying the diagrams and can be regarded as the indices for cytokine signal pathway diagrams. The results of the search are displayed as strings concatenated from the diagram title, keywords, description and contributor's name to give a clear explanation.

3 Version Control for the Web Diagram Databases

Updating the database in terms of adding extra diagrams and deleting old ones is simple but the modification of existing diagrams using the editor can introduce further problems because the new version of a diagram may not superseded the old one, both may continue as part of the database so we also need some sort of version control mechanism which is specific to these sort of diagrams. Versioning usually means the way users check out a copy for parallel editing, differencing finds differences and conflicts among versions and merging joins multiple versions together and produces a new version representing the union of the actions taken in the previous versions. WIDE adopts an optimistic check-out mechanism so that other users are not blocked from accessing the diagram database. Users can update an existing diagram and then submit it to be added to the database.

Newly submitted diagrams have to be checked by the domain experts to make sure they are reasonable and correct before they are transferred to the final database. This process involves finding different versions of a diagram, identifying conflicts within the different versions, and most importantly, merging those versions into one final state or a few alternative versions differenced and merged. Differencing and merging text information have been covered in CSCWriting applications, but there is no equivalent for diagrams and the differencing and merging functions for WIDE diagram revisions had to be developed from scratch. Handling diagrams is more complicated than text documents because, as well as changes to text within diagrams, the system must recognise changes in shape, colour, size, spatial position etc. too. To avoid unreasonable diagrams in the database and maintain consistency, newly submitted or modified diagrams need to be saved in a provisional directory to be reviewed by the domain experts before being transferred to the permanent database. The domain experts have full permission to manage the provisional directory. To differentiate between diagrams at different stages, the diagrams in the final database directory will be called *diagrams* and the diagrams in the

provisional directory will be called *revisions* (if they are based on existing diagrams) or *sketches* (for new diagrams without a parent). To distinguish these different types of files, they have different name conventions. The diagram database adopted an integrated naming convention by embedding the root diagram file name, user email address and a serial number into provisional file names as well as using different extensions for easy identification. All the revision names are automatically generated, so there will never be a problem of a revision saved with an improper filename.

To review the *raw revisions* (new versions of existing diagrams submitted by users) the domain experts need to identify which parts of the diagram have been updated. If there are other revisions of the same original diagram, the domain experts will have to compare all of them to locate compatible modifications and possible conflicts. In order to perform these functions easily and quickly, the editor had to include information about changes. The most direct way to achieve this is to set flags during the editing operations on modified objects. In order to merge diagrams successfully we must know, not only what changes have been made, but also how important each change is. The editing operations need to be stored with the raw revisions. To incorporate as many changes as possible, the change storing mechanism adopts the "just enough" method to include all the editing operations but filter out redundant ones (such as multiple MOVE functions where only the last one counts, or functions before a DELETE). Instead of assigning serial numbers to different change operations, binary numbers can be assigned and the mixture of the change operations can be calculated by bitmask. WIDE applies the activity identification scheme which uses the index of the element in the document to identify the same object across different revisions; the DELETE operation had to be modified to simply set a tag rather than remove the object from the vector.

The differencing tool of WIDE was designed to be used to compare a raw revision and the original root diagram although it would be possible to extend it. To present the differences the tool displays both diagrams with the differences between the original diagram and the revision highlighted in the revision. This is achieved by having a differencing state where all unchanged objects are drawn in grey while modified objects are highlighted in specific colours. Though divergent revisions can be saved as alternative diagrams in the Cytokine WebFacts diagram database, it must also be possible to merge two *raw revisions* as a new diagram in the database in order to incorporate modifications scattered in different revisions.

The rule of thumb for default merging of diagrams is to adopt the changes with highest priority across revisions. In cases where the weights of the change operations in both revisions are exactly the same the conflict can be resolved by adopting an asymmetric merge method similar to that of COOP [1] which is, to prefer the change in the main revision to the change in the added revision (the main and added revisions are identified by the user). Though the asymmetric merge may not be the ideal solution in some situations, it helps to resolve conflicts that usually require human involvement. The merged revision is displayed in the modification viewing state (differencing state) where unchanged objects are in grey and modifications are highlighted in different colours. An important issue in the merging interface is that, the result shown is the temporary result of "virtual merge". The merging interface just displays the possible result of the merging between two raw revisions as a reference for the domain experts. Only when

the domain experts are happy with the virtual result can they save it into the diagram database.

4 Discussion and Future Work

The existing system forms a relatively complete platform to maintain an interactive diagram database but, as always there are areas where it could be extended. For instance, importing and exporting diagrams with other formats needs to be developed.

The tools and techniques described here are not specific to cytokine signal pathway diagrams, they are fully extendable to other applications. Although cytokine pathway diagrams are not hierarchical, the URL links within diagrams mean they could easily be used in an hierarchical environment.

References

1. Ulf Asklund; "Identifying Conflicts During Structural Merge"; *Proceedings of Nordic Workshop on Programming Environment Research*; Lund, Sweden; June 1994; pp. 231-242
2. N.S. Barghouti, J.M. Mocenigo, and W. Lee; "Grappa: A GRAPh PAckage in Java"; *Graph Drawing, Lecture Notes in Computer Science (1353)*; Springer-Verlag; 1998; pp. 336-343
3. R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, et al.; "Basic Support for Cooperative Work on the World Wide Web"; *International Journal of Human-Computer Studies: Special Issues on Novel Applications of the WWW;* Academic Press, Cambridge; Spring 1997
4. S. Bridgeman, A. Garg, and R. Tamassia; "A Graph Drawing And Translation Service on the WWW"; Graph Drawing, Lecture Notes in Computer Science (1190); Springer-Verlag; 1997; pp. 45-52
5. P. Ertl and O. Jacob; "WWW-based Chemical Information System"; THEOCHEM 1997, 419; pp. 113-120
6. Xiao Mang Shou and Siobhán North; "WIDE: A WWW Interactive Diagram Editor"; *Proceedings of the First International Conference on the Practical Application of Java (PA-Java'99);* The Practical Application Company Ltd; April 1999; pp. 91-103
7. Xiao Mang Shou, PhD Thesis, University of Sheffield 2001; pp. 65-83

Handling Conceptual Multidimensional Models Using XML through DTDs*

Enrique Medina, Sergio Luján-Mora, and Juan Trujillo

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, Spain
{emedina,slujan,jtrujillo}@dlsi.ua.es

Abstract In the last years, several approaches have been proposed to easily capture main multidimensional (MD) properties at the conceptual level. In this paper, we present how to handle MD models by using the eXtensible Markup Language (XML) through a Document Type Definition (DTD) and the eXtensible Stylesheet Language Transformations (XSLT). To accomplish this objective, we start by providing a DTD which allows us to directly generate valid XML documents that represents MD properties at the conceptual level. Afterwards, we provide XSLT stylesheets to automatically generate HTML pages that correspond to different presentations of the same MD model.

1 Introduction

Data warehouses, OLAP applications and MD databases, based on the MD modeling, provide companies with much historical information for the decision making process. In recent years, there have been some proposals to accomplish the conceptual MD modeling of these systems [1,2,3]. Due to space constraints, we refer the reader to [4] for a detailed comparison and discussion.

In this paper, we present a way to handle the representation, manipulation and presentation of MD models accomplished with the object-oriented (OO) approach presented in [5] by using a standard format. To accomplish this goal, we define a DTD to validate XML documents that store all the MD properties. Furthermore, we combine XSLT stylesheets and XML documents in a transformation process to obtain HTML pages.

The remainder of this paper is structured as follows: Section 2 describes the basis of the OO conceptual MD modeling approach that we adopt in this paper. Section 3 presents how to represent MD models with the XML and the XSLT. Finally, in Section 4 we present our conclusions and future works.

2 Object-Oriented Multidimensional Modeling

In this section, we summarize how an OO MD model, based on the Unified Modeling Language (UML), can represent main structural MD properties at the conceptual level.

* This paper has been partially supported by the Spanish Ministery of Science and Technology, Project Number TIC2001-3530-C02-02.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 66–69, 2002.

In this approach, the main structural properties are specified by means of a UML class diagram in which the information is clearly separated into facts (*fact classes*) and dimensions (*dimension classes*). Then, fact classes are specified as composite classes in shared aggregation relationships of *n* dimension classes. *Many-to-many* relationships between facts and particular dimensions are indicated by the *1..** cardinality on the dimension class role (Figure 1 (a)). For nonadditive measures, additive rules are defined as constraints and are included in the fact class. Furthermore, derived measures can also be explicitly considered (indicated by */*) and their derivation rules are placed between braces near the fact class (Figure 1 (a)). This OO approach also allows us to define identifying attributes in the fact class, by placing the constraint *{OID}* next to an attribute name. In this way we can represent *degenerate dimensions* [6] (Figure 1 (a)).

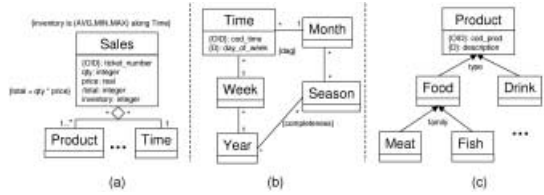


Figure 1. Multidimensional modeling using the UML

With respect to dimensions, every *classification hierarchy* level is specified by a *base class*. An association of classes specifies the relationships between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the dimension class (constraint *{dag}*). The DAG structure can represent both alternative path and multiple classification hierarchies. Every classification hierarchy level must have an *identifying* attribute (constraint *{OID}*) and a *descriptor* attribute (constraint *{D}*). The multiplicity *1* and *1..** defined in the target associated class role addresses the concepts of *strictness* and *non-strictness*. Moreover, defining the *{completeness}* constraint in the target associated class role addresses the completeness of a classification hierarchy (Figure 1 (b)). Finally, the *categorization of dimensions* is considered by means of generalization-specialization relationships (Figure 1 (c)).

3 Representing Multidimensional Models with XML

A relevant feature of a model should be its capability to share information in an easy and standard form. The XML [7] is rapidly being adopted as a specific standard syntax for the interchange of semi-structured data.

We have used the XML to store the MD models accomplished by our OO approach. The correct structure of the XML documents has been defined by means of a DTD. As an example of the DTD, the following fragment contains elements to express dimensions and their classification hierarchies:

```
<!ELEMENT DIMCLASSES (DIMCLASS+)>
<!ELEMENT DIMCLASS (ASOCLEVELS?, CATLEVELS?)>
<!ELEMENT ASOCLEVELS (ASOCLEVEL+)>
<!ELEMENT ASOCLEVEL (DIMATTS?, RELATIONASOCS?, METHODS?)>

<!ELEMENT DIMATTS (DIMATT+)>

<!ELEMENT DIMATT EMPTY>
<!ATTLIST DIMATT id ID #REQUIRED
   name CDATA #REQUIRED      atomic %Boolean; "true"

   type CDATA #REQUIRED      description CDATA #IMPLIED

   initial CDATA #IMPLIED    derivationRule CDATA #IMPLIED
   OID %Boolean; "false"     D %Boolean; "false">

<!ELEMENT RELATIONASOCS (RELATIONASOC+)>
<!ELEMENT RELATIONASOC EMPTY>
<!ATTLIST RELATIONASOC child IDREF #REQUIRED
                                          name CDATA #IMPLIED          description CDATA #IMPLIED

                                          roleA %Multiplicity; "1"     roleB %Multiplicity; "M"
                                                completeness %Boolean; "false">
                                          <!ATTLIST ASOCLEVEL     id ID #REQUIRED
                                          name CDATA #REQUIRED         description CDATA #IMPLIED

                                          showAtts %Boolean; "true"    showMethods %Boolean; "true"
                                                autoResize %Boolean; "true">
```

Notice that attributes within these elements in the DTD allow us to express all the MD properties at the conceptual level. For example, non-strictness may be defined by assigning the same value `M` to both attributes `roleA` and `roleB` in the DTD element RELATIONASOC, and completeness is defined by means of the DTD attribute `completeness`. Moreover, identifying and descriptor attributes within dimensions are defined using the DTD attributes `OID` and `D` in DIMATT.

Figure 2. An XML document displayed in Microsoft Internet Explorer

Figure 3. Model navigation in the web using a browser

Figure 2 shows the presentation of an XML document generated by our CASE tool in a popular web browser (*Microsoft Internet Explorer*). However, only some web browsers partly support the XML. As a consequence, we are currently forced to transform XML documents into HTML pages in order to publish them on the web. The best method to accomplish this task is the use of the XSLT [7], as a language for transforming XML documents. XSLT stylesheets describe a set of patterns (templates) to match both elements and attributes defined in a DTD, in order to apply specific transformations for each considered match.

We have used the XSLT processor *Instant Saxon 6.5*. This processor complies fully with XSLT 1.0 and also allows the use of some features from XSLT 1.1, such as `xsl:document` (this new instruction permits the creation of different outputs from the same XML document). Our XSLT stylesheet generates a collection of HTML pages with links between them.

The resulting HTML pages allow us to navigate through the different presentations of the model on a web browser. All the information about the MD properties of the model is represented in the HTML pages. For example, in Figure 3 the first page is shown as an example of navigation for one of the presentations that contains the definition of the Sales fact class. At the bottom, we can notice the name of the dimensions: Product and Time. These are links that allow us to navigate along the pages that contain the definition of those dimensions.

4 Conclusions and Future Works

We have presented how to store MD models accomplished by an OO conceptual approach, based on the UML, in XML documents. We have defined a DTD and we have provided XSLT stylesheets that allow us to generate different presentations of the same MD model on an HTML format. In this way, we have provided a standard format to represent main MD properties at the conceptual level.

We are considering whether to adapt our proposal to the new emerging standards. In this sense, we are currently working on providing an XML Schema instead of a DTD. With respect to the presentation, XSL FO can be used to specify in deeper detail the pagination, layout, and styling information of XML documents. However, there are no current tools that completely support XSL FO.

Furthermore, we are currently studying the Common Warehouse Metamodel (CWM), a Meta-Object Facility (MOF) compliant metamodel, as a common framework to easily interchange warehouse metadata. CWM stores metadata in the XML Metadata Interchange (XMI) format. This proposal provides designers and tools with common definitions but lacks the complete set of information an existing tool would need to fully operate.

References

1. Golfarelli, M., Rizzi, S.: A methodological Framework for Data Warehouse Design. In: Proc. of the ACM 1st Intl. Workshop on Data warehousing and OLAP (DOLAP'98), Washington D.C., USA (1998) 3–9
2. Sapia, C., Blaschka, M., Höfling, G., Dinter, B.: Extending the E/R Model for the Multidimensional Paradigm. In: Proc. of the 1st Intl. Workshop on Data Warehouse and Data Mining (DWDM'98). Volume 1552 of LNCS. (1998) 105–116
3. Tryfona, N., Busborg, F., Christiansen, J.: starER: A Conceptual Model for Data Warehouse Design. In: Proc. of the ACM 2nd Intl. Workshop on Data warehousing and OLAP (DOLAP'99), Kansas City, Missouri, USA (1999)
4. Abelló, A., Samos, J., Saltor, F.: A Framework for the Classification and Description of Multidimensional Data Models. In: Proc. of the 12th Intl. Conference on Database and Expert Systems Applications (DEXA 2001), Munich, Germany (2001) 668–677
5. Trujillo, J., Palomar, M., Gómez, J., Song, I.Y.: Designing Data Warehouses with OO Conceptual Models. IEEE Computer, special issue on Data Warehouses **34** (2001) 66–75
6. Kimball, R.: The data warehousing toolkit. 2 edn. John Wiley (1996)
7. World Wide Web Consortium (W3C): eXtensible Markup Language (XML). Internet: `http://www.w3.org/XML/`. (1997)

Implementing Data Mining in a DBMS

Konstantina Lepinioti and Stephen McKearney

Department of Computing Bournemouth University, Bournemouth, UK
{tlepinio,smckearn}@bournemouth.ac.uk

Abstract Abstract. We have developed a clustering algorithm called CLIMIS to demonstrate the advantages of implementing a data mining algorithm in a database management system (DBMS). CLIMIS clusters data held in a DBMS, stores the resulting clusters in the DBMS and executes inside the DBMS. By tightly coupling CLIMIS with the database environment the algorithm scales better to large databases. This is achieved through an index-like structure that uses the database to overcome memory limitations. We further improve the performance of the algorithm by using a technique called *adaptive clustering*, which controls the size of the clusters.

1 Introduction

We have developed a clustering algorithm, called CLIMIS, to demonstrate the advantages of tightly coupling a data mining algorithm with a database management system (DBMS). CLIMIS is based on the classic conceptual clustering algorithm COBWEB[1]. Unlike COBWEB, CLIMIS clusters data held in a DBMS, stores the resulting clusters in the DBMS and executes inside the DBMS.

One of the benefits of this approach is that by tightly coupling our algorithm with the database environment the algorithm scales better to large databases. This is achieved through an index-like structure that uses the database to overcome memory limitations. This structure makes CLIMIS memory independent.

Previous work on data mining investigated loose integration of data mining algorithms with a DBMS through ODBC-style technology[2]. ODBC, however, has a performance overhead that can affect the data mining process substantially. Our algorithm removes this overhead by executing as a database procedure using a direct link to the database.

2 Data Structure and Algorithm

We decided to base CLIMIS on COBWEB because COBWEB has a number of properties that complement the DBMS environment. It is an incremental, unsupervised algorithm that produces non-overlapping clusters. A DBMS is a dynamic system where new data arrives constantly. As COBWEB is incremental it supports restructuring of the clusters as new data is presented to the database. Our intention is for the end users to create the CLIMIS index in the same way they create database indexes. The unsupervised nature of COBWEB requires less user intervention during construction of the

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 70–72, 2002.

index. Finally, our goal is to integrate CLIMIS into the DBMS query language and non-overlapping clusters should be easier for the user to interpret.

CLIMIS builds a hierarchy of concepts stored as attribute-value probabilities using category utility[1] as a measure of quality. Most hierarchical algorithms, including COBWEB, require these probabilities to be stored at every level of the tree. The data structure that our algorithm uses stores data at the leaf nodes of the tree and uses database queries to calculate the internal nodes.

CLIMIS uses caching to avoid querying the database unnecessarily. The algorithm executes efficiently as the DBMS handles the data access. In addition, because only probabilities for the leaf nodes are stored on disc, the I/O cost of transferring data and the space required to store the entire tree are kept to a minimum. The clusters are stored in relations in the database, which makes them easy to access and manipulate. To ensure good performance the table storing attribute value probabilities is clustered on the attribute/values.

CLIMIS uses caching to reduce the cost of calculating category utility. Our study of the COBWEB algorithm has revealed that the most expensive part of category utility is the calculation of the conditional probability. Pre-calculated conditional probabilities are stored in memory and reused with a simple adjustment when new data arrives. In this way, the algorithm needs to access considerably less data to calculate category utility.

3 Clustering Large Databases

By default CLIMIS, like many conceptual clustering algorithms, builds a tree of clusters with singleton leaf nodes. Singletons do not convey a lot of information, as they do not represent a relationship between cases. Also, in large databases, creating singleton clusters for every record in the database produces a very tall tree, which affects performance. To further improve the performance of CLIMIS when mining large databases, we have adapted the clustering algorithm to perform *adaptive clustering*. Adaptive clustering sets a threshold size for each cluster and only sub-divides a cluster when its size reaches the threshold. The size of the database and the number of clusters that are required determine the threshold. This approach helps to reduce the size of the tree. Limiting the size of each cluster is a common technique [4,5] and we are currently investigating the accuracy of clusters generated in this way.

4 Application

The data structure used by CLIMIS allows data mining to be integrated with the querying system of the DBMS. This is achieved with the use of triggers[3] that update CLIMIS's data structure as new data arrives at the database. An additional benefit of storing the clusters within the database is that the entire database and data dictionary can be used to interpret the clusters.

5 Conclusions

Our approach is currently implemented in the Oracle Relational Database Management System[3] and can be applied to many commercially available DBMSs. We are currently investigating the overall performance and accuracy of the CLIMIS clustering algorithm.

References

1. Fisher D. H., 1987, Knowledge Acquisition Via Incremental Conceptual Clustering, Machine Learning (2), pp. 139-172.
2. Netz, A., Chaudhuri, S., Bernhardt, J., Fayyad, U., 2000, Integration of Data Mining and Relational Databases, in Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, pp. 285-296.
3. Oracle Relational Database Management System, 2002, `http://www.oracle.com/`.
4. Witten I. H., Frank E., 2000, Data Mining, Morgan Kaufmann Publishers.
5. Zhang T., Ramakrishnan R., Livny M., 1996, BIRCH: An Efficient Data Clustering Method for Very Large Databases, in Proceedings - ACM - SIGMOD International Conference on Management of Data, Montreal, Canada, pp. 103-114.

Fully Dynamic Clustering of Metric Data Sets

Stefano Lodi*

University of Bologna, Department of Electronics, Computer Science, and Systems
CSITE-CNR, viale Risorgimento 2, 40136 Bologna, Italy
slodi@deis.unibo.it

The goal of *cluster analysis* [10] is to find homogeneous groups, or *clusters*, in data. Homogeneity is often made precise by means of a *dissimilarity* function on objects, having low values at pairs of objects in one cluster. Cluster analysis has also been investigated in *data mining* [5], emphasising efficiency on data sets larger than main memory [4,6,8,9,16]. More recently, the growing importance of multimedia and transactional databases has stimulated interest in metric clustering, i.e. when dissimilarity satisfies the triangular inequality.

Most of the clustering algorithms proposed in data mining are static. However, in many important database applications data are dynamic. For example, a data warehouse collects updates from data sources during its off-line state, which lasts only shortly. Efficient clustering must exploit the opportunity to compute updated clusters from current clusters and the updates, that is, it has to be dynamic.

A *dynamic algorithm* [13] executes a *pre-processing stage*, followed by a sequence of *update stages*, which update the solution and an internal *state*, in response to updates of objects. If the accepted updates include both insertions and deletions, then we term the algorithm *fully dynamic*. Dynamic algorithms for metric data include IncrementalDBSCAN [3] and BUBBLE [6]. The former is hard to utilise on databases of transactions, because ordinary metrics misbehave in such domain. (See e.g. [9].) BUBBLE is only *partially* dynamic, i.e. accepts only insertions. Moreover, homogeneity is attained by minimising cluster radius, inducing thus a bias towards "globular" clusters.

In the following we discuss an external memory, graph-theoretic framework for the fully dynamic cluster analysis of metric data, based on dissimilarity transformation. Let (X, d) be a metric space and S a finite subspace of X. Let $x \in X$. With $\mathrm{NN}_S xk$ we denote the set of the k nearest neighbours of x in S, and with $\mathrm{NN}_S^{-1} xk$ the set of all points in S having x among their k nearest neighbours.

Cluster analysis based on dissimilarity transformation can be generalised by the following schema, where $K \in \mathbb{N}$, $\theta \in \mathbb{R}^+$ are parameters: (i) For each $y \in S$ compute $\mathrm{NN}_S yK$. (ii) Compute a transformed dissimilarity function, δ_{xy}, which is both *local*, i.e. δ_{xy} depends only on $\mathrm{NN}_S xK$ and $\mathrm{NN}_S yK$, and *sparse*, i.e. $\delta_{xy} = \infty$ if $\{x, y\} \cap (\mathrm{NN}_S xK \cup \mathrm{NN}_S yK) = \emptyset$. (iii) Compute the clusters as connected components of the graph on S whose edge set contains exactly the edges (x, y) such that $\delta_{xy} \leq \theta$.

Instantiations of such schema can be found in [7,12,15], where it is shown that dissimilarity transformation allows for accurate recognition of arbitrarily shaped clusters and a dramatic reduction of the chaining effect. In particular, we believe that method [12] is robust even on transactional data.

* This work has been partially supported by project D2I of the Italian MIUR (http//:www.dis.uniroma.it/~lembo/D2I).

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 73–76, 2002.

We now turn this scheme into a fully dynamic scheme. Let S be the initial instance. We store the following information in the state of the algorithm: (i) The sparse undirected weighted graph G_δ induced byδ, (ii) The connected components of the graph obtained from G_δ by removing all edges weighing more than θ. (iii) Two sparse directed graphs G_K and G_K^{-1}, storing, for each point $x \in S$, $\mathrm{NN}_S xK$ and $\mathrm{NN}_S^{-1} xK$, respectively.

The initialisation stage computes G_K, G_K^{-1}, G_δ and the output clustering as the set of connected components of the graph obtained by deleting from G_δ all edges having weight greater than θ. The update stage is structured into two layers. The outer layer performs a detailed analysis of the neighbourhoods of points influenced by the insertion or deletion and determines which updates to G_δ, G_K, G_K^{-1} are needed. The inner layer performs the requested updates and calls a fully dynamic algorithm to re-compute the connected components from the current components and the updates.

For $x, y \in X$, let the *rank* $\mathrm{rank}\, r_x^y y$ of y w.r.t x in S be the number of points in S that are closer to x than y, plus one. If $\mathrm{rank}\, r_x^y y = k$, then y is the kth neighbour of x. Let x, y be *mutual neighbours* if the rank of x w.r.t. y is at most K, and vice versa. We briefly outline the outer layer for the methods in [7] and [12]. Let $x^* \in X$ be an inserted or deleted point.

In [7], δ_{xy} is defined as $\mathrm{rank}\, r_y^x x + \mathrm{rank}\, r_x^y y$, if x and y are mutual neighbours, and ∞ otherwise. We retrieve all points $y \in S$ such that x^* has rank at most K w.r.t. y. For both insertion and deletion, if x^* and y are mutual neighbours, we remove an edge from G_δ, or add an edge with appropriate dissimilarity to G_δ. Then we retrieve the K nearest neighbours of y, in case of insertion, or $K+1$ nearest neighbours in case of deletion. The retrieved points may be at the other end of a new edge, or an edge to be removed, or an updated edge.

In fact, in case of insertion, an edge to the furthest neighbour of y, say z, must be removed if y and z are mutual neighbours, because x^* is inserted in the rank order before z, and z and y cease to be mutual neighbours. In case of deletion, an edge between y and its $(K+1)$th neighbour, say z, must be added if the rank of y w.r.t. z is at most K after the insertion of x^*. For points z that are non-furthest neighbours, the corresponding edge weight must be incremented, in case of insertion, or decremented, in case of deletion, by one or two units, depending on the rank of the new point x^* with respect to y and z.

In [12], δ_{xy} is defined as the number of neighbours of rank at most K shared by x and y, if x and y are mutual neighbours, and ∞ otherwise. The cases for adding or removing edges in G_δ are similar to the method in [7].

In the remaining cases, in order to determine if the number of neighbours of rank at most K that are shared by y and z changes, we need to compute the rank of the Kth neighbour of y w.r.t. z and vice versa, for insertion, or $(K+1)$th neighbour, for deletion. Moreover, the insertion of x^* with rank at most K w.r.t. both y and z, or deletion when its rank is at most K w.r.t. both y and z, may increment or decrement the number of shared neighbours.

Preliminary complexity results for the outer layer show the following. If we let F, B be the worst-case amount of time used to retrieve $\mathrm{NN}_S xK$, $\mathrm{NN}_S^{-1} xK$, respectively, and $|\mathsf{F}|$, $|\mathsf{B}|$ their respective cardinality, then the worst-case cost to list all necessary graph

updates after an insertion or deletion is a low order function of K, F, B, $|\mathsf{F}|$, $|\mathsf{B}|$, for all methods in [7,12,15]. In practice, we expect the dominant term to be proportional to $|\mathsf{B}| \cdot \mathsf{F}$.

Notice that dynamic metric access methods, such as the M-tree [2], efficiently support queries of the form $\mathrm{NN}_S xk$. Queries like $\mathrm{NN}_S^{-1} xk$ require a work around. If spatial objects defined by center and radius are supported, then we may construct an auxiliary structure containing a spatial object (x, ρ) for every point $x \in S$, where ρ is the distance from x to its Kth neighbour. In such structure, a *point query* at x returns the set $\mathrm{NN}_S^{-1} xk$. We are planning an in-depth experimental investigation on the actual costs of the queries involved in the outer layer.

Concerning the inner layer, a *dynamic graph algorithm* must be invoked. For clustering in internal memory, we may use the approach in [11] due to Holm et al., which achieves worst-case complexity $O(\log^2 N)$ for updates and $O(\log N / \log\log N)$ for connectivity queries.

For clustering in external memory, following [1], an external memory fully dynamic algorithm for connected components can be derived from parallel algorithms for the minimum spanning forest. If we evaluate I/O efficiency according to the *parallel disk model* [14], then a single update of the dissimilarity graph G_δ takes just a number of I/Os that is bounded by the number of I/Os needed to optimally sort N items in external memory.

Since clusters are usually stored as point-cluster look-up tables, in order to guarantee that connectivity queries can be answered in $O(1)$ I/Os, connected components in such explicit representation can be computed from the MSF again at a cost bounded by the cost of optimal external sorting (using the Euler tour technique in external memory).

References

1. Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, San Francisco, California, 22–24 January 1995.
2. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.
3. Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. 24th Int. Conf. Very Large Data Bases (VLDB)*, pages 323–333, 24–27 August 1998.
4. Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 226. AAAI Press, 1996.
5. Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, March 1996.
6. Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison Powell, and James French. Clustering large datasets in arbitrary metric spaces. In *Proc. 15th IEEE Conf. Data Engineering (ICDE)*, 23–26 March 1999.

7. K. Chidananda Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, 10(2):105–112, 1978.
8. S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 73–84, New York, June 1–4 1998. ACM Press.
9. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia*, pages 512–521. IEEE Computer Society, 1999.
10. John A. Hartigan. *Clustering algorithms*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons., New York, 1975.
11. Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 79–89, Dallas, Texas, USA, May 23–26 1998. ACM.
12. R. Jarvis and E. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, 22(11):1025–1034, November 1973.
13. Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, August 1994.
14. Jeffrey Scott Vitter. External memory algorithms and data structures. In James Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Press, Providence, RI, 1999.
15. M. Anthony Wong and Tom Lane. A *k*th nearest neighbour clustering procedure. *J. R. Statist. Soc. B*, 45(3):362–368, 1983.
16. Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Quebec, Canada, 4–6 June 1996. *SIGMOD Record* 25(2), June 1996.

Real World Association Rule Mining

Adriano Veloso, Bruno Rocha, Márcio de Carvalho, and Wagner Meira Jr.

Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
{adrianov,gusmao,mlbc,meira}@dcc.ufmg.br

Abstract Across a wide variety of fields, data are being collected and accumulated at a dramatic pace, and therefore a new generation of techniques has been proposed to assist humans in extracting usefull information from the rapidly growing volumes of data. One of these techniques is the association rule discovery, a key data mining task which has attracted tremendous interest among data mining researchers. Due to its vast applicability, many algorithms have been developed to perform the association rule mining task. However, an immediate problem facing researchers is which of these algorithms is likely to make a good match with the database to be used in the mining operation. In this paper we consider this problem, dealing with both algorithmic and data aspects of association rule mining by performing a systematic experimental evaluation of different algorithms on different databases. We observed that each algorithm has different strengths and weaknesses depending on data characteristics. This careful analysis enables us to develop an algorithm which achieves better performance than previously proposed algorithms, specially on databases obtained from actual applications.

1 Introduction

The capabilities of both generating and collecting data have rapidly increased during the last years. Advances in commercial and scientific data collection have generated a flood of data. Advanced database management systems and data warehousing technology allow to gather the flood of data and to transform it into databases of an enormous size.

Due to this situation, a number of techniques with the ability to intelligently and automatically transform the processed data into knowledge have been proposed. The most widely studied of these techniques is the association rule mining, which has an elegantly simple problem statement, that is, to find the sets of all subsets of items that frequently occur as database transactions, and to infer rules showing us how a subset of items influences the presence of another subset. The prototypical application is the analysis of sales on *basket* data, but besides this example, association rules have been shown to be useful in domains such decision support, university enrollments, e-commerce, etc.

Since its origin in [1], several algorithms [2,3,4,5,6,7,8,9,10,11] have been proposed to address the association rule mining task, and we observed that, although these algorithms share basic ideas, they present different strengths and weakness depending on database characteristics. Neglecting this fact, little work has been devoted to the data aspect in the knowledge discovery process, but for real world applications practitioners have to face the problem of discovery knowledge from real databases with different characteristics. Furthermore, the extant algorithms were bench-marked on artificial

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 77–89, 2002.

datasets, but the performance improvements reported by these algorithms on artificial data do not generalize when they are applied to real datasets, and these artificial improvements can also cause problems to practitioners in real world applications.

In this paper we deal both with the algorithmic and data aspects of association rule mining. We conduct an extensive experimental characterization of these algorithms on several artificial and real databases, presenting a systematic and realistic set of results showing under which conditions an algorithm is likely to perform well and under what conditions it does not perform well. We also study the differences between the artificial datasets, generally used as benchmarks, and real datasets from actual applications (i.e., e-commerce), showing how much these differences can affect the performance of the algorithms. Furthermore, we realized that not only do there exist differences between real and artificial data, but there also exist differences between real datasets. This careful analysis enables us to develop a superior algorithm, which achieves better runtimes than the previous algorithms, especially on real datasets.

The paper is organized as follows. The problem description and related works are given in the next section. In Section 3, we systematize the most common algorithms and the strategies used by each one. Section 4 presents our new algorithm, ADARM (ADaptive Algorithm for Association Rule Mining). In Section 5 we discuss and present the analysis of the algorithms described in the preceding section. Section 6 summarizes our paper and closes with interesting topics for future work.

1.1 Research Contributions of this Paper

Our first contribution is a complete experimental comparison against the state-of-the-art algorithms for mining association rules. In our experiments we employed not only synthetic datasets. In fact, as opposed to the great majority of previous works, we also employed different real datasets from actual applications. Understanding how each technique behaves on different types of data is the first step of developing efficient algorithms.

The other important contribution is ADARM, the algorithm developed during this work. ADARM combines relevant features of other algorithms, which highlight the advantages of each one depending on data characteristics. These features combined together, provide to ADARM a better performance over different types of databases.

2 Problem Description and Related Works

The goal of association rule mining is to discover if the occurrence of certain items in a transaction can imply the occurrence of other items, or in other words, to find associative relationships between items. If indeed such interesting relationships are found, they can be put to various profitable uses such as personalization, recommendations, etc. Given a set of items, we must predict the occurrence of another set of items with a certain degree of confidence. This problem is far from trivial because of the exponential number of ways in which items can be grouped together. To state this problem we need some definitions. Let $I = \{I_1, I_2, ..., I_m\}$ be a set of m attributes or items. Let D be a set of transactions where each transaction is a subset of I, and the transaction is uniquely identified by a tid. Let C be a subset of I, also called an itemset. If C has k items it is

called a k-itemset. We define the $support$ of C with respect to D to be:
$\sigma(C) = \mid \{t \mid t \in D, C \subseteq t\} \mid$
For example, consider a set of transactions as shown in Fig. 1. The items set I for these transactions is $\{Bread, Beer, Diaper, Milk\}$. The support of $\{Diaper, Milk\}$ is $\sigma(Diaper, Milk) = 3$.

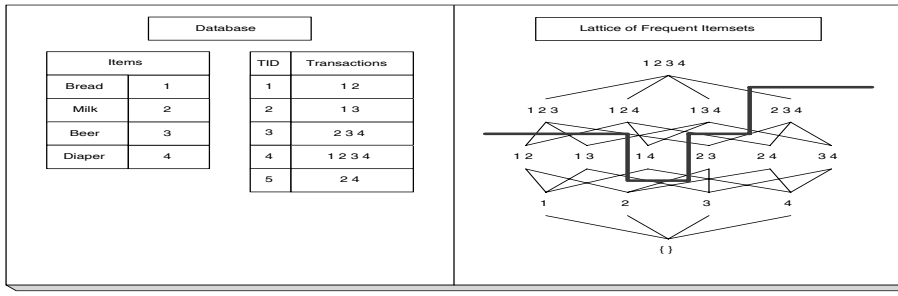


Figure 1. Transactions and Search-Space of Itemsets (minsup=40%)

An association rule is an expression of the form $X \Rightarrow Y$, where $X \subseteq I$ and $Y \subseteq I$. The support s of the rule $X \Rightarrow Y$ is defined as $\sigma(X \cup Y)/ \mid D \mid$, and the confidence c is defined as $\sigma(X \cup Y)/\sigma(X)$. For example,consider a rule $\{Diaper, Milk\} \Rightarrow \{Beer\}$ (i.e. presence of diaper and milk in a transaction tends to indicate the presence of beer in the same transaction). The support of this rule is $\sigma(Diaper, Milk, Beer)/5 = 40\%$. The confidence of this rule is $\sigma(Diaper, Milk, Beer)/\sigma(Diaper, Milk) = 66\%$. A rule that has a very high confidence is often very important, because it provides an accurate prediction on the association of the items in the rule. The support of a rule is also important, since it indicates how frequent the rule is in the transactions. This is one of the reasons why most algorithms disregard any rules that do not satisfy the minimum support condition specified by the user. This filtering due to the minimum support required is also critical in reducing the number of association rules to a manageable size. The number of possible rules is proportional to the number of subsets of I, which is $2^{|I|}$. Hence the filtering is absolutely necessary in most practical settings.

The task of discovering association rules is to find all rules $X \Rightarrow^{s,c} Y$, such that s and c are at least equal to the corresponding thresholds. This task is composed of two steps. The first step is to find all the frequent itemsets, and the second step is to generate association rules for these frequent itemsets. The first step is much more expensive than the second. Hence major efforts in the research community have been focused on finding algorithms to compute the frequent itemsets, as in this work.

2.1 Related Work

Several algorithms for mining association rules have been proposed in the literature. The APRIORI algorithm [2] is the best known previous algorithm, and it uses an efficient candidate generation procedure, such that only the frequent itemsets at a level

are used to construct candidates at the next level. However it requires multiple database scans. Many variants of APRIORI have arisen. The DIC algorithm [3] dynamically counts candidates of varying length as the database scan progresses, and thus is able to reduce the number of scans. The PASCAL algorithm [9] introduces a novel optimization, called *counting inference*, which can infer the support of a candidate without accessing the database. The ECLAT algorithm [4] is based on *vertical database format* and lattice problem decomposition. These two features allow a better usage of resources. The FPGROWTH algorithm [10] uses a novel structure called *frequent pattern tree*, which avoids generating a huge number of candidates.

All above algorithms generate all frequent itemsets. Methods for finding the maximal frequent itemsets include the MAXMINER algorithm [7], which uses pruning techniques to quickly narrow the search space. The MAFIA algorithm [8] uses pruning strategies to remove non-maximal itemsets, but it mines a superset of the maximal frequent itemsets and requires another step to eliminate non-maximal itemsets. The GENMAX algorithm [6] uses a backtrack search and a number of optimizations to prune the search space for maximal frequent itemsets. Recently, some interest also has been devoted to the closed frequent itemset mining. The ACLOSE algorithm [11] is based on Apriori and directly mines closed frequent itemsets. ACLOSE first identifies the generators, that is, the smallest frequent itemsets that determines a closed itemset, then it computes the closure of all generators found. The CHARM algorithm [5] improves the search for closed itemsets . It prunes candidates based not only on subset in-frequency but on non-closure property as well, making fewer database scans than ACLOSE.

Although there exists a large number of proposed algorithms, not one of them was developed exclusively for actual applications. Further, the efficiency of these algorithms is generally very dependent on database characteristics. These are the main topics of this paper.

3 Algorithms for Mining Frequent Itemsets

Three approaches have been proposed for mining frequent itemsets. The first is traversing iteratively the set of all itemsets in a bottom-up or level-wise manner. During each iteration corresponding to a level, a set of candidates is generated by joining the frequent itemsets discovered during the previous iteration, the supports of all candidates are counted and infrequent ones are discarded. This approach works because an itemset cannot be frequent if it has an infrequent subset.

The second approach is based on the extraction of maximal frequent itemsets [1], from which all subsets are frequent. As a very important characteristic of these algorithms, it is not necessary to explicitly examine every single frequent itemset, potentially reducing the number of candidates examined for finding all frequent itemsets. However, just determining the maximal frequent itemsets is not sufficient enough to generate association rules, since we do not have the support associated with each subset. This is the main negative aspect of this approach, since a complete scan over the entire database is required in order to compute the frequency of each subset.

[1] A frequent itemset is maximal if it has no frequent superset.

The third approach does not necessarily need to examine all frequent itemsets. In this approach the frequent closed itemsets[2], which is the smallest representative subset of all frequent itemsets without loss of information, are extracted from the database in a level-wise manner. Then, all frequent itemsets, as well as their supports, are derived from the frequent closed itemsets and their supports without accessing the database. Hence not all itemsets are explicitly examined, and no additional scan is necessary in order to generate association rules.

3.1 The APRIORI Algorithm

APRIORI is the first efficient algorithm for association rule mining. It introduces a pruning trick that all subsets of a frequent itemset must also be frequent, thus infrequent itemsets are quickly pruned. It performs a breadth-first search for frequent sets, allowing an efficient candidate generation, since all frequent sets at a level are known when starting to count candidates at the next level. A depth-first search would improve the memory utilization by maintaining less candidates, but APRIORI cannot perform this type of search. The reason is evident when considering how the horizontal database format works: First, a set of candidates is set up, second, the algorithm passes all transactions and increments the counters of the candidates. That is, for each set of candidates one pass over all transactions is necessary. With a breadth-first search such a pass is needed for each level of the search space. In contrast, a depth-first search would make a separate pass necessary for each class that contains at least one candidate. The costs for each pass over the whole database would contrast with the relatively small number of candidates counted in the pass. Although APRIORI employs an efficient candidate generation, it is extremely I/O intensive, requiring as many passes over the database as the size of the longest candidate. This process of support computation may be acceptable for sparse databases with short-sized transactions. However, when the itemsets are long, APRIORI may require substantial computational efforts, hardly debasing its scalability. Thus, the success of APRIORI critically relies on the fact that the length of the frequent itemsets in the database are typically short.

3.2 The ECLAT Algorithm

ECLAT uses vertical database format, where it has available for each itemset its tidset, the set of all transaction tids where it occurs. The vertical format has the following major advantages over the horizontal format: Firstly, computing the support is faster with the vertical layout since it involves only the intersections of the tidsets. Secondly, there is an automatic "reduction" of the database before each scan in that only those itemsets that are relevant to the following scan of the mining process are accessed from disk. Finally, the vertical layout is more versatile in supporting various search strategies.

ECLAT performs a depth-first search for frequent itemsets, which is based on the concept of *equivalence classes* [4]. Two *k*-itemsets belong to the same equivalence class F_K if they share a common *k-1* length prefix. Each class may be processed independently, thus ECLAT divides the lattice of itemsets into smaller sub-lattices, where each one corresponds to an equivalence class and is processed independently in memory,

[2] A frequent itemset is closed if it has no frequent superset with the same frequency.

minimizing the secondary memory operations. The depth-first search starts from 1-itemsets and keeps intersecting tidsets of itemsets within the same equivalence class.

In contrast to APRIORI, ECLAT does not know all frequent itemsets at a level before starting the computation of the candidates at the next level, diminishing pruning effectiveness. For instance, ECLAT only knows the frequent itemsets within each equivalence class. This low-quality of pruning may be acceptable when the database is small, and the cost to process an infrequent candidate is not worst than the cost of verifying if all its immediate subsets are frequent. However, in large databases where we may really avoid database scans, other strategies of pruning become necessary.

3.3 The GENMAX Algorithm

GENMAX [6] uses vertical database layout and a backtrack search to enumerate all maximal frequent itemsets. The maximal frequent itemsets solely determine all frequent itemsets, and they are specially interesting in dense databases, where it is infeasible to explicitly enumerate all frequent itemsets. The efficiency of GENMAX stems from the fact that it eliminates branches that are subsumed by an already mined maximal frequent itemset. In the backtrack search the solution is represented as a set $I = \{i_0, i_1, ...\}$, where each i_j is chosen from a finite *possible set*, P_j. Initially I is empty; it is extended one item at a time, as the search space is traversed. Given a candidate of length l, $I_l = \{i_0, i_1, ..., i_{l-1}\}$, the possible values for the next item i_l comes from a subset $C_l \subseteq P_l$ called the *combine set*. If $y \in P_l - C_l$, then nodes in the subtree with root node $I_l = \{i_0, i_1, ..., i_l - 1, y\}$ will not be considered by GENMAX. Each iteration tries to extend I_l with every item x in the combine set C_l. An extension is valid if the resulting itemset I_{l+1} is frequent and is not a subset of any already known maximal frequent itemset. The next step is to extract the new possible set of extensions, P_{l+1}, which consists only of items in C_l that follow x. The combine set, C_{l+1}, consists of those items in the possible set that produce a frequent itemset when used to extend I_{l+1}. The backtrack search performs a depth-first traversal of the search space.

3.4 The CHARM Algorithm

CHARM mines the set of frequent closed itemsets, which is much smaller than the set of all frequent itemsets. CHARM is unique in that it simultaneously explores both the itemset space and transaction space, unlike the other methods that which only explores the itemset search space. It prunes candidates based not only on subset in-frequency, but also based on no-closure property. CHARM uses vertical database format, and the two basic operations are an union of two itemsets and an intersection of their tidsets. The search method employed by CHARM is similar to the backtrack search employed by GENMAX but instead of maximality checking CHARM performs closure checking.

4 Real World Association Rule Mining

In this section we present the motivations of developing algorithms for association rule mining on real databases. Also, we present ADARM, an algorithm for real-world association rule mining which combines some of the techniques described above. In contrast with other works, we always keep in mind the need to develop an algorithm for

real databases. Thus, before developing ADARM, we studied some characteristics of different real datasets, and their differences from the synthetic ones. Next we describe the dataset's characteristics and which features were selected in the development of ADARM.

4.1 Dataset Characteristics

Now we will describe some data characteristics, which will help us to better understand the effectiveness of the selected features. Both synthetic and real datasets were used to help the development of ADARM. The synthetic datasets mimic the transactions in a retailing environment, where people tend to buy sets of items [4]. We have employed two synthetic datasets that are described next. The first one, called T10.I4.D50K, comprises 50,000 transactions over 1,000 unique items, each transaction containing 10 items on average. The second synthetic dataset, called T25.I10.D10K, comprises 10,000 transactions over 500 unique items, each transaction has 25 items on average. We also evaluated the characteristics of real data from actual applications. The first real dataset, VBook, was extracted from a 45-day log from an electronic bookstore and comprises 136,809 customer transactions. Here, a transaction contains the books bought by a customer during a visit to the bookstore. VBook has 10,249 items, and the average size of the transactions is 5. The second actual dataset is called KddCup, which comes from click-stream data from a small dot-com company called Gazelle.com, a leg-ware and leg-care retailer, which no longer exists. A portion of this dataset was used in the KDD-Cup 2000 competition. It has 59,601 transactions over 498 items with an average transaction length of 2.5 and a standard deviation of 4.9. The last real dataset represents the access patterns of a WEB portal. The dataset, WPortal, comprises 432,601 transactions over 1,182 unique items, and each transaction contains on average 3.6 items.

We begin our study observing the average size of transactions. Figure 2(a) shows a big difference between artificial and real datasets. As we can observe, in the real datasets the short-sized transactions are much more frequent than the others. In contrast, the artificial datasets present more medium-sized transactions. We also observed the frequency distribution of the items in these datasets. Figure 2(b) shows that, not only real and artificial data are different, but also there exist differences between the real datasets themselves. The synthetic datasets present a constant frequency distribution, which is followed by an abrupt frequency decay. A similar behavior is observed on the KddCup dataset from a retail application, but is not observed on the other real datasets. Finally, Figure 2(c) depicts the frequency distribution of frequent itemsets. Surprisingly, all real datasets present different frequency distributions. Despite the obvious differences between real and artificial data [12], there also exist differences between the real datasets. This complicates the development of algorithms for real-world association rule mining, which must use several techniques that, combined together, provide a more homogeneous behaviour over different types of databases, adapting to database characteristics.

4.2 The ADARM Algorithm

In this section we present the features selected in the development of ADARM. Some features are specially effective depending on database characteristics. Our algorithm is adaptive in that the computational efforts performed by each feature are also utilized, if needed, by other feature.

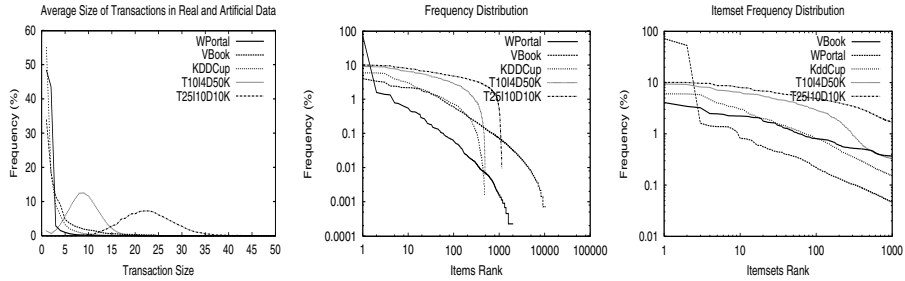


Figure 2. a) Transactions Distributed by Size (left), b) Popularity Distribution of the Items (middle), and c) Popularity Distribution of Frequent Itemsets (right)

Database Layout. Figure 3 shows a comparison of the time spent by both APRIORI and ECLAT for processing frequent itemsets of different sizes. As we can observe, the computation of 2-itemsets in ECLAT is extremely inefficient, both in synthetic and real data, when compared to APRIORI. This fact was first reported in [5], and it is caused by the vertical database format, which in the worst case, performs n^2 tidset intersections, where n is the number of frequent 1-itemsets. Contrast this with the horizontal approach used by APRIORI that performs only $O(n)$ operations. It is well-known that a large number of 2-itemsets are not frequent and, worse still, note that the tidsets of each 1-itemset involved in the intersection are generally the longest ones, since the 1-itemsets are more frequent, making a single intersection even more costly.

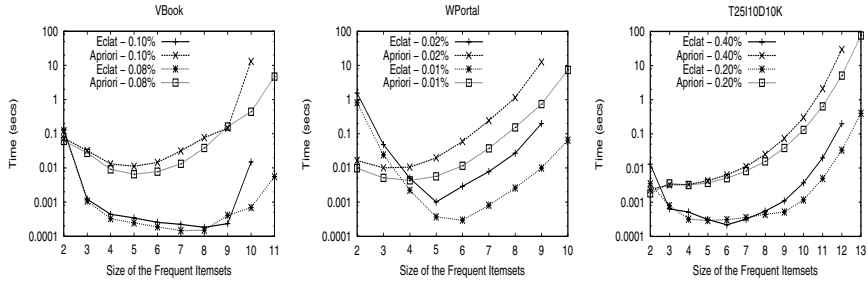


Figure 3. Average Time spent for processing each Itemset of different Size

After all 2-itemsets are found, ECLAT achieves better performance than APRIORI. This is because the reduction in the size of the tidsets show the advantages of the vertical format employed by ECLAT when compared to APRIORI which performs several data scans. These observations show us that we must employ an hybrid approach, which initially uses the horizontal format, like APRIORI, and then starts to use the vertical format, like ECLAT.

Counting Inference. The *counting inference* strategy was introduced as a novel optimization of APRIORI. We refer the reader to [9] for a fuller description of this strategy, which relies on the concept of key patterns[3], from which the support of the non-key patterns can be inferred without accessing the database. To achieve more efficiency by using this strategy, the algorithm must keeps a large number of subsets (i.e., only their supports), while it is generating the respective supersets. This is because the larger the number of subsets verified, the larger is the chance of finding a key-pattern. In that way, a breadth-first search would achieve better results. However, if one changes the depth-first search for maintaining the supports of the subsets already computed, the modified algorithm would achieve an acceptable efficiency.

Finally, this strategy is very sensitive, both to data characteristics and to the minimum support used. As we may observe in Figure 2(c), for lower support values, some datasets present a step-based behavior which indicates a large number of itemsets with the same support. This fact indicates that a potentially large number of support inferences will occur while mining this database.

Pruning Techniques and Database Scans. As mentioned earlier, to perform counting inference, the algorithm will have to keep the support of some subsets already computed. However, if this strategy does not work well, we can use the information kept to perform the pruning inter-classes, that is, before a candidate is evaluated over the database, we observe if all its immediate subsets are frequent. If not, so we must remove this candidate from consideration. On such pruning, the branch reordering is very important, since if we first generate infrequent itemsets we will prune a large number of candidate which are supersets of these infrequent subsets. Figure 2(b) shows that this branch reordering will not be effective for pruning in synthetic datasets, since the items present a similar support. In contrast, this pruning seems to be very efficient on real datasets, since the items present very different supports (excepting the KddCup).

Even if this pruning does not work well, we can use the computation performed to find the two smallest tidsets in order to perform the least costly intersection. Note that, an itemset can be computed based on any two of its subsets, and choosing the two with the smallest support is a very efficient way to minimize the cost to process a candidate.

Figure 4 presents the results of running ADARM on different datasets. As can be observed in Figure 4(a) the pruning was more effective on real data, as was the counting inference, depicted in Figure 4(b). These two features cause a decrease in the number of candidates in real data. Figure 4(c) shows the improvement in choosing the smallest tidset to perform the least costly intersection. The improvements were caused both by the smaller number of candidates generated by ADARM and the least costly tidset intersections. As expected, each feature presented different effects on different datasets, but surprisingly, in a given dataset, the most effective feature may vary depending on the minimum support employed, highlighting again the need of developing adaptive algorithms, even if they will be applied to a specific database.

[3] An itemset is a key pattern if it has no proper subset with the same frequency.

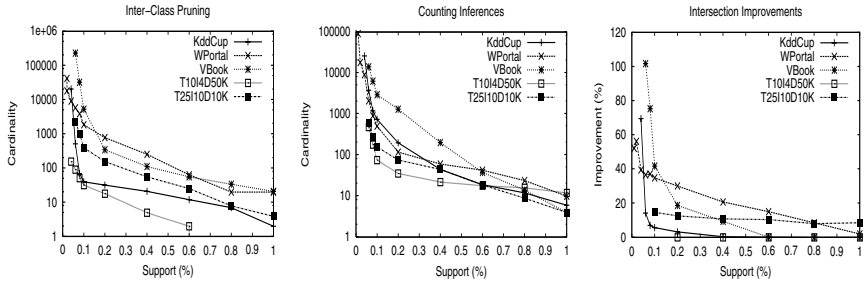


Figure 4. a) Pruning (left), b) Inferences (middle), c) Intersection Cost (right)

5 Experimental Evaluation

In this section we present the performance evaluation and comparison of five state-of-the-art algorithms for association rule mining: APRIORI, ECLAT, FPGROWTH, CHARM and GENMAX [4]. We also compared these algorithms against ADARM running on five datasets, which were described in the last section. Two of the datasets are synthetic: T10.I4.D50K and T25.I10.D10K, while three come from different e-commerce applications: WPortal, VBook and KddCup. The performance measure was the total execution time. All the experiments were run on a 750MHz processor with 1GB main memory.

T10.I4.D50K. Figure 5 shows the performance of the algorithms on a sparse dataset. On such dataset, for higher supports, APRIORI is the best algorithm, while ADARM is the worst. As the minimum support decreases the performance of APRIORI also decreases and quickly becomes the worst algorithm for lower supports. The features used in the development of ADARM are not very effective on such datasets. The number of frequent itemsets is large compared to the number of candidates and nearly all frequent itemsets are key patterns. On this dataset, FPGROWTH is the best algorithm, imposing an improvement of more than one order of magnitude over APRIORI.

T25.I10.D10K. This dataset presents long-sized transactions, making APRIORI the worst algorithm in all cases, as can be seen in Fig. 6. Again, FPGROWTH presents the best results, being very scalable on such datasets. Finally, ADARM is slightly better than ECLAT and GENMAX. In fact, in this dataset the proportion of itemsets which are not key patterns is very significant, enabling ADARM to reduce considerably the number of support counts performed.

[4] An additional scan was performed in order to find the support of all subsets.

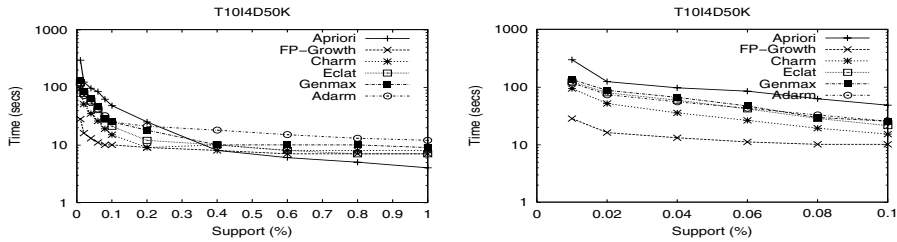


Figure 5. Running Times on the T10I4D50K dataset

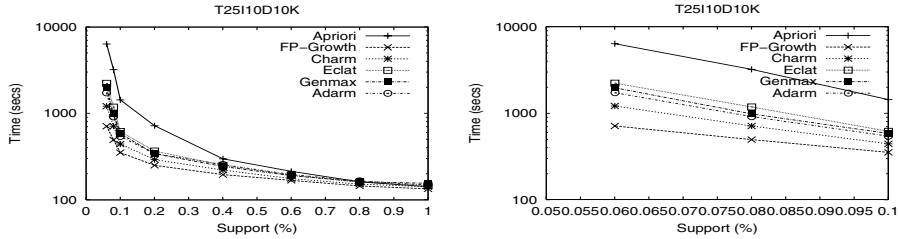


Figure 6. Running Times on the T25I10D10K dataset

WPortal - Access Patterns. As we can observe in Fig. 7, ADARM is the best algorithm on such dataset. This can be explained by the high pruning effectiveness. We observed that the equivalence class formed by the item "index.html" comprises many more itemsets than the other equivalence classes. Note that, in this situation, the branch reordering can provide a great improvement in the pruning, since a potentially large number of infrequent subsets were already verified in other classes. Again, APRIORI was the worst performer, but the differences are not as large as on the synthetic datasets.

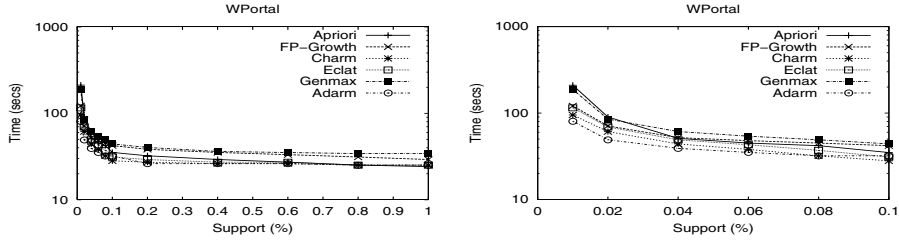


Figure 7. Running Times on the WPortal dataset

VBook - Customer Preferences Patterns. This dataset is composed of strongly correlated items, where a large number of itemsets are not key patterns. Hence using counting

inference, ADARM has to perform many fewer support counts than APRIORI, ECLAT, GENMAX and FPGROWTH. The same observation holds for CHARM, that uses the closure mechanism to reduce the number of support counts. Due to a little improvement over CHARM, ADARM is the best performer on this dataset, as can be seen on Fig. 8. We believe that the advantage of ADARM over CHARM was caused by the least costly intersection performed by ADARM, demonstrating the efficiency of ADARM.

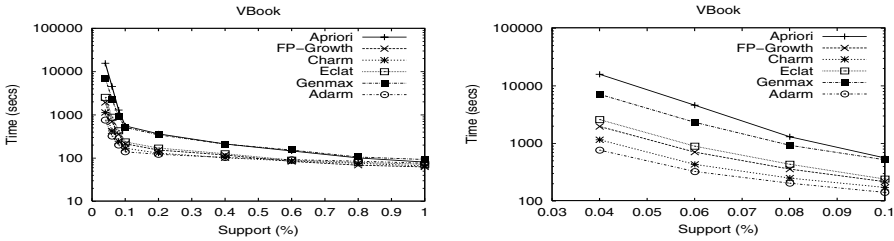


Figure 8. Running Times on the VBook dataset

KddCup - Click-Streams. Figure 9 depicts the performance of the algorithms on the KddCup dataset. For higher supports, ADARM does not perform well, contrasting with APRIORI and ECLAT, which perform the best. Decreasing the minimum support, both APRIORI and GENMAX lose the scalability. Definitively, a maximal based approach does not work well for lower supports on the real datasets used. In contrast, ADARM and CHARM stay scalable and remain competitive until the minimum support reaches 0.06%. From this value on, as we can observe in Fig.s 4(a), (b), and (c), the strategies employed by ADARM present an abrupt growth in effectiveness. This high efficiency makes ADARM the best performer for lower supports on such database.

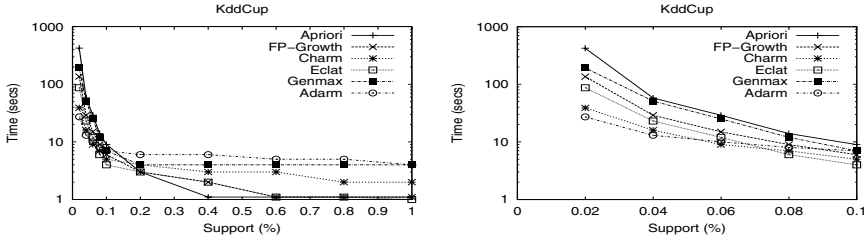


Figure 9. Running Times on the KddCup dataset

6 Summary

In this paper we show some differences between real and synthetic datasets. These differences are inducing an artificial improvement, since the algorithms are over-fitting the synthetic datasets. Most importantly, we show that even the real datasets themselves present some differences, turning extremely hard the development of association rule mining algorithms for real world applications, since on such real environments, the algorithms must be adaptive in order to do not present dependences on specific data characteristics. Finally, we present ADARM, an adaptive algorithm developed for real world association rule mining. It uses several features of other algorithms, which are combined in such a way that minimizes both computational costs and data dependence, providing to ADARM a better performance on different real datasets.

Our work opens up important avenues for future works. The synthetic dataset generator appears to be too restrictive, generating obsolete data distributions. It is well known that the frequency distribution of WWW patterns and requests obeys the *Zipf* law. We plan to develop a new generator that the users can use to produce distributions which obey the *Zipf* law. Another obvious avenue is to improve ADARM to combine more features, and potentially achieving better performance on more types of data.

References

1. R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of the Conf. on Management of Data*, 1993.
2. R. Agrawal, A. Swami. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, September 1994.
3. S. Brin, R. Motwani, J. Ullman. Dynamic Itemset Counting and Implication Rules for Market Basket Data . In *Proc. of the Intl. Conf. on Management of Data*, 1997.
4. M. Zaki, S. Parthasarathy, W. Li. Algorithms for Fast Discovery of Association Rules. In *Proc. of the 3rd Conf. on Knowledge Discovery and Data Mining*, 1997.
5. M. Zaki, C. Hsiao. ChARM: An Efficient Algorithm for Closed association Rule Mining. *Technical Report 99-10, Rensselaer Polytechnic Institute*, October 2000.
6. K. Gouda, M. Zaki. Efficiently Mining Maximal Frequent Itemsets. In *1st IEEE International Conference on Data Mining*, November 2001.
7. R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM-SIGMOD Intl. Conf. on Management of Data*, June 1998.
8. D. Burdick, M. Calimlim, J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Intl. Conf. on Data Engineering*, April, 2001.
9. Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal. Mining Frequent Patterns with Counting Inference. *SIGKDD Explorations*, December 2000.
10. J. Han, J. Pei, Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the ACM-SIGMOD Intl. Conf. on Management of Data*.
11. N. Pasquier, Y. Bastide. Discovering Frequent Closed Itemsets for Association Rules. In *Proc. of the 7th Intl. Conf. on Database Theory*, January 1999.
12. Z. Zheng, R. Kohavi, L. Mason. Real World Performance of Association Rule Algorithms. In *Proc of Intl. Conf. on Knowledge Discovery and Data Mining*, 2001.

Implementation and Comparative Evaluation of Maintenance Policies in a Data Warehouse Environment

Henrik Engström[1], Sharma Chakravarthy[2], and Brian Lings[3]

[1] Department of Computer Science, University of Skövde, Sweden
henrik@ida.his.se
[2] Computer Science and Engineering Department, University of Texas at Arlington
sharma@cse.uta.edu
[3] Department of Computer Science, University of Exeter, UK
B.J.Lings@exeter.ac.uk

Abstract Data warehouse maintenance is the task of updating a materialised view to reflect changes to autonomous, heterogeneous and distributed sources. Selection of a maintenance policy has been shown to depend on source and view properties, and on the user specified criteria (such as staleness, response time etc.), which are mapped on to evaluation criteria. In our previous work, we have analysed source and view characteristics, and user requirements to derive a cost-model. Maintenance policy selection has thus been cast as an optimisation problem.

This paper takes a complementary approach to evaluating maintenance policies, by implementing a test-bed which allows us to vary source characteristics and wrapper location. The test-bed is instrumented to allow costs associated with a policy to be measured. An actual DBMS (InterBase) has been used as a relational source and an XML web server has been used as a non-relational source. The experiments clearly show that maintenance policy performance can be highly sensitive to source capabilities, which can therefore significantly affect policy selection. They have further substantiated some of the conjectures found in the literature. Some of the lessons learnt from this test-bed implementation and evaluation are reviewed.

1 Introduction

Information is becoming an increasingly important corporate asset, with consequential requirements for its efficient collection, organisation and maintenance. For decision-making and other analytical tasks, it is often necessary to integrate data from a number of sources. These sources may be operational databases controlled by the organisation that conducts the analysis, but may also involve external sources. The latter may include web sites and read-only databases. A data warehouse (DW) can be used to support selective integration and analysis of distributed information. At a very general level, a DW can be seen as a set of materialised views over heterogeneous, autonomous, and distributed sources [15,10]. A consequence of materialisation is that a DW may need to be maintained when sources are updated.

A DW maintenance policy determines when (e.g. periodically or immediately) and how (e.g. incrementally or by recompute) DW views should be updated to reflect

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 90–102, 2002.

changes in sources. Heterogeneity implies that no specific data model can be assumed, and that an interface can have a varying degree of support for maintenance. Autonomy implies that sources cannot be altered to participate in maintenance. DW maintenance is inherently dependent on the capabilities provided by the sources. This is obviously the case for sources external to an organisation, but may also be true for sources controlled by the organisation, due to limitations related to security or legacy issues.

There have been several previous approaches to DW maintenance [1,20,17] which are based on assumptions that sources either provide the required capabilities or can be extended to do so through wrapping. However, wrapping to extend source capabilities will introduce overhead, which may impact on system performance and user quality of service (QoS). The selection of a maintenance policy, based on QoS, system performance or both, may hence be affected by source capabilities. In [5] we introduced user-oriented definitions for QoS, and developed a set of mutually independent criteria for characterising source capabilities. These have been used to produce a number of meaningful maintenance policies. They have been further analysed in [7], which introduces a cost-model to explore optimal policies (for a single source) for a given combination of QoS requirements and source capabilities.

In this paper, we describe a test-bed that has been developed to support various source and DW characteristics, as well as DW maintenance policies. We present results from experiments to compare the performance of different policies. An important result is that optimal policy can be strongly dependent on source capabilities. This can be shown independently of whether optimisation is based on minimising system overhead or meeting requirements on QoS. This is largely ignored in the published literature

The paper is organised as follows. In the next section we give a short introduction to previous work on DW maintenance and our approach to it. In section three we present the test-bed architecture, implementation, and the features supported. In section four we present a number of experiments designed to explore how source capabilities affect maintenance, and analyse the results. Finally, in section five we suggest future work and add some final conclusions.

2 Background

A number of studies [1,19,12,17,20] address DW maintenance, i.e. how to update materialised views based on heterogeneous, autonomous sources. DW research is strongly focused on incremental maintenance in a relational context where sources may provide delta changes, although there exist approaches based on assumptions about different data models and application scenarios [2,8,18,13]. Much interest has been shown in view consistency [20,1] and the development of algorithms which preserve various degrees of consistency.

2.1 Our Approach

It is common to consider recompute policies as unrealistic, mainly due to an assumption that the size of a view is likely to be very large. Sources which do not provide deltas are assumed to be extended through wrapping. We claim that, depending on the scenario,

this may lead to solutions which are inferior to recomputing. Based on our observations, the approach adopted in this paper is to include wrapper activity (i.e. cost) in any comparative analysis of maintenance policies. To the best of our knowledge, this has not been done previously. We have:

- Developed a test-bed which includes: a source that can be configured to have any combination of source capabilities; a wrapper that maintains a view according to any of the policies; and has instrumentation to measure performance in terms of stated evaluation criteria.
- Executed a number of experiments in which policies and capabilities are varied, their computation time recorded, and the results analysed.

This paper considers the representative subset of maintenance policies shown in Table 1. These policies are the result of combining incremental and recompute with immediate, periodic, and on-demand timing - the most commonly mentioned timings in the literature.

Table 1. A representative set of maintenance policies

Policy	Abbreviation	Description
Immediate incremental	I_I	On change → find and send changes
Immediate recompute	I_R	On change → recompute
Periodic incremental	P_I	Periodically → find and send changes
Periodic recompute	P_R	Periodically → recompute
On-demand incremental	O_I	When queried → find and send changes
On-demand recompute	O_R	When queried → recompute

Among these, P_I and P_R are parameterised on maintenance frequency p. Exactly how these policies are implemented depends on the nature of the view and on source capabilities. In the literature, there are examples of both system-oriented and user quality oriented evaluation criteria, and combinations thereof. System oriented criteria include processing (CPU and I/O), storage requirements (in source as well as DW), and communication overhead. QoS criteria [5] include consistency, view staleness and response time. In this paper, we consider strong consistency [20], maximal staleness [7] and average response time.

3 Test-Bed Implementation and Its Features

To support various source capabilities and analyse how the presence or absence of these capabilities affects the selection of maintenance policy, we have developed a test-bed in which a view can be maintained with any of the policies described in section 2, and with various combinations of source capabilities. The purpose of implementing an extensive test-bed is to study DW maintenance policies to understand the impact of source capabilities and wrapper localisation.

3.1 System Design

Fig. 1 shows the major components of the system. Full details of the platform are available in [6].

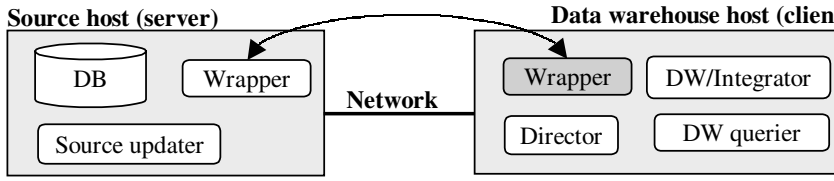


Figure 1. Major components of the test-bed

The system has been developed as a distributed Java application on two Sun Ultra Sparc machines under Solaris, communicating via a dedicated hub. Maintenance is handled by a configurable wrapper, which can be located either in the server or in the client. Care has been taken to maintain independence of the various processes, to eliminate synchronisation effects.

DB. The source is modelled as a relation in an Interbase 6.0 database, initialised for each experiment to give appropriate view selectivity and update behaviour. Tuples selected or updated will be randomly distributed through the relation. Where source capabilities are not provided directly, the test-bed emulates them at no (measurable) cost.

Source Updater. During an experiment the updater executes Poisson distributed updates, deletes and inserts. The size of the affected tuple set is normally distributed around a specified value. This process operates independently from other processes in the system.

Wrapper. The wrapper is responsible for implementing maintenance policies. In our case this means either recomputing a view or incrementally maintaining it, at times dictated by the policy. Immediate policies are triggered from the source, on-demand policies from the querier, and periodic policies through a local timer. The wrapper process can be run either on the server or on the client.

DW/Integrator. The task of the integrator is to maintain the warehouse view and respond to queries. In the developed environment the warehouse is in practice only an integrator.

Querier. The warehouse querier is an independent process which issues queries to the warehouse. As with the updater, events are Poisson distributed.

Director. This is a central component for initialising the data needed for experiments and for collecting and recording measurements. During experiments three different metrics are collected: staleness (Z) and response time (RT) for each query, and integrated cost (IC) - the sum of source processing, warehouse processing and communication. Staleness is computed for individual queries based on: the time when queries are returned; which update is reflected in the query result; and when the following update is committed. Details of how staleness is defined can be found in [7].

We also collect additional information that can be useful in analysing experiments, for example to detect errors or to gain an indication of the reliability of the results.

3.2 Support for Source Capabilities

When experiments are executed, it should be possible to configure the source to have any combination of source capabilities. It is important to differentiate between a source having a capability and the wrapper compensating for it. In our system, the distinction is not inherently sharp as we control all components of the system. In a real system, with an autonomous source, the distinction will be clear: either the source provides a capability which can be used, or the wrapper has to operate without it. In this section we give a brief introduction to the source capabilities considered. We discuss how each of them has been realised in the test-bed, and how the wrapper will act when it is not making use of each.

A central activity in maintenance is the detection and representation of changes. In [5], three orthogonal change detection capabilities are defined, as shown in Table 2.

Table 2. Classification of change detection capabilities

CHAW	Change aware - the ability to, on request, tell when last change was made
CHAC	Change active - the ability to automatically detect and report that changes have been made
DAW	Delta aware - the ability to deliver delta changes

It is apparent that all of these capabilities have an impact on maintenance. For example, CHAW may be used to avoid unnecessary recomputation of a view; CHAC is necessary for immediate notification of updates, and DAW will deliver actual changes.

Another important property of a data source is the query interface available to clients. A source is defined as *view aware* if "*it has a query interface that returns the desired set of data through a single query*" [7], where the desired set of data is the subset of the source used in the view. With a non view-aware source, the whole data set may have to be retrieved and processed to derive a view.

If a source lacks a certain capability, it is sometimes possible to extend it by wrapping. As an example, it is possible to derive changes even when a source is not DAW,

by sorting and comparing two snap-shots[1] [14,3]. However, the autonomy of sources implies that there is a limitation on which mechanisms can be used to extend a source. Distribution may also imply that wrapping has to be done remotely. Engström et al. [5] define a source to be "remote" if it cannot be wrapped on the server side. In other words, remote is synonymous with client-side wrapping.

View Aware -VAW. The important implication of a source being non-VAW is that a larger fraction of it will have to be retrieved, in general. Interbase is VAW for views expressed in SQL. When the wrapper utilises VAW the desired fraction of a relation will be retrieved through a select query. Otherwise the full relation will be sent to the wrapper, which will iterate through the relation to compute the view.

Delta Aware - DAW. We utilise Interbase triggers functionality to provide DAW capability. Triggers are defined to insert source table changes into an auxiliary table. When a source is specified as non-DAW, the wrapper saves a sorted snapshot of the view which is compared with a recently sorted view when changes are to be derived.

Change Active - CHAC. An important remark concerning DAW is that a source is only required to deliver changes *on request*. For I_I and I_R sources are required to immediately notify clients about changes. It is interesting to note that, although there has been extensive research on active database technology, it is very rare or even impossible to achieve true CHAC from commercial DBMSs. Many systems support internal triggers, which may initiate an action when the database is updated, but that is not the same as being able to make external notifications on transaction commit. In the test-bed the source updater notifies the wrapper after each commit. This emulates CHAC capability in a way we claim is acceptable for our purposes. We carefully design the message passing to avoid blocking, and we monitor the communication overhead it introduces.

It is not possible for the wrapper to compensate for lack of CHAC capability: polling for changes will inevitably introduce delays, which may lead to missed updates. It also introduces the problem of setting an appropriate periodicity.

Change Aware - CHAW. CHAW capability is useful in maintenance to avoid unnecessary computation. The wrapper uses it as a first step to check whether changes have occurred. Although many data sources have the CHAW capability (for example, files and web-pages) it is, interestingly, not provided in standard SQL. There is no way, that we are aware of, to query the server on the time a relation was last updated. To emulate CHAW, the wrapper uses notifications from the updater which are sent to emulate CHAC.

The wrapper in our system will not try to compensate for lack of CHAW capability; the cost would be on the same order of magnitude as maintaining the view.

[1] Typically, by performing the difference operation. It should be noted that this is an expensive operation in most situations.

Wrapper Location. Wrapper localisation is determined when an experiment is initialised. We use Java RMI to migrate the wrapper to the source environment. From there it will interact with the integrator through RMI. If wrapping is to be done in the client, the wrapper operates as a thread in the warehouse environment.

3.3 Summary

In this section we have presented an implemented test-bed which enables comparison of DW maintenance policies. The cost of maintenance in terms of system overhead (IC) and QoS (staleness and response time) is recorded, and a number of parameters can be controlled, including: source and view size; update size and distribution; update and query frequencies; source capabilities; wrapper localisation; and maintenance policy (including periodicity for periodic policies). We have focused the presentation on source capabilities and wrapper operation as these are of particular interest in this paper. We acknowledge a number practical limitations in the test-bed as configured. Firstly, we are only interested in delays introduced up to the point at which data enters the warehouse. The data warehouse itself has been simulated in main memory. Secondly, relation size has been kept modest. In spite of this the total elapsed time for experiments has been over six weeks. Thirdly, alternative vendor specific replication facilities have not been explored.

4 Experiments and their Analyses

The test-bed presented in the previous section has been used to run a large number of experiments. Different parameters have been varied to enable a systematic evaluation and comparison of policies. In this section we will present the results and make some interesting observations.

Throughout this section we will use the following abbreviations for experiment parameters:

- p refers to periodic maintenance frequency (1/period);
- c refers to update frequency;
- q refers to query frequency.

In all experiments the tuple size was set to 1K and the size of the relation to average 30000 tuples. View selectivity is 20%, giving an average view size of 6Mb. Update transactions were, on average, 100 tuples with 25% inserts, 25% deletes and 50% updates. For each experiment measures are collected for 150 queries. The same experiment is repeated three times with different seeds used to initialise random number generation. The median values for the three experiments have been used for analysis. The first 10% of queries have been excluded from measurements as they are considered to be in the initialisation time.

The source table has the following attributes: the primary key (PK); a View Attribute (VA), used to select which rows belong to the DW view; a Guide Attribute (GA) column, which directs the course of the update stream; and a content field, which dictates the size of a tuple. The only index defined is for PK.

Before an experiment starts, the relation is populated through insert statements. The values for VA and GA are randomly generated to give the expected sizes for the view and for updates. Random character data is generated for the content field.

The results from each experiment are stored in a separate file; a set of experiments can be analysed by filtering out the relevant data from the associated files.

4.1 Description of Experiments

We initially concentrated on establishing the soundness of the system, by comparing policy behaviour with results reported in the literature. Once this had been established we focused on examining the role of source capabilities. The following is a classification of the major types of experiment that have been conducted:

- Analysing the impact of DAW and CHAW. For incremental policies all four possible combinations of DAW and CHAW have been used. They are all meaningful, in the sense that CHAW is used to detect whether maintenance is necessary and DAW is used to derive the delta once maintenance has been initiated. For recompute policies, only CHAW has been varied. It is not meaningful to alter DAW as recompute policies do not make use of deltas even if they are provided.
- Analysing the impact of VAW. To do this we have compared all policies, both for a source that is VAW and one that is not.
- Localisation of wrapper. The set of experiments described above have all been executed with the wrapper located in the server and in the client.

For each of the above alternatives, various values for parameters p, c and q have been used. We have also conducted experiments where the source, view and update sizes have been varied. Finally, the role of the machines has been switched to make sure that the results are not dependent on the choice of machines for DW and sources.

No experiments have been designed to explore the impact of CHAC. The reason for this is that the wrapper cannot compensate for the absence of CHAC. In other words, this means that the results for I_I and I_R assume CHAC. If a source is not CHAC, I_I and I_R can simply not be used.

4.2 Results for a Relational Source

Although we have conducted a large number of experiments, we report only on the salient results. The reader is referred to [6] for additional details.

Initial experiments were conducted to verify that the behaviour of policies in terms of costs was as expected from the literature. Fig. 2 shows a comparison of the integrated cost (IC) for policies I_R and O_R when update frequency is altered, q is 0.02 and the source is not CHAW.

As can be seen, the cost for O_R is independent of update frequency. On the other hand, I_R increases linearly with update frequency. This is as expected for a source which is not CHAW, as each change requires a full recompute of the view. This means that the choice between I_R and O_R is dependent on the relation between c and q. I_R is cheaper for $c < q$; O_R is cheaper for $c > q$.

Through such experiments we have been able to substantiate a number of results from the literature. In the rest of this section we present results which, we claim, contribute to an understanding of DW maintenance in general, and particularly its dependence on source capabilities.

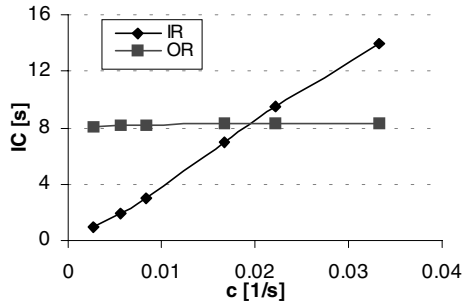


Figure 2. Integrated cost as a function of update frequency

All Policies Are Meaningful. A rudimentary result established is that all suggested policies are meaningful, in the sense that there are situations in which each one is optimal. As has been shown in the literature [11], on-demand maintenance is preferable if queries are infrequent; immediate is preferable if updates are infrequent. For periodic policies, cost (IC and staleness) depends on periodicity.

The choice between incremental and recompute depends on the relative size of updates, something well known from the literature [16,9,4]. In addition, it depends on source capabilities. If wrapping is done in the client and the source is non-DAW, recompute has a lower cost than incremental.

Source Capabilities Impact on Policy Selection. The experiments give strong evidence that all the suggested source capabilities may influence policy performance. This is true for both QoS and system costs. We have been able to verify that all capabilities, and wrapper localisation, may impact on policy selection.Table 3 shows the IC for policies when CHAW, DAW, and wrapper localisation is varied.

By comparing rows whose source capabilities differ in only one place, it is possible to conclude that each capability impacts on policy selection. For example, row 3 has O_R as optimal policy while row 7 has O_I, which implies that wrapper localisation affects policy selection. In the same way we see that row 3 and row 4 have different optimal policies, implying CHAW impacts on selection. Finally, rows 2 and 4 have different optimal policies, implying DAW has an impact.

Concerning VAW, we have identified situations where it impacts on policy selection - but only for very specific combinations of source characteristics and parameter settings. This indicates that, among the suggested policies, VAW has the least impact

Table 3. The IC per query (in seconds) when p=c=q

	Wrapper	CHAW	DAW	I_I	I_R	P_I	P_R	O_I	O_R
1	Client	True	True	1.6	9.5	0.4	5.1	0.4	4.0
2	Client	False	True	1.6	9.6	0.4	8.2	0.6	8.3
3	Client	True	False	10.7	9.5	6.4	5.1	4.6	4.0
4	Client	False	False	10.7	9.6	9.8	8.2	9.6	8.3
5	Server	True	True	1.6	12.6	0.5	7.4	0.5	5.4
6	Server	False	True	1.6	12.6	0.5	11.3	0.6	11.2
7	Server	True	False	9.6	12.6	5.7	7.4	4.0	5.4
8	Server	False	False	9.6	12.6	8.4	11.3	8.4	11.2

on *relative* policy cost. It is important to note, however, that the absolute costs differ significantly for policies when a source is VAW compared to when it is not.

Returning to Table 3, an interesting observation is that DAW is critical for incremental policies to be superior when the source is wrapped in the client. As an example, IC is significantly higher for a source without DAW than with. More importantly, it is higher than the corresponding recompute policy. This is an interesting result, as it contradicts a well established rule-of-thumb that incremental policies are always more efficient than recompute policies unless the size of the delta is very large.

Staleness. If policies need to be chosen based on staleness, the situation is quite different. For this requirement, periodic policies are penalised for not being synchronised with either updates or queries. Also, the setting of periodicity becomes crucial. As an example, Fig. 3 a shows the combined cost of staleness and IC as a function of p, for P_I and P_R with a source that is DAW but not CHAW.

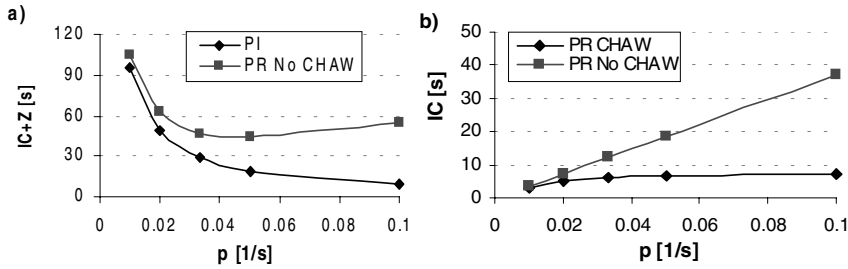


Figure 3. a) The combined cost of IC and staleness for periodic policies as a function of periodicity. b) The impact of CHAW on the IC for P_R when p is varied

The shapes of these curves are typical for this situation; when maintenance is infrequent, staleness is high, but with frequent maintenance IC grows. For P_R, IC grows

faster than staleness declines, giving the U-shape. If the goal is to minimise Z+IC, it may be hard to find the p-value which gives a local minimum for P_R. An interesting observation is that CHAW makes this setting much less critical. Figure 3b shows IC for periodic recompute as a function of p with and without CHAW. The graph shows only a slight increase in cost when the source is CHAW, while the gradient is steeper when the source is not CHAW. CHAW makes it possible to reduce the cost of frequent polling, which may be needed to reduce staleness. For the combined IC+Z cost in Figure 3a, the presence of CHAW will give P_R a shape similar to P_I (but with a higher value).

Response Time. The major difference between policies in response time is that O_I and O_R have substantially higher response time than other policies under the same conditions. This is as expected, as the delay to conduct maintenance is included in the response time for those policies [4]. It is interesting to note that the presence of CHAW reduces response time for O_I and O_R.

The Impact of Policy Timings. In the process of conducting experiments we made an interesting observation. When p=c=q the cost of periodic, immediate and on-demand policies is not, as expected, identical. For most experiments immediate has the highest IC. This may be explained by the fact that immediate is penalised for the consequential synchronisation with updates: the database does not reach an idle state before it is queried for the deltas.

We find this issue worth raising as it brings in a new dimension to maintenance which is rarely considered. Depending on the staleness requirement, it may be preferable to avoid immediate maintenance.

4.3 Result for XML Source

An important property of a DW system is that sources are heterogeneous, which means that not only relational sources and views can be assumed. To better understand the impact source system characteristics have on maintenance we have extended the implementation to include a web-source which delivers XML-pages. Each XML object is stored in a separate file. Through HTTP-requests, clients can request an XML-document containing a set of objects.

This source has been used to conduct some of the experiments described in the previous section. The source data had 3000 XML objects each with a size of 5K. The view involves 10% of these objects. Preliminary results show a close resemblance with the relational source in terms of relative policy performance. As an example, Table 4 shows IC for all policies for a selection of parameter settings.

The lowest cost for each setting (row) is marked with bold typeface and as can be seen all policies are marked once. Furthermore, with client wrapping and a non-DAW source, recompute was again found to be the best policy.

4.4 Desiderata

The development of the test-bed has given us some valuable insights into the capabilities that may be provided by a DBMS accessed through the standard access protocol JDBC.

Table 4. The IC per query (in seconds) for policies with different configuration

p	c	Wrapper	I_I	I_R	P_I	P_R	O_I	O_R
0.02	0.02	client	4.2	3.9	4.3	3.8	3.7	**3.4**
0.02	0.02	server	3.2	5.0	3.2	4.8	**2.8**	4.3
0.02	0.01	client	2.0	**1.8**	4.3	3.9	3.8	3.4
0.02	0.01	server	**1.6**	2.3	3.2	5.0	2.8	4.3
0.01	0.02	client	4.2	3.9	1.7	**1.4**	3.7	3.4
0.01	0.02	server	3.2	5.0	**1.3**	1.9	2.8	4.4

We have also learnt what is required by a wrapper to extend a source which does not offer the desired capabilities.

It is interesting to note that it was not possible to produce CHAW from the chosen DBMS in a straightforward way. It is a fairly simple service to provide, and has been shown to have significant impact on performance. It was possible to produce DAW by defining an auxiliary table and some triggers. This, however, requires administration privileges which may not be granted by autonomous sources. Finally, to provide CHAC we had to introduce an artificial signalling channel from the updater to the wrapper. This is an acceptable solution in a test-bed, but is not an option in a real environment. It may have been possible to produce external notifications as trigger actions, but not through a JDBC connection and not on transaction commit.

Other source types (other relational DBMSs, web-servers, temporal databases etc.) may provide other capabilities, but it is unlikely that a source will provide all capabilities. The experiments clearly show that, for autonomous sources exporting data to external clients, each suggested capability can impact on performance.

5 Conclusions and Future Work

Although materialised view maintenance has been a research issue for more than 20 years there are still issues which are not well understood, and new circumstances which invalidate rules of thumb established under different conditions. In this paper, a test-bed has been presented which we have used to compare representative maintenance policies in a single source scenario. The test-bed allows us to configure sources to provide any combination of source capabilities. We have identified situations in which the long-held rule that incremental maintenance is more efficient than recompute would have led to reduced system performance as well as lower quality of service. The suggested source capabilities may be useful for a better understanding of the conditions for maintenance. We believe that the test-bed we have developed will be a useful tool to assist system developers in understanding the trade-offs involved in maintenance policy selection.

When conducting the work reported here we have been forced to draw boundaries and limit the search-space for obvious reasons. We see two main directions in which the study could be extended. First, more experiments within the current simulation environment could be conducted. We have selected a number of representative experiments initially, but there are many more experiments which could be usefully conducted.

Secondly, the simulation model could be developed to capture more detailed information or to simulate other scenarios. We are currently extending the system to handle multiple source scenarios where the DW view contains joined data from different sources.

References

1. D. Agrawal, A.E. Abbadi, A.K. Singh, T. Yurek: Efficient View Maintenance at Data Warehouses. *SIGMOD Conference (1997)*
2. M.A. Ali, A.A.A. Fernandes, N.W. Paton: Incremental Maintenance of Materialized OQL Views. *DOLAP (2000)*
3. S.S. Chawathe, H. Garcia-Molina: Meaningful Change Detection in Structured Data. *SIGMOD Conference (1997)*
4. L.S. Colby, A. Kawaguchi, D.F. Lieuwen, I.S. Mumick, K.A. Ross: Supporting Multiple View Maintenance Policies. *SIGMOD Conference (1997)*
5. H. Engström, S. Chakravarthy, B. Lings: A User-centric View of Data Warehouse Maintenance Issues. *17th British National Conference on Databases* (2000)
6. H. Engström, G. Gelati, B. Lings: A Benchmark Comparison of Maintenance Policies in a Data Warehouse Environment. Technical report HS-IDA-TR-01-005 (2001)
7. H. Engström, S. Chakravarthy, B. Lings: A Systematic Approach to Selecting Maintenance Policies in a Data Warehouse Environment. *EDBT (2002)*
8. S. Gatziu, A. Vavouras: Data Warehousing: Concepts and Mechanisms. *Informatik (Zeitschrift der Schweizerischen Informatikorganisationen)* 0:1 (1999)
9. A. Gupta, I.S. Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin* 18(2) (1995)
10. J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge: The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin* 18(2) (1995)
11. E.N. Hanson: A Performance Analysis of View Materialization Strategies. *SIGMOD Conference* (1987)
12. R. Hull, G. Zhou: A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. *SIGMOD Conference* (1996)
13. A. Labrinidis, N. Roussopoulos: WebView materialization. *SIGMOD Conference* (2000)
14. W.J. Labio, H. Garcia-Molina: Efficient Snapshot Differential Algorithms for Data Warehousing. *VLDB* (1996)
15. D. Lomet (editor), J. Widom (editor): Special Issue on Materialized Views and Data Warehousing. *IEEE Data Engineering Bulletin* 18(2) (1995)
16. A. Segev, J. Park: Updating Distributed Materialized Views. *IEEE TKDE* 1(2) (1989)
17. D. Theodoratos, M. Bouzeghoub: Data Currency Quality Satisfaction in the Design of a Data Warehouse. *IJCIS* 10(3) (2001)
18. A. Vavouras, S. Gatziu, K.R. Dittrich: The SIRIUS Approach for Refreshing Data Warehouses Incrementally. *BTW* (1999)
19. G. Zhou, R. Hull, R. King, J.C. Franchitti: Data Integration and Warehousing Using H2O. *IEEE Data Engineering Bulletin* 18(2) (1995)
20. Y. Zhuge: Incremental Maintenance of Consistent Data Warehouses. PhD Thesis Stanford University (1999)

Establishing a Taxonomy of Quality for Use in Information Filtering

Mikhaila Burgess, W. Alex Gray, and Nick Fiddian

Department of Computer Science, Cardiff University, UK
{M.Burgess,W.A.Gray,N.J.Fiddian}@cs.cf.ac.uk

Abstract When searching for information within a distributed heterogeneous environment, it is often difficult to ascertain the quality of the obtained results. Sometimes it may be possible to estimate the quality, but as the amount of available information grows this becomes increasingly difficult and time consuming. One possible solution is to develop a method of using quality as a filter to reduce the amount of irrelevant information that is returned by customising it to a user's requirements, defined in terms of quality characteristics.
Before this can be done the general term 'quality' must be explicitly defined, and its characteristics identified. This paper therefore discusses our research into creating a domain-independent taxonomy of quality that can be used to assist in information evaluation and filtering within various information retrieval environments.

1 Introduction

The primary thrust of our project is to develop a system capable of assisting users in retrieving information of a specified quality from within a distributed heterogeneous environment, such as the Internet. This could be information about tangible products, services, data, or documents from such sources as databases or web pages. As more information becomes available users start to suffer from information overload [15,18,20], where it becomes increasingly difficult, or even impossible, to find the information they need. Their desired information may well be included in the returned results from some database query or Internet search, but sometimes so much irrelevant information is also returned that it becomes impossible to manually search through all of the returned results to identify those that are relevant. It is our belief that this problem can be alleviated by enabling the user to state their desired level of quality, then using this information to automatically filter the results obtained as a result of the user's query. This will return a set of information more suited to the user's requirements.

A method for representing information about quality therefore needs to be defined. Previous projects that have used quality attributes to filter information have only selected those attributes relevant to their particular area, such as in [16] and [10], so that each time a new system is developed a new set of attributes must be collated from scratch. The taxonomy presented in this paper is the first step in describing a general model of quality that can be incorporated into any information retrieval (IR) system. By providing a pre-defined model of quality from which the desired attributes for some domain can be easily selected, these IR systems will be able to improve the relevance of obtained information, thereby also increasing user satisfaction.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 103–113, 2002.

1.1 Potential Users

It is envisaged that many different types of user will benefit from using a model that allows them to state their desired level of information quality. Individuals could employ it to reduce the amount of information returned to them when using an Internet search engine or shopping agent. Their personal quality ratings could be stored within a dynamic personal profile that evolves over time by recording individual quality requirements, learning user preferences and, if required, asking for user feedback regarding the quality of the information they have used. This quality feedback could then also be shared with other users, if desired, such as in [10].

Group profiles are also possible. User communities, such as companies, research groups, or academic departments, may well have common interests when looking for information. In this case it would be desirable to share both user feedback and quality ratings amongst all the group members, as their information needs will be very similar.

1.2 Project Aim

The aim of this project is to create an experimental multi-agent environment to assist users in finding the information they require, of the best available quality, from a set of distributed data sources. By developing a taxonomy of quality that can be incorporated into this, or any other, system we hope to reduce the problem of information overload by using it to increase the precision of the results obtained from an information search. This will be done by providing a formal description of quality, consisting of the various characteristics associated with it, the potential range of values of a characteristic, and descriptions of the settings within which each characteristic is applicable. When complete this model will enable users to apply ratings to these characteristics in order to describe the level of quality they desire from some information.

Our agents will initially focus on the quality of the information available within known data sources, for testing purposes. However, once satisfied that our taxonomy improves the quality of returned results the quality evaluation process can be extended to look at the quality of the data sources themselves (coarse-grained) as well as the data contained within them (fine-grained). The principles developed can then be applied in a larger, unknown environment, such as the Internet.

2 Defining Quality

Quality is not an easily definable term, as it is not absolute. It has many different aspects and its meaning varies across different situations and users [8][1]. A formal method is therefore needed to describe this term, freeing users from the need to produce detailed quality requirements when requesting information, but still allowing them to stress the importance of various desirable characteristics, to define their required level of quality.

2.1 Taxonomy of Quality

During our research we have developed a high level, domain independent, taxonomy of quality, for application within such domains as the Internet and e-commerce. Its development was based upon both people's general perception of quality [13] and their

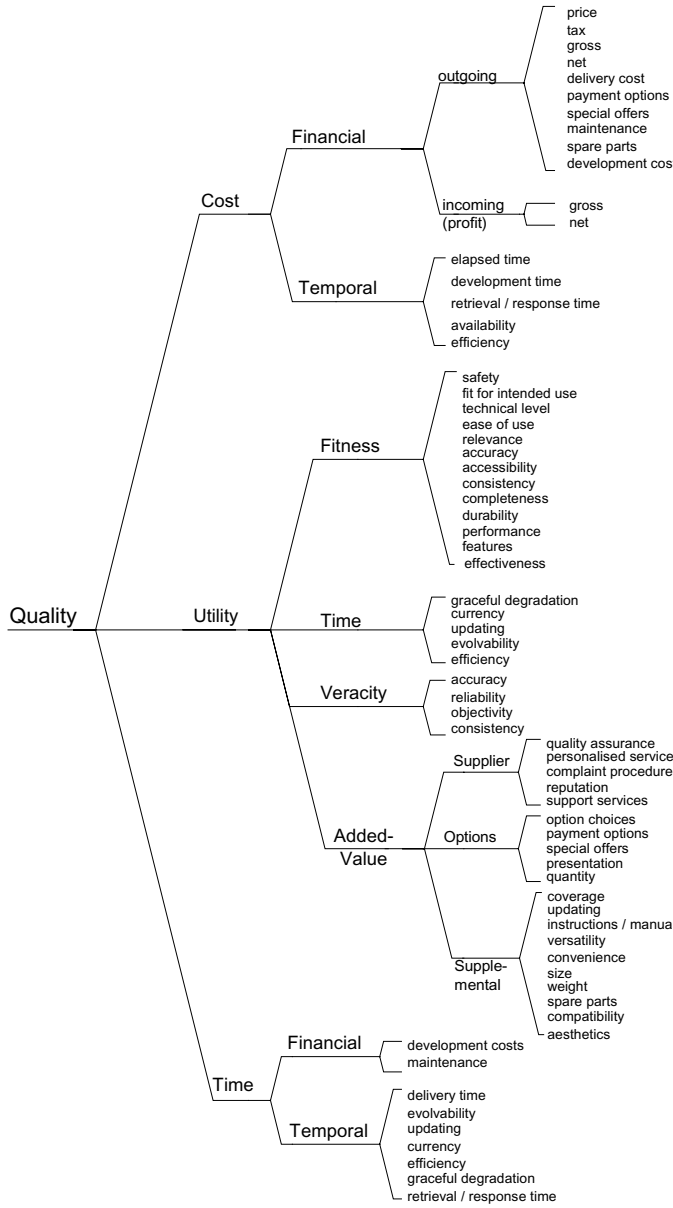
domain specific interpretations, including those from information retrieval [23,24], enterprise modelling [14], software engineering [9], software agents [21], shopping agents [5,12,19,25,28], database research [6,11,22], multi-attribute utility theory (MAUT) and general consumer issues [27]. The terms within our taxonomy were acquired by combining information obtained from technical literature (including the aforementioned references), research into quality, and discussions with potential users.

Within the process of developing or purchasing some product there are three main factors that need to be considered - its em cost, the em time taken to either develop or obtain it, and the item em requirements. We have therefore adopted these elements as the top level of our taxonomy, with one minor change. The term em requirements implies features that are essential within some product, but when discussing quality this is not necessarily the case. Features that are an additional benefit need also to be included to cater for the user who has an idea about the qualities they would like in a product, but which are not necessarily essential. The term em requirements has therefore been replaced by the more general term *utility*. The current version of our taxonomy can be seen in Fig. 1 below. It contains all the currently identified attributes of quality, organised into such groups that the user should be able to find the attributes they desire by following an intuitive path through the hierarchy. This has been done to enable the user to easily find any attribute they require, without needing to search the entire taxonomy.

2.2 Quality Attributes

As each person has different opinions regarding the meaning of the term 'quality' this taxonomy should not be viewed as an exhaustive collection of quality attributes. It is not static. As it is used more attributes will be added, those not relevant will be removed, and the placement of the attributes may be altered. Indeed, it is envisaged that this taxonomy can be restructured when required, with the hierarchy reflecting the importance of each category and attribute to the situation. Thus a user or group of users will develop their own version of the taxonomy tree and make minor adjustments to this if the current requirement needs a different quality filter. These adjustments may involve adding or removing individual attributes, or entire categories, as their need dictates.

As can be seen in Fig. 1, the categories of quality are not discrete: some terms are repeated. This has been allowed as the meaning of these terms differs depending upon the context in which they are used. It also ensures that attributes can be found when following an intuitive path though the taxonomy. For example, the term em financial appears twice in the taxonomy, to represent both monetary (em cost category) and time (em time category) expenditures. As the number of domain-specific attributes added to the taxonomy increases, it will become unmanageably large. This is avoided by the contents of the main taxonomy being restricted to only general quality attributes, with those particular to individual domains being recorded in smaller, domain-specific sub-taxonomies. Therefore, when using these quality metrics, more than one taxonomy may be required - the general taxonomy plus one or more domain-specific taxonomies. Given this taxonomy, we need to know how the characteristics it embodies can be represented within an IR system to allow comparisons and negotiations based on quality ratings.



q

Figure 1. Taxonomy of information quality

2.3 Representing Quality Ratings

2.4 Quantitative and Qualitative Characteristics of Quality

A number of the characteristics that we have discovered are quantitative in nature. This means they can be easily represented within an IR system as their values can be stated as specific quantities, e.g. 'purchase price'. But this is not always the case. Many characteristics are qualitative measures of quality, such as 'reliability' and 'efficiency'. Although difficult to quantify, these characteristics are just as important as the quantitative measures in defining a user's quality requirements. After examining the properties of these qualitative characteristics, and looking at other research projects including [19], [26] and [17], it is our belief that they are all quantifiable as they can be represented as ordered sets of values. From these sets, or scales of values, users can select their desired level of quality for qualitative characteristics, which can then be used within an IR system to compare quality ratings.

2.5 Rating Scales

The scales used for stating desired levels of quality will be based upon that used in MAUT [4], as shown in Fig. 2, part (a). The required level of quality can then be chosen for each characteristic by either selecting a value from the scale via a slider on the user interface, or stating 'no preference'. To make it more intuitive for the user these scales

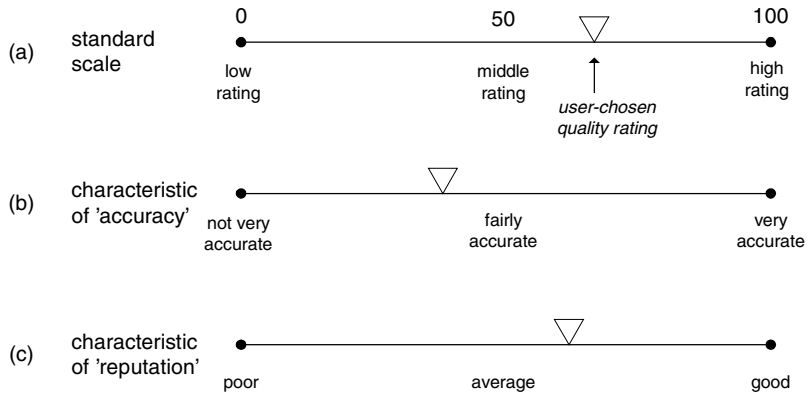


Figure 2. Scales of quality characteristic ratings

may be labelled differently to make selecting the desired level of quality easier, as in Fig. 2 parts (b) and (c). The values used within the IR system for quality comparisons will however remain the same as those in part (a), but this would be hidden from the user for simplicity. Along with the quality rating values for each desired attribute the

user will also be able to state the importance of each rated value. This can be used by a search agent to calculate which attribute values can be relaxed during a search, and by how much. If the user does not provide either a rating or importance value for some attribute then default values may be used. Initially these will be simple values, but as user feedback can be used to adjust these defaults they will become more suitable as the system matures.

2.6 Obtaining Information about Quality

To be able to rate information based upon its quality we must be able to obtain values for the various quality characteristics. These can be acquired in various ways, such as from the information supplier, through user feedback, automatically by means of quality evaluation software, or from independent third parties, such as 'Which' [27] . The latter two would, hopefully, be objective, unbiased and therefore very reliable, but might also be difficult, time consuming and expensive to obtain. A cheaper alternative would therefore focus on feedback from users and suppliers. One possible method would be for the supplier of information to rate its quality as it is placed into the information marketplace. This may well be biased, as the supplier is unlikely to say that the information they provide is poor. However, this can be used along with feedback from users to provide a more accurate evaluation of quality over time, as illustrated in Fig. 3. Initially, few users will have supplied feedback regarding information quality, so

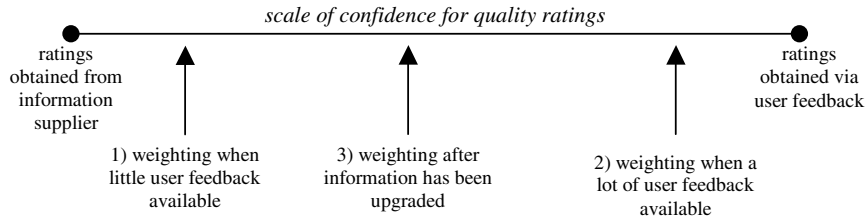


Figure 3. Illustration of weighting confidence in supplier and user quality ratings

supplier provided ratings will be the most reliable (1). Although the supplier is likely to be subjective their ratings should, generally, be more reliable than the opinions of a small number of anonymous users, who may have extremely differing views or biased opinions. This is particularly likely to be the case if the supplier has a good reputation - a quality measure that should also be taken into account in this scenario. Over time the number of users providing feedback will increase, meaning more emphasis can be placed on its reliability (2). Eventually the user supplied ratings will become far more reliable than those of the suppliers. If the supplier then upgrades the supplied information, emphasis will again move towards the supplier provided ratings, until enough users

have rated its quality after the change (3). This system would lead to one value for each quality characteristic, determined by combining the ratings from both supplier and user feedback. Weighted values could be used to add more emphasis to either the supplier or user feedback ratings, depending on the situation and user preference. The precise method used to combine these ratings is an aspect of future research. It is of course possible that some user supplied feedback is not reliable, as it may be unfairly biased or discriminatory. However, as this problem is currently being researched elsewhere, e.g. [3], it will not be included as part of our project.

3 Using the Taxonomy of Quality

Not all of the included characteristics of quality will be applicable to every situation. For example, when looking to purchase a car the characteristic of 'purchase price' is likely to be very important, but less so the characteristic of 'weight'. A personalised quality evaluation process is therefore necessary to ascertain the quality needs of each individual user in their current context.

3.1 User Defined Quality Ratings

Each user should be able to set quality ratings for the characteristics that are important to them in their current situation, without needing to worry about the rest. A method therefore needs to be developed for deciding which characteristics the user is required to rate, as asking them to spend time selecting the ones they want from all those available would be counterproductive, especially as the number of available characteristics increases.

3.2 Setting and Relaxing Quality Ratings

Once a method for choosing characteristics to be rated has been developed, the user can then set their desired level of quality in one of two ways:

1. setting specific required values for each desired characteristic;
2. choosing thresholds within which the quality will be of an acceptable level (either upper boundaries, lower boundaries, or both).

These values will define the users' em solution space - the area that bounds the information that will be considered by the user to be of good quality. Setting specific values, or very tight thresholds, will place a strong restriction on the results that can be returned, so the ability to state the importance of complying with that request should also be included, allowing the system to return results that do not necessarily exactly match the user's initial requirements. This is particularly important if the user's request is unrealistic. This could occur when either the user has entered a sensible request, but one for which no matches exist in the current environment, or when unrealistic quality ratings have been entered for conflicting characteristics. For example, when discussing software it would not be possible to find code that is both minimal and secure as the

addition of security features increases code size. This is a feature of many of the characteristics in this taxonomy, as many of them affect other characteristics: where a high rating for one characteristic may increase (positive) or decrease (negative) the rating for another.

By stating which quality ratings are essential and which are (merely) desirable these ratings can be relaxed during the search, expanding the solution space, to find information that closely matches the user's needs. For example, if a user requests information on some product that is both cheap and immediately available, but nothing can be found, the system can continue to search by relaxing one or more parameters until either some result is found or the relaxation becomes so extreme that it no longer meets the user's requirements.

3.3 User Profiling

When searching for information, users are likely to have similar quality requirements under similar circumstances. Although not part of this research programme, user profiling techniques could be employed to learn these individual requirements over time, developing a personalised quality profile for each user, or user community. Quality requirements will change over time so this profile will be dynamic, constantly learning user preferences and adapting as both the user and situation changes. This quality profile can then assist users when searching for quality information by reducing the number of ratings and threshold values the user needs to supply. As the profile becomes more accurate the user will only need to rate the quality of important characteristics, leaving their profile to automatically supply the rest, making the rating process faster and less intrusive for the user.

The main problem that needs to be addressed in this area is the fact that each user may have many different profiles, e.g. at work, at home, regarding hobbies, etc, and their quality requirements will differ depending on which they are assuming. This therefore needs to be taken into account when using quality profiling within these different settings. This is however less of a problem when generating user community quality profiles as the scenario for the community profile is likely to remain relatively more static.

4 Future Work

Within our research we are currently looking into how the various characteristics of quality can be measured, and how they interact with each other, for example causing conflicts. Once completed, this knowledge, along with the taxonomy itself, will be incorporated into a system to investigate how the inclusion of quality ratings can improve the results obtained when searching for information in some environment. As yet we have not addressed the problem concerning how to obtain information about quality, be that from the supplier, user, or other parties. Once this has been researched more closely we can look at combining weighted quality ratings to generate more accurate measures of quality. Research then needs to be done into how this information can be stored for future use, for example in dynamic user profiles. The taxonomy needs to be developed

further before it can be incorporated into a fully functional IR system. We have not yet started creating domain-specific taxonomies, or looked into how the various characteristics interact and affect each other. A methodology also needs to be defined for enabling each taxonomy to evolve over time. As the taxonomies are used more characteristics will have to be added to accommodate individual needs, as each are used by different people and in different situations. As this is a substantial area of research our project will initially concentrate on developing an experimental multi-agent environment for testing our general taxonomy, developed using one of the existing multi-agent development toolkits, such as ZEUS from BT [29]. Initially this environment will contain a selection of known datasets, for testing purposes. During this preliminary stage quality ratings will be applied to the information within the datasets rather than to the sources themselves. However, this will be expanded as our research progresses to rate the quality of large numbers of unknown datasets.

The agents within this experimental system will be able to negotiate on the user's behalf to find the information that best meets the user's quality needs, relaxing individual or categories of characteristics, or by increasing the solution space dynamically as necessary in the negotiation process.

5 Conclusion

Although quality of information is extremely important to a lot of people, in different areas of work, no-one has previously looked at developing a taxonomy of quality that can be used for evaluating the quality of information across a wide range of applications. Through our research we have developed a domain-independent taxonomy of quality, initially containing over fifty separate attributes. A number of these are quantitative so are straightforward to store and manipulate. Those that are qualitative can also be made quantifiable by representing them as ordered sets of values, from which the required rating can be chosen. It is therefore possible to define parameters by which each characteristic can be measured. After defining the parameters of these quality metrics it will be possible to obtain data regarding the quality of information, from various different sources. This information may be obtained directly from the supplier, from a reliable, objective third party, or automatically though the use of quality evaluation software. It could also be obtained via feedback from users or user communities.

Having created this general taxonomy of quality we can now continue with the development of an associated quality ontology. This will be followed by the creation of a system to assist users in evaluating the quality of information, obtained from various sources within an experimental multi-agent environment, to test its effectiveness at searching for information of user-defined quality. After this initial testing stage, the system will be expanded to search for information within a larger environment, such as the Internet. The meaning of quality will vary between individuals and situations, so only those quality characteristics that are relevant to the user's current requirements need be taken into account. This will result in a personalised quality evaluation process, to increase the amount of relevant results recovered from an information search, reduce information overload and therefore to significantly increase user satisfaction.

Acknowledgements

We would like to thank EPSRC [7] for supporting M. Burgess with a studentship (award number 99302270), and BT [2] for providing additional support.

References

1. Bouch, A., Kuchinsky, A., Bhatti, N: Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service. Proc. Conf. on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands (2000) 297-304
2. BT URL: `http://www.bt.com`
3. Dellarocas, C.: Mechanisms for Coping with the Unfair Ratings and Discriminatory Behaviour in online Reputation Reporting Systems. Proc. 21st Int.'l Conf. on Information Systems (ICIS 2000) (2000) 520-525
4. Dodgson, J., Spackman, M., Pearman, A., Phillips, L.: Multi-Criteria Analysis: A Manual. DETR, Rotherham, UK (2001)
5. Doorenbos, R.B., Etzioni, O., Weld, D.S.: A Scalable Comparison-Shopping Agent for the World Wide Web. Proc. 1st Int. Conf. on Autonomous Agents (Agents'97), Marina del Rey, CA., USA (1997)
6. Engström, H., Chakravarthy, S., Lings, B.: A User-Centric View of Data Warehouse Maintenance Issues. Proc. 17th British Nat. Conf. on Databases (BNCOD 17), Exeter, UK (2000), 68-80
7. EPRSC `http://www.epsrc.ac.uk`
8. Firquin, B.: Quality is Free ... But How Do You Implement Total Quality and Restructure Information Technology At The Same Time? Proc. Conf. of User Services, Cleveland, USA (1992)
9. Gillies, A.C.: Software Quality - Theory and Management. International Thomson Publishing (1996)
10. Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J.: Combining collaborative filtering with personal agents for better recommendations. Proc. 16th Nat. Conf. Artificial Intelligence (AAAI-99) (1999) 439-446
11. Grefen, P.W.P.J, Apers, P.M.G.: Integrity control in relational database systems - An overview. Data and Knowledge Engineering Press, 10 (1993) 187 - 223
12. Guttman, R.H., Maes, P.: Agent-Mediated Integrative Negotiation for Retail Electronic Commerce. Proc. Workshop on Agent Mediated Electronic Trading (AMET'98), Minneapolis/St Paul, USA (1998)
13. Juran, J.M., Godfrey, A.B.: Juran's Quality Handbook. 5th edn., McGraw-Hill (1999)
14. Kim, H.M., Fox, M.S.: Formal Models of Quality and ISO 9000 Compliance: an information systems approach. Proc. 48th Annual Quality Congress, Las Vegas, NV USA (1994)
15. Kotz, D., Gray, R.S.: Mobile Agents and Future of Internet. ACM Operating Systems Review, 33:3 (1999) 7-13
16. Lesser, V.R., Horling, B., Klassner, F., Raja, A., Wagner, T., Zhang, X.: BIG: A Resource-Bounded Information Gathering Agent. Proc. 15th Nat. Conf. Artificial Intelligence (AAAI-98) (1998) 539-546
17. Lesser, V., Horling, B., Raja, A., Zhang, X., Wagner, T.: Resource-Bound Searches on an Information Marketplace. IEEE Internet Computing, 4:2 (2000) 49-58
18. Maes, P.: Agents that Reduce Work and Information Overload. Communications of the ACM, 37:7 (1994) 31 - 40
19. Maes, P., Guttman, R.H., Moukas, A.G.: Agents that Buy and Sell: Transforming Commerce as we Know It. Communications of the ACM, 42:3 (1999) 81 - 91

20. Montebello, M.: Personal Information Retrieval over the WWW. Ph.D. Thesis, Cardiff University, UK (1999)
21. Nwana, H.S., Ndumu, D.T.: An Intro to Agent Technology. Lecture Notes in Artificial Intelligence, Vol. 1198, Springer-Verlag (1997)
22. Redman, T.: Data Quality for the Information Age. Artech House Publishers (1996).
23. Rolker, C., Kramer, R.: Quality Of Service transferred to Information Retrieval - An Adaptive Information Retrieval System. Proc. 8th Int. Conf. Information Knowledge Management, Kansas City, MO USA (1999)
24. Selberg, E., Etzioni, O.: The MetaCrawler Architecture for Resource Aggregation on the Web. IEEE Expert, 12:1 (1997) 8 - 14
25. Tete-a-Tete URL: `http://ecommerce.media.mit.edu/tete-a-tete`
26. Wallnau, K., Hissam, S., Seacord, R.: Building Systems from Commercial Components, Addison Wesley (2001)
27. Which?Online URL: `http://www.which.com`
28. Zacharia, G., Moukas, A., Guttman, R., Maes, P.: An Agent System for Comparative Shopping at the Point of Sale. Technologies for the Information Society: Development & Opportunities. Rogers, J.Y., et al (eds), IOS Press, (1998)
29. ZEUS Agent Development Kit URL: `http://www.labs.bt.com/projects/agents/zeus/`.

Improving the Refined Triggering Graph Method for Active Rules Termination Analysis

Alain Couchot

Laboratoire d'Informatique de Cergy-Pontoise, (LICP), France
couchot-a@wanadoo.fr

Abstract This article extends the Refined Triggering Graph Method for active rules termination analysis. The Refined Triggering Graph Method has been proposed to analyse active rules sets termination, in the context of object oriented databases. The RTG method exploits the notion of triggering formula. A triggering formula is a logic formula binding two rules. This logic formula captures the triggering link between a rule postcondition and a rule precondition. But only DB-independent atoms contained in rules postprecondition can be taken into account by the RTG analysis. We improve the RTG Method, taking into account the DB-dependent atoms contained in rules prepostconditions. To achieve this, we introduce the notion of *descendant* of a triggering formula and the notion of *complex condition of a rule*. The *descendant of a triggering formula* captures the transformations of an object, which satisfies the triggering formula, due to the rules actions. The *complex condition of a rule* captures the triggering link between a rule R and the rules which can trigger R. Many more termination situations can be detected, thanks to our improvement.

1 Introduction

The next generation of databases will be able to react to modifications of the environment, and to update the stored data according to these modifications. This introduction of the reactivity within the databases results from two present trends: on the one hand, the development of rules languages (production rules, active rules, deductive rules), and, on the other hand, the development of the object oriented technologies. Rules languages and object oriented technology can be seen as complementary: on the one hand, the object oriented technology allows to integrate in the same entity the structural and the behavioural aspects, on the other hand, the rules languages allow to describe the reaction of an entity according to its environment.

However, although the rules are intended to facilitate the design and the programming work, writing a rules set actually remains tricky, often devolved upon specialists. Indeed, a rules set is not a structured entity: the global behaviour of a rules set can be hard to predict and to control [1]. In particular, research works have brought to the fore the rules *termination* problem (the rules execution can be infinite in some situations), or the rules *confluence* problem (the same rules do not necessarily yield the same results, depending on the rules execution order). We are here interested in the active databases rules termination. The active rules generally follow the paradigm Event-Condition-Action [16]. The event is an instantaneous fact, which can occur inside or outside the system. The condition is generally a query upon the database. The action is generally a database update.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 114–133, 2002.

In section 2, we present the related work; in section 3, we expose the Refined Triggering Graph Method; in section 4, we develop an example which motivates our proposition; in section 5, we propose the notion of *triggering formula of a complex path*; in section 6, we propose the notion of *descendant of a triggering formula*; in section 7, we introduce the notion of *complex condition of a rule*; in section 8, we expose our termination algorithm; section 9 concludes.

2 Related Work

The active rules termination is an undecidable problem [2]. Research works have proposed sufficient conditions to guarantee the termination of a rules set at run time or at compile-time.

Methods proposed for run-time analysis [5, 9] suffer from the following drawback: it is necessary to know the initial database state, and the result is just available for the tested initial database state.

Methods proposed for compile-time analysis can provide sufficient conditions available for any database state.

We are here interested by compile-time analysis methods. Some methods propose a translation of the active rules in terms of rewriting systems [20], in Condition-Action rules [11], or in deductive rules [13, 18]. However, these methods are rather complex to use, cannot be automated and are reserved for relational databases. [4] introduces a *behavioural stratification* of the active rules. Some conditions allow to conclude that termination is guaranteed if, inside each stratum, termination can be guaranteed. It is necessary to guarantee the termination by means of other methods inside a stratum.

However, most of the works about the active rules termination use the concept of *triggering graph*. [12] introduces, for the first time, the notion of *triggering graph*; this notion is clarified by [1]: this graph is built by means of a syntactic rules analysis; the nodes of the graph are rules, and two rules r_1 and r_2 are bound by an oriented edge from r_1 towards r_2 if the action of r_1 raises a triggering event of r_2. The presence of cycles in such a graph means a non-termination risk for the rules set. The absence of cycles in the triggering graph guarantees the termination. However, this analysis is conservative: the satisfiability of the Condition part is not taken into account. The analysis of the triggering graphs is refined by [3, 5] by means of *activation graphs*. In an *activation graph*, the rules r_1 and r_2 are bound by an oriented edge from r_1 towards r_2 if the action of r_1 can make TRUE the condition of r_2, and a rule *R* is bound to itself if the condition of *R* can still be TRUE after the action execution of *R*. The non-termination implies then the presence of cycles in the triggering graph and in the activation graph. An algorithm for building of the activation graphs is proposed by [10] for the relational databases. However, only self-deactivating rules are taken into account by activation graphs.

The previous technique is widened by [22]: the concepts of *activator* and *deactivator* are proposed. The rule r_1 is a *deactivator* of the rule r_2 if the action of r_1 always makes **FALSE** the condition of the rule r_2. The rule r_1 is an *activator* for the rule r_2 if the action of r_1 can change the truth value of the condition of the rule r_2 from **FALSE** to **TRUE**. A rule *R* is labelled "finite" if each activator of *R* is labelled "finite", and if each cycle containing *R* also contains a deactivator of *R*.

The Refined Triggering Graph Method [21, 28] completes the triggering graph analysis. An edge (r_1, r_2) can be removed from the triggering graph as follows: the post-

condition of r_1 is combined with the condition of r_2 by means of variables unification between the variables used in the action of r_1 and the variables used in the event of r_2. If it is impossible to satisfy the obtained formula (qualified as *triggering formula*), the edge (r_1, r_2) can be removed from the triggering graph. But the variables of the triggering formula must not be updated by rules actions. The Path Removing Technique [23] extends the triggering formulas as follows: if the rule r_1 can indirectly trigger the rule r_2 via a path *P*, a *generalised triggering formula* is built along the path *P*. [23] proposes to "remove a path" instead of a node. To achieve this, a new triggering graph is constituted, equivalent to the initial triggering graph, containing duplicate nodes. The Cycle Unrolling Technique [24] brings another extension. A generalised triggering formula is built along a path, which corresponds to a cycle executed (*k*+1) times: if it is impossible to satisfy the triggering formula, a *k-cycle* is detected. A method is proposed to "unroll" a *k*-cycle in a triggering graph: an equivalent triggering graph is built, containing duplicate nodes.

[29] considers the influence of the composite conjunction events on the termination: two kinds of edge are introduced in the triggering graphs (*total edges* and *partial edges*) in order to take into account the composite conjunction events. [14] improves [29] and [21, 23, 24], taking into account both the overall condition of a rules path and the composite conjunction events, thanks to the notion of *composite path*.

[30] exploits the monotonic bounded operations in order to remove some edges of the triggering graphs: for instance, if a rule action deletes an object of a class, and if no rule action creates objects in this class, the delete operation is bounded and the corresponding rule can just occur a finite number of times. This principle is detailed by [17], which exposes a method to calculate the maximum iterations number of a cycle, due to some database limit values.

[7, 8] uses the abstract interpretation to simulate the execution of the active rules, using abstract database states. A concretisation function allows to link abstract states and real states. If the rules execution terminates for abstract states, then termination is guaranteed for real database states.

[15] allows the termination analysis in a modular design context, thanks to the notions of *private event* and *public event*. So, termination can be guaranteed, even when no designer knows all the active rules.

We present in this paper an improvement of the Refined Triggering Graph Method [21, 28]. The Refined Triggering Graph Method [21, 28], improved by the Path Removing Technique [23] and the Cycle Unrolling Technique [24], detects the deactivation of an overall condition of a rules path, but only atoms which contain no attributes updated by the database (or attributes updated by rules removed from the triggering graph for [23, 24]) are included in triggering formulas. We propose to include also atoms which contain attributes updated by the database (even updated by rules not removed from the triggering graph) in triggering formulas. So, many more termination situations can be detected. To achieve this, we introduce the notion of *descendant of a triggering formula* and the notion of *complex condition of a rule*.

3 The Refined Triggering Graph Method

We present in this section the Refined Triggering Graph Method [21, 28]. We first introduce the abstract architecture of an active object oriented database assumed by [21]. We next present the notions of *triggering formulas* and *generalised triggering formulas*.

3.1 Abstract Architecture of an Active Object Oriented Database

We adopt the abstract architecture of an active object oriented database assumed by [21]. This architecture consists of three components: an object store, events, and active rules. The object store is a collection of objects maintained by the active database. Objects are affected by two types of events: external (explicitly signalled to the system) and internal (corresponding to the database operations).

Each object is represented by a ground object term:

$$o = t_0 : \underline{c}[l_1 := t_1, \ldots, l_n := t_n]$$

t_0 is the unique object identifier (oid), $\underline{c}$ is the class of o, and $l_i := t_i (i = 1, \ldots, n)$ is an attribute-value pair, l_i is an attribute name and $t_i$is a ground term representing the value of l_i for o.

Each event has a recipient object. An event is represented by a ground event term. An event term has the format:

$$e = d(s_0@s_1, \ldots, s_n)$$

where d is the event name, s_0 is the oid of the recipient object of the event, and $s_1, \ldots, s_n (n \geq 0)$ are ground terms representing the event parameters.

Active rules specify how objects respond to event occurrences. The syntax of an active rule is:

Rule:	R
Event:	e
Condition:	c
Action:	$Y_0.l_1 = Z_1, \ldots, Y_0.l_k = Z_k$
Raised events:	$e_1, \ldots, e_k$

where

- *R* is the rule label,
- $e = d(Y_0@\ldots)$ is the event term,
- Y_0 is the oid term of the recipient object,
- c is a logic formula,
- $Y_0.l_1 = Z_1, \ldots, Y_0.l_k = Z_k$ represents the updated recipient object, and
- $e_1, \ldots, e_k (k \geq 0)$ are event terms (raised by the rule action).

The formula c is a conjunction of primitive conditions. A primitive condition can be either an *atom* or a *negated atom*. An atom is specified by means of a first-order language. A required property of active rules is that each variable occurring in the Action part must also occur in the Condition part or in the Event part.

For each rule *R*, a set of event occurrences $E(R)$ is maintained by the rules processing. Before the rules processing, the set $E(R)$ contains the occurrences of the events triggered by the transaction.

The rules processing is the following:

1. ($Choose$). A rule *R*, such that $E(R)$ is not empty, is non-deterministically chosen. An event occurrence *event_occurrence* of $E(R)$ is non-deterministically chosen.
2. ($Match$). The variables of the Event part of *R* (that is the recipient object variable and the parameters variables of the Event part) are instantiated with *event_occurrence*. The event *event_occurrence* is removed from $E(R)$.

3. ($Evaluate$). The condition of *R* is satisfiable if there is a substitution σ such that the formula cσ is **TRUE** for the current database state. σ is then a ground substitution for the variables of the Action part of *R*.
4. (Act). If the condition is satisfiable, the recipient object of the rule is updated. For each rule R' such that the name of the triggering event of R' matches with a name of one of the events raised by the action of *R*, the set $E(R')$ is updated.
5. (*Return to step 1*). If there is at least a rule R'' such that $E(R'')$ is not empty, return to step 1.

The rules processing can clearly be infinite. The termination static analysis studies the rules triggering (step 4 of the processing) and the satisfiability of rules conditions (step 3 of the processing).

3.2 Triggering Formula and Generalised Triggering Formula

We first present the notion of *triggering graph*, which is the basis of the termination analysis. We introduce then the notions of rule *precondition* and rule *postcondition*. This allows us then to present the notion of *triggering formula*.

Triggering Graph. Let RS be an arbitrary active rule set. The *triggering graph* is a directed graph where each node corresponds to a rule $Ri \in RS$. A directed arc $(R_2 \leftarrow R_1)$ belongs to the triggering graph iff the action of rule R_1 generates an event which triggers rule R_2.

If there is no cycle in the triggering graph, termination of the rules set is guaranteed. When a cycle (or several) appears in the triggering graph, the RTG Method proposes to analyse the edges of the cycle, using the *triggering formulas*.

Precondition and Postcondition of a Rule. Let *R* be a rule specified as below:

Rule: *R*
Event: $d(Y_0@Y_1, \ldots, Y_m)$
Condition: $c_1, \ldots, c_p$
Action: $Y_0.l_1 = Z_1, \ldots, Y_0.l_k = Z_k$
Raised events: $d'(X_0@X_1, \ldots, X_n) \ldots$

We set:

$Condition\_Set(R) = \{c_1, \ldots, c_p\}$
$Pre\_Set(R) = Condition\_Set(R)$
$Post\_Set(R) = Condition\_Set(R) \cup \{Y_0.l_j = Z_j / j = 1, \ldots, k\}$

The precondition and the postcondition of the rule are defined as follows:

$Precondition(R) = \wedge_{(c \in Pre\_SubSet(R))}(c)$
$Postcondition(R) = \wedge_{(c \in Post\_SubSet(R))}(c)$

where $Pre\_SubSet(R)$ and $Post\_SubSet(R)$ are respectively subsets of $Pre\_Set(R)$ and $Post\_Set(R)$ determined by a selection criterion described below.

Triggering Formula. Let $(R \leftarrow R')$ be an arc of the triggering graph from R' to *R*. A triggering formula $Triggering\_Formula(R \leftarrow R')$ is built as follows:

$Triggering\_Formula(R \leftarrow R') =$
$Precondition(R) \wedge Postcondition(R') \wedge ParameterFormula(R \leftarrow R')$

where:

- $ParameterFormula(R \leftarrow R')$ captures the parameter transfer from R' to *R* in an arbitrary execution sequence where the execution of the action of R' generates an event which triggers *R*,
- $Precondition(R)$ and $Postcondition(R')$ are determined using the following selection criterion: $Precondition(R)$ and $Postcondition(R')$ just contain primitive DB-independent conditions (an atom is said to be $DB - independent$ iff it does not involve function and predicate symbols whose interpretation depends on the current database state).

If the logic formula $Triggering\_Formula(R \leftarrow R')$ is not satisfiable (that is: can never be **TRUE**), the edge $(R \leftarrow R')$ can be removed from the triggering graph. DB-dependent conditions cannot be included in triggering formulas, since intermediate rules can modify the variables included in DB-dependent conditions between the action of R' and the condition evaluation of *R*.

Equality atoms $(X.l = t)$ (where X is an object variable) are called $selectors$. A DB-dependent selector $(X.l = t)$ of $Post\_Set(R')$ (respectively $Pre\_Set(R)$) can be included in $Postcondition(R')$ (respectively in $Precondition(R)$) if the selector is a *rules-independent selector*.

- If t is a variable, the selector $(X.l = t)$ is a *rules-independent selector* if no rule action contains an atom $(X_0.l_0 = t_0)$ such that $(l = l_0)$;
- If t is a constant, the selector $(X.l = t)$ is a *rules-independent selector* if no rule action contains an atom $(X_0.l_0 = t_0)$ such that $(l = l_0)$ and $(t \not= t_0)$.

Generalised Triggering Formula. The triggering formulas can be generalised for a path containing n rules. If it is impossible to satisfy the generalised triggering formula along the path $Path$, $Path$ cannot occur. But no arc of $Path$ can be removed from the triggering graph if some arcs of $Path$ participate in other cycles. The Removing Path Technique [23] allows to remove a path from a triggering graph. The Cycle Unrolling Technique [24] allows to remove from the graph a path corresponding to a cycle executed k times. These two techniques modify the triggering graph: some duplicate nodes are introduced and some edges can sometimes be removed.

Overview of the Proposed Extension. We propose an extension of the RTG Method. This extension allows to build triggering formulas along rules paths, including rules-dependent selectors. To achieve this, we introduce the notion of *complex rule condition*. A complex condition of a rule *R* includes information about postconditions of the rules which can trigger *R*. This information takes into account the $descendants$ of the rules-dependent selectors included in postconditions: a descendant of a selector $Selector$ is a logic formula capturing the possible transformations of objects which satisfy $Selector$. An iterative computation of the complex conditions is proposed, using the previously computed complex conditions.

4 Motivating Example

We present here an example, which motivates our proposition. This example will be used to illustrate the introduced notions within the paper. We consider a banking application. Four active rules are defined.

4.1 Example

$RuleR_1$: When the allowed overdraft of an account is updated, if the loan rate of the account is equal to 8, the loan capacity of the account is set to 6000. $RuleR_2$: When the loan capacity of an account is updated, if the loan rate of the account is equal to 4, the allowed overdraft of the account is set to 20. $RuleR_3$: When the loan capacity of an account is updated, if the account is a stocks account, the loan rate of the account is set to 4. $RuleR_4$: When the allowed overdraft of an account is updated, if the account is a standard account, the loan rate of the account is set to 8.

Rule: R_1
Event: account_ overdraft_update_event(A_1)
Condition: A_1.rate = 8
Action: A_1.capacity = 6000
Raised event: account_capacity_ update_event(A_1)

Rule: R_2
Event: account_capacity_ update_event(A_2)
Condition: A_2.rate = 4
Action: A_2.overdraft=20
Raised event: account_overdraft_ update_event(A_2)

Rule: R_3
Event: account_capacity_update_event(A_3)
Condition: A_3.type = stocks_account
Action: A_3.rate = 4
Raised event: account_rate_ update_event(A_3)

Rule: R_4
Event: account_overdraft_update_event (A_4)
Condition: A_4.type = standard_account
Action: A_4.rate = 8
Raised event: account_rate _update_event(A_4)

4.2 Termination Analysis of the Example

We try now to analyse the termination of the motivating example, using the main algorithms proposed in the literature.

RTG Method. Let us try to analyse termination of this rules set by means of the RTG method [21, 28]. The triggering formula $Triggering\_Formula(R_1 \leftarrow R_2)$ is:
$Triggering\_Formula(R_1 \leftarrow R_2) = (A_2.\text{overdraft} = 20) \wedge (A_2 = A_1)$
The atoms (A_2.rate = 4) and (A_1.rate = 8) cannot be included in the triggering formula, since they are rules-dependent atoms. Thus, the RTG method cannot guarantee the termination of this rules set.

Activation Graph. Let us try to use [3] to analyse the termination of this rules set. The triggering graph is depicted by Figure 1. Let us build the activation graph. No rule

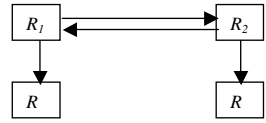


Figure 1. Triggering graph of the motivating example.

is a self-deactivating rule, since the condition of each rule can still be true after the execution of the action of the rule. So there is an edge from each rule to itself in the activation graph. There is also an edge from the rule R_3 to the rule R_2 and an edge from the rule R_4 to the rule R_1. No further information is given by the activation graph (Figure 2). Termination cannot be guaranteed by [3].

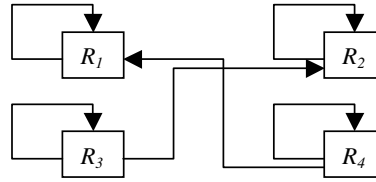


Figure 2. Activation graph of the motivating example.

Unrolling Cycle. Let us try now to use the Unrolling Cycle method [24], and let us analyze the cycle $(R_1, R_2, R_1, R_2, R_1)$. We try to build a formula concerning the cycle $(R_1, R_2, R_1, R_2, R_1)$ from the conditions of the rules of the cycle. The algorithm [24] can include in the formula atoms containing attributes which are not updated by rules, or atoms containing attributes which are only updated by rules removed from the triggering graph (that is: rules previously removed by the algorithm). The conditions of the rules R_1 and R_2 contain attributes which can be updated by rules of the triggering graph (the rules R_3 and R_4, which are not removed from the triggering graph), so no atom can be included in the formula, and the algorithm stops. [24] cannot guarantee the termination of this example.

Abstract Interpretation. Let us try now to use the abstract interpretation [7, 8]. The principle of this method is to simulate the rules processing using an abstract database (an abstract database approximates a number of real databases), and to test the satisfiability of the rules conditions during the rules processing thanks to the information

about the abstract database. The use of this method requires that we can list all the initially triggered rules before the rules processing. But, in this case, the transaction can trigger an arbitrary number of occurrences of each rule before the rules processing (for example, the transaction can trigger two occurrences of R_1 and three occurrences of R_2 before the rules processing). Thus, it is impossible to list the rules initially triggered by the transaction before the rules processing. Instance-level triggers are considered in a later paper [9] but the abstract interpretation approach still requires an enumeration of each kind of initial triggering event. The abstract interpretation [7, 8, 9] cannot guarantee the termination of the example.

4.3 Discussion

Let us analyse in more detail the behaviour of this rules set. If the rate of an account is set to 4 by the rule R_3 , this account is a stocks account (since the condition of R_3 has been evaluated to **TRUE** before the action of R_3). In the same way, if the rate of an account is set to 8 by the rule R_4 , this account is a standard account. No rule action can update the type of an account. So, if the rate of an account is set to 4, this rate can no more be set to 8 during the same rules processing. Thus, the path $(R_1; R_2; R_1)$ is impossible. An infinite loop is impossible. But the RTG method cannot draw this conclusion, because the RTG method cannot include in a triggering formula atoms containing variables which can be modified by rules actions.

We propose to include in the triggering formula the $descendants$ of a rules-dependent atom due to the rules actions. For instance, the descendant of the atom (A_2.rate = 4) due to the rule R_3 is (A_2.type = stocks_account) $\wedge$ (A_2.rate = 4), and the descendant of the atom (A_2.rate = 4) due to the rule R_4 is (A_2.type = standard_account) $\wedge$ (A_2.rate = 8). Thanks to the atom descendants, we will be able to build a logic formula containing rules-dependent atoms: the *descendant of a triggering formula*. The descendant of a triggering formula will be exploited to build the *complex condition of a rule*.

5 Triggering Formula of a Complex Path

In this section, we introduce the notion of *triggering formula of a complex path*. This notion will be used to compute the descendants of a rules-dependent atom. A *complex path* is an entity composed of all the paths which lead to a given rule. A *triggering formula of a complex path* is a logic formula capturing the rules triggering along all the paths of the complex path.

5.1 Elementary Path and Complex Path

We first define the notion of *elementary path*. We introduce then the notion of *complex order M path of a rule*.

Elementary Path. Let $R_1, R_2, \ldots R_i, \ldots R_n$ be n rules (not necessarily all distinct) of a triggering graph such that there is an edge from R_{i+1} to R_i. The tuple $(R_1, R_2, \ldots R_n)$ makes up an *elementary path*. We adopt the following notation: $R_1 \leftarrow R_2 \leftarrow \ldots \leftarrow R_i \leftarrow \ldots \leftarrow R_n$ (Note that we adopt a notation in the opposite direction of the edges, for convenience). R_1 is said to be the *last rule* of the elementary path. R_n is said to be the *first rule* of the elementary path.

Complex Order *M* Path of a Rule. The notion of *complex order M path of a rule Rule* will be used to build a logic formula containing the rules-independent atoms included in the postconditions of the rules preceding *Rule* in the triggering graph.

We introduce the following notation: if $Path_1$ and $Path_2$ are two elementary paths, we denote the set $\{Path_1, Path_2\}$ as follows: $Path_1 \vee Path_2$.

The *complex order M path of Rule* is the set containing all the distinct elementary paths having M nodes (not necessarily all distinct) and having *Rule* as last node. We will use the following notation:

$$Comp\_Path(Rule; M) = Elementary\_Path_1 \vee \ldots \vee Elementary\_Path_p$$

If $Path_1, Path_2$ and $Path_3$ are three elementary paths such that $Path_2$ and $Path_3$ have the same last rule, and such that there is an edge from the last rule of $Path_2$(and $Path_3$) to the first rule of $Path_1$, we define:

$$Path_1 \leftarrow (Path2 \vee Path3) = (Path_1 \leftarrow Path_2) \vee (Path_1 \leftarrow Path_3)$$

The function giving the complex order M path of *Rule* is then the following:

$$Comp\_Path(\text{incoming variables:}Rule, M; \text{outgoing variable:}Path_{out})$$

Let $R_1, R_2, \ldots R_p$ be the rules such that there is an edge from R_i to *Rule*
IF $M > 1$
THEN $Path_{out} = (Rule \leftarrow Comp\_Path(R_1, M-1)) \vee$
$(Rule \leftarrow Comp\_Path(R_2, M-1)) \vee \ldots \vee (Rule \leftarrow Comp\_Path(R_p, M-1))$
ELSE $Path_{out} = (Rule)$

Example. See Figure 1.
$Comp\_Path(R_3, 4) = R_3 \leftarrow R_1 \leftarrow R_2 \leftarrow R_1$
$Comp\_Path(R_4, 6) = R_4 \leftarrow R_2 \leftarrow R_1 \leftarrow R_2 \leftarrow R_1 \leftarrow R_2$

5.2 Rules-Independent Triggering Formula of a Path

The *rules-independent triggering formula* of a path is a logic formula, which contains the rules-independent atoms of the postconditions of the rules of the path.

Rules-Independent Triggering Formula of a Elementary Path. Let us consider a elementary path $Path = R_1 \leftarrow R_2 \ldots \leftarrow R_n$. The *rules_independent formula of the path RI_Formula(Path)* is defined as:
$RI\_Formula(Path) =$
$Postcondition(R_1) \wedge ParameterFormula(R_1 \leftarrow R_2) \wedge$
$Postcondition(R_2) \wedge \ldots \wedge$
$ParameterFormula(R_{n-1} \leftarrow R_n) \wedge Postcondition(R_n)$
where the selection criterion for $Post\_SubSet(R_i)(1 \leq i \leq n)$ is the following:
$Post\_SubSet(R_i)$ contains the rules-independent atoms of $Post\_Set(R_i)$.

Rules-Independent Triggering Formula of a Complex Path. Let *Rule* be a rule. We associate to the complex order M path of *Rule* a rules-independent formula:
$RI\_Formula(Comp\_Path(Rule; M))$
If $Comp\_Path(Rule; M) = Path_1 \vee Path_2 \vee \ldots \vee Path_p$,then:
$RI\_Formula(Comp\_Path(Rule; M)) =$
$RI\_Formula(Path_1) \vee RI\_Formula(Path_2) \vee \ldots \vee RI\_Formula(Path_p)$
$RI\_Formula(Comp\_Path(Rule; M))$ expresses a necessary condition, which the database state has to satisfy, at any time of the rules processing, if the action of *Rule* is executed during the rules processing.

Example. Let us consider the rule R_3 of our motivating example. We have:
$Comp\_Path(R_3, 4) = R_3 \leftarrow R_1 \leftarrow R_2 \leftarrow R_1$
$RI\_Formula(Comp\_Path(R_3, 4)) = (A_3.\text{type} = \text{stocks\_account}) \wedge (A_3 = A_{11})$
$\wedge\ (A_{11}.\text{capacity} = 6000) \wedge (A_{11} = A_2) \wedge (A_2.\text{overdraft} = 20) \wedge (A_2 = A_{12}) \wedge$
$(A_{12}.\text{capacity} = 6000)$
$= (A_3.\text{type} = \text{stocks\_account}) \wedge (A_3.\text{capacity} = 6000) \wedge (A_3.\text{overdraft} = 20)$
(Note that, when a rule appears several times in a path, the variables of the rule are indexed with an occurrence number of the rule).

6 Descendant of a Triggering Formula

In order to determine the possible effects of rules actions on objects satisfying rules-dependent selectors, we introduce the notion of *descendant of a rules-dependent selector*. This notion will allow us to define the *descendant of a triggering formula*.

6.1 Total Triggering Formula

Let $Formula(R \leftarrow R')$ be a triggering formula:

$$Formula(R \leftarrow R') =$$
$$Precondition(R) \wedge Postcondition(R') \wedge ParameterFormula(R \leftarrow R')$$

We say that $Formula(R \leftarrow R')$ is a *total triggering formula* iff

- the selection criterion used for $Pre\_SubSet(R)$ is the following:
  $Pre\_SubSet(R)$ contains the rules-independent atoms of $Pre\_Set(R)$ and the rules-dependent selectors of $Pre\_Set(R)$, and
- the selection criterion used for $Post\_SubSet(R')$ is the following:
  $Post\_SubSet(R')$ contains the rules-independent atoms of $Post\_Set(R')$ and the rules-dependent selectors of $Post\_Set(R')$.

Let us note that, since $Formula(R \leftarrow R')$ contains rules-dependent selectors, $Formula(R \leftarrow R')$ cannot be used by the RTG method for termination analysis. The satisfiability of a total triggering formula is not meaningful, since variables of the formula can be updated between the action execution of R' and the condition evaluation of *R*.

Example. Let us consider our motivating example. Let us consider the following formula:
$Total\_Formula(R_2 \leftarrow R_1) =$
$(A_1.\text{rate} = 8) \wedge (A_1.\text{capacity} = 6000) \wedge (A_1 = A_2) \wedge (A_2.\text{rate} = 4)$
is a total triggering formula. Note that the satisfiability of this formula is not meaningful, since the account rate can be updated between the action of R_1 and the condition evaluation of R_2.

6.2 Descendant of a Rules-Dependent Selector

Let *Rule* be a rule. Let $Total\_Formula(R \leftarrow R')$ be a total triggering formula binding the rules *R* and R'. Let *Selector* be a rules-dependent selector $(X.l = t)$ contained in *Postcondition(R')*.

The intuition behind the notion of *descendant* of a selector is the following: let us consider an object X that satisfies: $(X.l = t)$; the descendant of $Selector$ due to *Rule* captures the possible transformation of the attribute l of X due to the action of the rule *Rule*.

The descendant of *Selector* due to *Rule* is a logic formula that unifies the object variable X of *Selector* and the recipient object variable of *Rule*, and that expresses the modification of the attribute l by the action of *Rule*.

The rule *Rule* is specified as follows:
Event: $d(Y_0@Y_1, \ldots, Y_m)$
Condition: c
Action: $Y_0.l_1 = Z_1, \ldots, Y_0.l_k = Z_k$
Raised events: $e_1, \ldots, e_k$

We define the *descendant* of *Selector* due to the rule *Rule* as:
$Descendant(Selector; Rule) =$
$(X = Y_0) \wedge RI\_Formula(Comp\_Path(Rule; M)) \wedge (((l = l_1) \wedge (Y0.l_1 = Z_1)) \vee$
$((l = l_2) \wedge (Y_0.l_2 = Z_2)) \vee \ldots \vee ((l = l_k) \wedge (Y_0.l_k = Z_k)))$

- $(X = Y_0)$ captures the variable unification between the object variable X of $Selector$ and the object variable Y_0 of the rule action of *Rule*;
- $RI\_Formula(Comp\_Path(Rule; M)$ captures the rules-independent atoms along $Comp\_Path(Rule; M)$;
- $((l = l_i) \wedge (Y_0.l_i = Z_i))$ captures the modification of the attribute l by the action of $Rule$.

The number M is set by the designer, and is called the *descendant depth*. The logic formula $Descendant(Selector; Rule)$ expresses a necessary condition, which the database state has to satisfy, from the moment of an update of the attribute l of the object X by the action of *Rule*, until the moment of an update of the attribute l of X by an other rule action.

Example. Let us consider our motivating example. Let us consider the following total triggering formula:
$Total\_Formula(R_2 \leftarrow R_1) =$
$(A_1.\text{rate} = 8) \wedge (A_1.\text{capacity} = 6000) \wedge (A_1 = A_2) \wedge (A_2.\text{rate} = 4)$

Let us pose: $Selector = (A_1.\text{rate} = 8)$.
We set the descendant depth to 2.
We have: $Descendant(Selector; R_3) = (A_1 = A_3) \wedge (A_3.\text{type} = \text{stocks\_account})$
$\wedge(A_3.\text{rate} = 4) \wedge(A_3 = A_{11}) \wedge (A_{11}.\text{capacity} = 6000)$
$= (A_1.\ \text{type} = \text{stocks\_account}) \wedge(A_1.\text{rate} = 4) \wedge(A_1.\text{capacity} = 6000)$

6.3 Descendant of a Total Triggering Formula

Let $Total\_Formula(R \leftarrow R')$ be a total triggering formula. The *descendant of the total triggering formula* is a logic formula that expresses the condition evaluation of *R*, and that takes into account the triggering of *R* by R'. This logic formula includes all the descendants of the rules-dependent atoms contained in $Postcondition(R')$.

The *descendant of Total_Formula*$(R \leftarrow R')$ is the following formula:
$Descendant(Total\_Formula(R \leftarrow R')) = Precondition(R) \wedge$
$Descendant(Postcondition(R')) \wedge ParameterFormula(R \leftarrow R')$
where:
•$Precondition(R)$ and $ParameterFormula(R \leftarrow R')$ are defined as for $Total\_Formula(R \leftarrow R')$;
•$Descendant(Postcondition(R'))$ is obtained from $Postcondition(R')$, replacing each rules-dependent selector *Selector* of $Postcondition(R')$ by:
$Selector \vee (V_{(\text{for each}Rule \in G)}(Descendant(Selector; Rule))$
(G is the triggering graph)
Contrary to the satisfiability of a total triggering formula, the satisfiability of the descendant of a total triggering formula is meaningful: the descendant of a total triggering formula expresses a necessary condition, which the database state has to satisfy, at the moment of the evaluation of the condition of *R*, when *R* is triggered by R', if the condition of *R* is satisfiable. If the descendant of $Total\_Formula(R \leftarrow R')$ is **FALSE**, this means that the condition of *R* cannot be evaluated to **TRUE** when *R* is triggered by R'.

Example. Let us consider our motivating example.
Let us consider the following total triggering formula:
$Total\_Formula(R_2 \leftarrow R_1) =$
$(A_1.\text{rate} = 8) \wedge (A_1.\text{capacity} = 6000) \wedge (A_1 = A_2) \wedge (A_2.\text{rate} = 4)$
We set the descendant depth to 1. We have:
$Descendant(Total\_Formula(R_2 \leftarrow R_1)) =$
$((A_1.\text{rate} = 8) \vee ((A_1 = A_3) \wedge (A_3.\text{rate} = 4) \wedge (A_3.\text{type} = \text{stocks\_account})) \vee ((A_1 = A_4) \wedge (A_4.\text{rate} = 8) \wedge (A_4.\text{type} = \text{standard\_account}))) \wedge (A_1.\text{capacity} = 6000) \wedge (A_1 = A_2) \wedge (A_2.\text{rate} = 4)$
$= (A_2.\text{rate} = 4) \wedge (A_2.\text{type} = \text{stocks\_account}) \wedge (A_2.\text{capacity} = 6000)$

7 Complex Condition of a Rule

We introduce the notion of *complex condition of a rule*. The complex condition of a rule *R* is a logic formula that the database state has to satisfy, at the moment of the evaluation of the condition of *R*, if the condition of *R* is satisfiable. This logic formula includes information about the rules which can trigger *R*. We propose then the notion

of *complex postcondition*, and the notion of *elaborate triggering formula*. The intuition behind an *elaborate triggering formula* is to allow an iterative computation of the complex conditions.

7.1 Complex Order 2 Condition of a Rule

The *complex order* 2 *condition of a rule R* is the disjunction of the formulas $Descendant(Total\_Formula(R \leftarrow R_i))$, for all the rules R_i such that there is an edge from R_i to *R*:

Let $R_1, R_2, \ldots R_p$ be the rules such that there is an edge from R_i to *R*
$Comp\_Condition(R; 2) =$
$Descendant(Total\_Formula(R \leftarrow R_1)) \vee Descendant(Total\_Formula(R \leftarrow R_2)) \vee \ldots \vee Descendant(Total\_Formula(R \leftarrow R_p))$
The complex order 2 condition of R takes into account the complex order 2 path of R. If $Comp\_Condition(R; 2)$ is equal to **FALSE**, this means that the condition of *R* cannot be evaluated to **TRUE** during a rules processing.

Example. Let us consider our motivating example.
$Comp\_Condition(R_2; 2) =$
$Descendant(Total\_Formula(R_2 \leftarrow R_1)) =$
$(A_2.\text{rate} = 4) \wedge (A_2.\text{type} = \text{stocks\_account}) \wedge (A_2.\text{capacity} = 6000)$

7.2 Complex Order 2 Postcondition of a Rule

A complex condition of the rule *R* can be used to build a more precise postcondition of *R* (that is: including more information). The rule *R* is specified as follows:
Event: $d(Y_0@Y_1, \ldots, Y_m)$
Condition: c
Action: $Y_0.l_1 = Z_1, \ldots, Y_0.l_k = Z_k$
Raised events: $e_1, \ldots, e_k$
The *complex order 2 postcondition* of the rule *R* is:
$Comp\_Postcondition(R; 2) =$
$Comp\_Condition(R; 2) \wedge (Y_0.l_1 = Z_1) \wedge (Y_0.l_2 = Z_2) \wedge \ldots \wedge (Y_0.l_k = Z_k)$
(Note that the satisfiability of $Comp\_Postcondition(R; 2)$ is not meaningful, since the database state has not necessarily to satisfy this formula.)

Example. Let us consider our motivating example.
$Comp\_Postcondition(R_2; 2) = (A_2.\text{rate} = 4) \wedge (A_2.\text{type} = \text{stocks\_account}) \wedge (A_2.\text{capacity} = 6000) \wedge (A_2.\text{overdraft} = 20)$

7.3 Elaborate Triggering Formula

We can compute a complex condition of a rule *R*, using the information contained in the previously computed complex postconditions of the rules which can trigger *R*. So, we

will obtain a more precise complex condition (that is: including more information). To achieve this, we define the notion of *elaborate triggering formula*. We give an iterative definition of the *elaborate order N triggering formula*.

We set: $Elaborate\_Formula(R \leftarrow R'; 2) = Total\_Formula(R \leftarrow R')$.

Let us consider N such that $N \geq 3$. Let *R*, R' be two rules such that the complex order $(N-1)$ postcondition of R' has previously been computed. We call an *elaborate order N triggering formula* the following formula:
$Elaborate\_Formula(R \leftarrow R'; N) = Precondition(R) \wedge$
$Comp\_Postcondition(R'; N-1) \wedge ParameterFormula(R \leftarrow R')$
(where $Precondition(R)$ and $ParameterFormula(R \leftarrow R')$ are defined as for $Total\_Formula(R \leftarrow R')$). As for a total triggering formula, we define the *descendant of an elaborate triggering formula*:
$Descendant(Elaborate\_Formula(R \leftarrow R'; N)) = Precondition(R)($
$\wedge Descendant(Comp\_Postcondition(R'; N-1)) \wedge ParameterFormula(R \leftarrow R')$
where:
• $Precondition(R)$ and $ParameterFormula(R \leftarrow R')$ are defined as for $Total\_Formula(R \leftarrow R')$;
• $Descendant(Comp\_Postcondition(R'; N-1))$ is obtained from $Comp\_Postcondition(R'; N-1)$,
replacing each rules-dependent selector $Selector$ of $Comp\_Postcondition(R'; N-1)$ by:
$Selector \vee (\vee_{(mbox foreach Rule \in G)}(Descendant(Selector; Rule))$
We use the elaborate order N formulas to compute the *complex order N condition* of a rule. Let $R_1, R_2, \ldots R_p$ be the rules such that there is an edge from R_i to *R*. The *complex order N condition* of *R* is computed as follows:
$Comp\_Condition(R; N) =$
$Descendant(Elaborate\_Formula(R \leftarrow R_1; N)) \vee$
$Descendant(Elaborate\_Formula(R \leftarrow R_2; N)) \vee \ldots \vee$
$Descendant(Elaborate\_Formula(R \leftarrow R_p; N))$
We use the complex order N condition of R to compute the *complex order N postcondition* of *R* (*R* is specified as at paragraph 7.2):
$Comp\_Postcondition(R; N) = Comp\_Condition(R; N) \wedge (Y_0.l_1 = Z_1) \wedge (Y_0.l_2 = Z_2) \wedge \ldots \wedge (Y_0.l_k = Z_k)$
The satisfiability of $Elaborate\_Formula(R \leftarrow R'; N)$ is not meaningful, but the satisfiability of $Descendant(Elaborate\_Formula(R \leftarrow R'; N))$ is meaningful: this logic formula expresses a necessary condition, which the database state has to satisfy, at the moment of the evaluation of the condition of *R*, when *R* is triggered by R', and if the condition of *R* is satisfiable. The logic formula $Descendant(Elaborate\_Formula(R \leftarrow R'; N))$ takes into account the information about the complex order $(N-1)$ path of R'. In the same way, the satisfiability of $Comp\_Condition(R; N)$ is meaningful. (Note that the satisfiability of $Comp\_Postcondition(R; N)$ is not meaningful).

We can compute more and more precise complex conditions, using the previously computed complex postconditions. Notice that the complex order N condition of *R* takes into account the complex order N path of *R*.

Example. Let us consider our motivating example. We set the descendant depth to 1. We have
$Comp\_Postcondition(R_2; 2) =$
(A_2.rate = 4) $\wedge$(A_2.type = stocks_account) $\wedge$
(A_2.capacity = 6000) $\wedge$(A_2.overdraft = 20)

Let us compute $Elaborate\_Formula(R_1 \leftarrow R_2; 3)$:
$Elaborate\_Formula(R_1 \leftarrow R_2; 3) =$
(A_1.rate= 8) $\wedge$ ($A_1 = A_2$) $\wedge$ (A_2.rate = 4)$\wedge$
(A_2.type = stocks_account) $\wedge$(A_2.capacity= 6000) $\wedge$ (A_2.overdraft = 20)
$Descendant(Elaborate\_Formula(R_1 \leftarrow R_2; 3)) =$
(A_1.rate= 8) $\wedge$ ($A_1 = A_2$) $\wedge$ ((A_2.rate= 4)$\vee$
(($A_2 = A_3$) $\wedge$ (A_3.rate= 4) $\wedge$ (A_3.type = stocks_account))$\vee$
(($A_2 = A_4$) $\wedge$ (A_4.rate= 8) $\wedge$ (A_4.type = standard_account))$\wedge$
(A_2.type = stocks_account) $\wedge$ (A_2.capacity= 6000) $\wedge$ (A_2.overdraft= 20)
= (A_1.rate= 8) $\wedge$ (A_1.type = standard_account))$\wedge$
(A_1.type = stocks_account) $\wedge$ (A_1.capacity= 6000)$\wedge$
(A_1. overdraft = 20) = **FALSE**

So, we have:
$Comp\_Condition(R_1; 3)$ =
$Descendant(Elaborate\_Formula(R_1 \leftarrow R_2; 3))$ = **FALSE**

8 Termination Algorithm

We can now sketch our termination algorithm. This algorithm is executed at compile-time. The designer sets a time limit for termination static analysis (beyond the time limit, the algorithm is stopped), and the descendant depth.

(Initialisation)
G = initial triggering graph
REPEAT
remove from G the rules without incoming edge
remove from G the edges without origin node
UNTIL (no rule is removed from G)
(Iterative computation of complex order N conditions)
N = 2
REPEAT
FOR each rule *R* of G
compute *Comp_Condition(R ; N)*
IF $Comp\_Condition(R; N)$ = **FALSE**
THEN BEGIN
remove *R* from G
REPEAT
remove from G the rules without incoming edge
remove from G the edges without origin node

UNTIL (no rule is removed from G)
ENDIF
ENDFOR
$N = N + 1$
UNTIL (time limit is reached) **OR** (termination is detected)

Termination is detected when G contains no more rule. When the algorithm is stopped by the time limit, termination is not detected.

The complex order N conditions are simplified as follows:

- The complex order N conditions are rewritten using the disjunctive normal form.
- The satisfiability of a conjunction of atoms/negated atoms is analysed by means of a satisfiability algorithm [19, 25, 26, 27]. The choice of the satisfiability algorithm depends on the first-order language used to specify the atoms. When a conjunction of atoms/negated atoms is detected as not satisfiable, this conjunction of atoms/negated atoms is replaced by **FALSE**.
- When a rule *R* has been removed from G, the descendants of selectors due to $R : Descendant(Selector; R)$ (for any selector *Selector*), and the descendants of elaborate triggering formulas: $Descendant(Elaborate\_Formula(R' \leftarrow R; k))$ (for any rule R' and for $2 \leq k \leq N$), included in $Comp\_Condition(Rule; N)$ (for any rule *Rule*), are set to **FALSE**.

The complex order N condition of a rule takes into account information about the complex order N path of the rule. The number N is called the *termination analysis depth*. For a defined time limit, the termination analysis depth which can be reached depends on:

- the number of rules;
- the complexity of the triggering graph, that is the number of rules which can trigger a rule;
- the number of primitive conditions of the rules;
- the complexity of the primitive conditions;
- the number of selectors included in rules actions;
- the number of rules-dependent selectors included in rules conditions and actions;
- the number of rules actions which can update the same attribute;
- the descendant depth;
- the satisfiability of the descendants at each intermediate analysis depth.

The RTG Method [21, 28] corresponds to a termination analysis depth equal to 2 and a descendant depth equal to 0. The Path Removing Technique [23] corresponds to a termination analysis depth equal to the length of the cycles, and a descendant depth equal to 0, and the Cycle Unrolling Technique [24] corresponds to a termination analysis depth equal to a multiple of the length of the cycles, and a descendant depth equal to 0. Note that, beside the fact that our algorithm can include rules-dependent selectors in triggering formulas, our algorithm takes into account rules paths which are not necessarily included in cycles, while the RTG Method [21, 28], the Path Removing Technique [23] and the Cycle Unrolling [24] only take into account rules paths included in cycles.

Example. Let us analyse the termination of our motivating example. We set the descendant depth to 1.

The termination analysis depth equal to 2 gives the following results:
$Comp\_Condition(R_2; 2) =$
$(A_2$.rate$= 4) \wedge (A_2$.type = stocks_account$) \wedge (A_2$.capacity$= 6000)$
$Comp\_Condition(R_1; 2) =$
$(A_1$.rate$= 8) \wedge (A_1$.type = standard_account$) \wedge (A_1$.overdraft$= 20)$
$Comp\_Condition(R_3; 2) =$
$((A_3$.rate$= 8) \vee (A_3$.rate$= 4)) \wedge (A_3$.type = stocks_account$) \wedge (A_3$.capacity$= 6000)$
$Comp\_Condition(R_4; 2) =$
$((A_4$.rate$= 4) \vee (A_4$.rate$= 8)) \wedge (A_4$.type = standard_account$) \wedge (A_4$.overdraft$= 20)$
The termination analysis depth equal to 3 gives the following results:
$Comp\_Condition(R_1; 3)$
$= Descendant(Elaborate\_Formula(R_1 \leftarrow R_2))$ =**FALSE**
R_1 is removed from G.
R_2 and R_3 are removed from G.
R_4 is removed from G.

So, termination is guaranteed by our algorithm for the motivating example, with a termination analysis depth equal to 3 and a descendant depth equal to 1. Note that no previous termination algorithm is able to give this result (see paragraph 4.2). Extension to the Relational Model. The RTG method has been proposed for the object oriented model, and we have also presented our improvement for the object oriented model. But it is possible to fit the RTG method and our improvement of the RTG method to the relational model. We give an intuition of this extension using a small (and informal) example. We use a relational table $CLIENT$, and we informally define two rules.

$RuleR_1$:
$Event$: update of the attribute $type$ for some clients (the set $Set_1 1$ of the concerned clients is transferred to the Condition part), $Condition$: selection of the clients of the set $Set_1 1$, whose loan rate is greater than 10, (the set $Set_1 2$ of the selected clients is transferred to the Action part), $Action$: update of the attribute bonus for the clients of the set $Set_1 2$.
$RuleR_2$:
$Event$: update of the attribute bonus for some clients, (the set $Set_2 1$ of the concerned clients is transferred to the Condition part), $Condition$: selection of the clients of the set $Set_2 1$, whose loan rate is less than 5, (the set $Set_2 2$ of the selected clients is transferred to the Action part) $Action$: update of the attribute type for the clients of the set $Set_2 2$.

We can build a formula, concerning the triggering of the rule R_2 by the rule R_1. We first observe that we can unify a client concerned by the action of R_1 and a client concerned by the event of R_2. Informally, the formula can be expressed as follows: "the loan rate of the client is greater than 10 and the loan rate of the client is less than 5", that is: $(CLIENT$.rate $> 10)(\wedge(CLIENT$.rate $< 5)$. This formula is similar to a triggering formula.

9 Conclusion

We have presented a significant improvement of the Refined Triggering Graph Method [21, 28]. We have introduced the notions of:

- *complex path of a rule*: this entity includes all the elementary paths having a given rule as last node;
- *rules-independent triggering formula of a complex path*: this logic formula captures the rules-independent atoms included in postconditions of a complex path;
- *descendant of a selector due to a rule*: this logic formula expresses the transformation of an object, which satisfies the selector, due to a rule action;
- *descendant of a triggering formula*: this logic formula expresses the possible transformations of objects which satisfy the triggering formula, due to the rules actions;
- *complex condition of a rule*: this logic formula takes into account the descendants of triggering formulas concerning the rule;
- *elaborate triggering formula*: this logic formula captures the triggering link between two rules taking into account the complex postcondition of the triggering rule.

Our algorithm can include in triggering formulas rules-dependent atoms. This is impossible for the Refined Triggering Graph Method [21, 28], the Path Removing Technique [23], the Cycle Unrolling Technique [24]. Thanks to our improvement, many more termination situations can so be detected, since the RTG Method, improved the Path Removing Technique and the Cycle Unrolling, is a special case of our algorithm, where the descendant depth is equal to 0.

In the future, we will develop heuristics to choose the descendant depth, according to the type of the rules and to type of the triggering graph.

References

1. A. Aiken, J. Widom, J.M. Hellerstein. Behavior of Database Production Rules: Termination, Confluence and Observable Determinism. In *Proc. Int'l Conf. on Management of Data (SIGMOD)*, San Diego, California, 1992.
2. J. Bailey, G. Dong, K. Ramamohanarao. Decidability and Undecidability Results for the Termination Problem of Active Database Rules. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, Seattle, Washington, 1998.
3. E. Baralis, S. Ceri, S. Paraboschi. Improved Rule Analysis by Means of Triggering and Activation Graphs. In *Proc. Int'l Workshop Rules in Database Systems (RIDS)*, Athens, Greece, 1995.
4. E. Baralis, S. Ceri, S. Paraboschi. Modularization Techniques for Active Rules Design. In *ACM Transactions on Database Systems, (TODS)*, 21(1), 1996.
5. E. Baralis, S. Ceri, S. Paraboschi. Compile-Time and Run-Time Analysis of Active Behaviors. In *IEEE Transactions on Knowledge and Data Engineering*, 10 (3), 1998.
6. J. Bailey, L. Crnogorac, K. Ramamohanarao, H. Søndergaard: Abstract Interpretation of Active Rules and its Use in Termination Analysis. In *Proc. of the Int'l Conf. on Database Theory (ICDT'97)*, Delphi, Greece, 1997.
7. J. Bailey, A. Poulovassilis. An Abstract Interpretation Framework for Termination Analysis of Active Rules. In *Proc. DataBase Programming Languages (DBPL'99)*, Kinloch-Rannoch, Scotland, 1999.
8. J. Bailey, A. Poulovassilis. Abstract Interpretation for Termination Analysis in Functional Active Databases. In *Journal of Intelligent Information Systems*, 12 (2-3), 1999.
9. J. Bailey, A. Poulovassilis, P. Newson. A Dynamic Approach to Termination Analysis for Active Database Rules. In *Proc. Int'l Conf. on Deductive Object Oriented Databases (DOOD 2000)*, London, UK, 2000.

10. E. Baralis, J. Widom. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Santiago, Chile, 1994.
11. E. Baralis, J. Widom. An Algebraic Approach to Static Analysis of Active Database Rules. In *ACM Transactions on Database Systems (TODS)*, 25(3), 2000.
12. S. Ceri, J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Brisbane, Queensland, Australia, 1990.
13. S. Comai, L. Tanca. Using the Properties of Datalog to prove Termination and Confluence in Active Databases. In *Proc. Int'l Workshop on Rules in Database Systems (RIDS)*, Skoevde, Sweden, 1997.
14. A. Couchot. Improving Termination Analysis of Active Rules with Composite Events. In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, Munich, Germany, 2001.
15. A. Couchot. Termination Analysis of Active Rules Modular Sets. In *Proc. Int'l Conf. on Information and Knowledge management (CIKM)*, Atlanta, Georgia, USA, 2001.
16. U. Dayal, A.P. Buchmann, D.R. Mc Carthy. Rules are Objects too: A Knowledge Model for an Active Object Oriented Database System. In *Proc. Int'l Workshop on Object-Oriented Database Systems*, Bad Münster am Stein-Ebernburg, Germany, 1988.
17. S. Debray, T. Hickey. Constraint-Based Termination Analysis for Cyclic Active Database Rules. In *Proc. Int'l Conf. on Deductive Object Oriented Databases (DOOD)*. London, United Kingdom, 2000.
18. S. Flesca, S. Greco. Declarative Semantics for Active Rules. In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, Vienna, Austria, 1998.
19. J.P. Jouannaud, C. Kirchner. Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification. In J.L. Lassez and G. Plotkin, editors. *Computational Logic*, pp. 257-321. MIT Press, 1991.
20. A.P. Karadimce, S.D. Urban. Conditional Term Rewriting as a Formal Basis for Analysis of Active Database Rules. In *Proc. Int'l Workshop on Research Issues in Data Engineering (RIDE-ADS)*, Houston, Texas, USA, 1994.
21. A.P. Karadimce, S.D. Urban. Refined Triggering Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database. In *Proc. Int'l Conf. on Data Engineering (ICDE)*, New-Orleans, Louisiana, USA, 1996.
22. S.Y. Lee, T.W. Ling. Refined Termination Decision in Active Databases. In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, Toulouse, France, 1997.
23. S.Y. Lee, T.W. Ling. A Path Removing Technique for Detecting Trigger Termination. In *Proc. Int'l Conf. on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.
24. S.Y. Lee, T.W. Ling. Unrolling Cycle to Decide Trigger Termination. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, 1999.
25. A. Martelli, U. Montanari. An Efficient Unification Algorithm. *ACM Trans. on Programming Lang. and Syst.*, 4(2):258-282, 1982.
26. D. Rosenkrantz, H.B. Hunt. Processing Conjunctive Predicates and Queries. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Montréal, Canada, 1980.
27. G. Smolka. Feature-Constraint Logics for Unification Grammars. *Journal of Logic Programming*. 12:51-87, 1992.
28. M.K. Tschudi, S.D. Urban, S.W. Dietrich, A.P. Karadimce. An Implementation and Evaluation of the Refined Triggering Graph Method for Active Rule Termination Analysis. In *Proc. Int'l Workshop on Rules in Database Systems*, Skoevde, Sweden, 1997.
29. A. Vaduva, S. Gatziu, K.R. Dittrich. Investigating Termination in Active Database Systems with Expressive Rule Languages. In *Proc. Int'l Workshop on Rules in Database Systems*, Skoevde, Sweden, 1997.
30. T. Weik, A. Heuer. An Algorithm for the Analysis of Termination of Large Trigger Sets in an OODBMS. In *Proc. Int'l Workshop on Active and Real-Time Databases*. Skoevde, Sweden, 1995.

An Algorithm for Determining Related Constraints

Gaihua Fu[1], Jianhua Shao[1], Suzanne M. Embury[2], and W. Alex Gray[1]

[1] Department of Computer Science
Cardiff University, Cardiff, CF24 3XF, UK
{scmgf,J.Shao,W.A.Gray}@cs.cf.ac.uk
[2] Department of Computer Science
University of Manchester, Manchester, M13 9PL, UK
SEmbury@cs.man.ac.uk

Abstract Constraints are a class of business rules that many organisations implement in their information systems. However, it is common that many implemented constraints do not get documented. This has led researchers to consider how to recover constraints from implementations. In this paper, we consider the problem of how to analyse the set of constraints extracted from legacy systems. More specifically, we introduce an algorithm for determining which constraints are related according to some criteria. Since constraints are typically fragmented during their implementation, the ability to determine a set of related constraints is useful and important to the comprehension of extracted constraints.
Keywords: Reverse Engineering, Constraint Business Rule, Constraint Analysis.

1 Introduction

Constraints are a class of business rules that describe the conditions under which an organisation operates [12,19], and they are typically enforced in information systems by managing and constraining the behavioural elements, such as events and methods, so that undesirable manipulation of business objects will not occur. Unfortunately, while most information systems implement constraints, few document them accurately [18,19,13,17]. This makes it difficult for an organisation to understand the set of constraints it actually enforces and to evolve them effectively in response to business changes.

To help the situation, researchers have considered how to develop techniques to support the extraction of constraints buried in source code [7,5,13,17]. While these techniques are useful in identifying constraints from physical implementations, they are rather limited in what they can do to help users understand the extracted constraints. Most existing systems simply present the constraints as extracted and do not support their analysis. This is not satisfactory for two reasons:

- Constraints are typically fragmented when implemented. As a result, semantically related constraints may not be extracted in a collective manner. Thus, it is important that we analyse the extracted constraints, so that the “broken" links among the related constraints may be recovered.

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 134–149, 2002.

– Extracted constraints may be used by different user groups for different purposes. Data analysts may want to know which constraints are responsible for the value assumed by a specific business object, whereas system maintainers may want to understand whether there exists a conflict among the constraints implemented in the system. It is desirable, therefore, that we can analyse the extracted constraints for various properties.

Consider the following example. Suppose that we have the following two constraints that are used by a mobile phone service promoter:

```
BR1  The company offers 2 categories of free-talk time:
     300 minutes and 600 minutes.
BR2  Only Vodafone customers are entitled
     to 600 free-talk time.
```

Clearly, if we wish to understand why in the company's customer database, every customer is offered some free-talk time, but only Vodafone customers are given 600 minutes, then the above two constraints (or business rules) must be interpreted and understood together.

In this paper, we consider the problem of how to analyse the set of extracted constraints to determine which ones are related with respect to some criteria. We present an algorithm for analysing the constraints expressed in $\mathcal{BRL}$, a meta-level language that we have proposed to represent the extracted constraints [9]. Our algorithm supports the derivation of related constraints for a given set of business objects, and allows some implicit constraints to be derived as part of the process. The proposed algorithm has a polynomial complexity, and can therefore scale to non-trivial applications.

The paper is organised as follows. In Section 2, we review the related work in constraint representation and analysis, and briefly explain how constraints are expressed in $\mathcal{BRL}$. Section 3 discusses what we mean by related constraints. An algorithm is given in Section 4 for analysing constraints expressed in $\mathcal{BRL}$ and for determining the related ones. In Section 5, we give a complete example to show how the proposed algorithm works. Conclusions are given in Section 6.

2 Representation and Analysis of Constraints

One important factor that affects constraint analysis is the language in which constraints are represented. Most constraint languages proposed in the literature represent constraints at the *instance* level [1,21,16,20,14]. These languages are typically based on First Order Logic (FOL), and specify a constraint by stating what the instances of a class must satisfy. For example, `BR2` given in Section 1 may be represented at the instance level in FOL as follows:

$$\forall x, y_1, y_2 (service(x) \wedge service\_freeTime(x, y_1) \wedge freeTime(y_1) \wedge service\_network(x, y_2) \wedge network(y_2) \wedge (y_1 = 600) \rightarrow (y_2 = Vodafone))$$

The advantage of this type of constraint language is its expressive power [10]. However, constraints expressed at the instance level are less intuitive, less readable, and

difficult to analyse. For example, it is not obvious to see whether the above formula is about `service`, `network`, `freeTime` or all of them. As another example, consider the checking of whether the constraint "Minimum number of customers is 100" entails the constraint "Minimum number of customers is 90". Conceptually, this could be done by an integer comparison. However, when they are represented at the instance level in FOL:

$$\exists x_1, \ldots, x_{100}(customer(x_1) \wedge \ldots \wedge customer(x_{100}) \wedge (x_1 \not= x_2) \wedge (x_1 \not= x_3) \ldots \wedge (x_{99} \not= x_{100}))$$

$$\exists x_1, \ldots, x_{90}(customer(x_1) \wedge \ldots \wedge customer(x_{90}) \wedge (x_1 \not= x_2) \wedge (x_1 \not= x_3) \ldots \wedge (x_{89} \not= x_{90}))$$

a theorem prover must be used to show that the second formula is implied by the first.

The alternative approach is to represent constraints at the *meta* level [2,6,15]. The idea is based on the observation that since constraints are intended to express the properties that must be met by every instance of a class, they essentially deal with classes themselves. Thus, representing constraints at the class or meta level can be more natural. There exists a large body of work in description logics (DL) that attempts to represent class properties at the meta-level [4,11]. For example, to express the constraint "every existing customer subscribes to at least one service from one of the networks: Vodafone, Orange, one2one and BTcellnet" in Classic [3], we write the following:

$$\begin{aligned} customer \sqsubseteq (&AND\ ATLEAST(1, service) \\ &(ALL\ network \\ &(\textit{ONE-OF}\ Vodafone, Orange, one2one, BTcellnet))) \end{aligned}$$

Here, no variables representing the instances are present, hence the expression is easier to understand. While DLs offer an effective framework for representing concept-based knowledge, when used to represent the extracted constraints, they suffer from several limitations. First, since DLs are designed as concept languages, most of them do not have a sufficient range of constructs to express constraints. For example, `BR2` given earlier is hard to be expressed in most DLs. Second, though most DLs have efficient procedures for deciding subsumption and satisfiability properties, many constraint analysis problems encountered in the reverse engineering of constraints cannot be reduced to subsumption and satisfiability in DLs.

To represent constraints in such a way that they are simple to comprehend and easy to analyse, we have proposed a language called $\mathcal{BRL}$ [9]. $\mathcal{BRL}$ is based on predicate logic and represents constraints at the meta-level. The basic term used in $\mathcal{BRL}$ is *structure*, which represents a class of objects in the real world. For example, `service(network, freeTime)` is a structure. A structure can be nested, i.e. the components of a structure can be a structure itself. For example, `order(customer, service(network, freeTime), recommender)` is a structure too. Structures are restricted to be acyclic, that is, a structure can not be the component of itself, either directly or indirectly. To refer to the components of a structure in $\mathcal{BRL}$, we introduce the term *path expression*. A path expression has the form $S_1.S_2 \ldots S_{m-1}.S_m$,

where $S_1, \ldots, S_m$ ($m \geq 1$) are structure names and the dot "." stands for the *component-of* relationship between the structures appearing on either side of it. S_1 is called the *source* and S_m the *target*. For example, `order.customer` is a path expression which specifies that `customer` is a component of `order`.

Different from a general predicate language [1,16,21], $\mathcal{BRL}$ does not allow arbitrary predicates to be used in constructing atomic formulas. Instead, it provides only a set of pre-defined predicates to capture the semantics which can realistically be expected from the reverse engineering of constraints. For example, `BR1` given earlier is represented in $\mathcal{BRL}$ as:

$$ENUMERATE(freeTime, \{300, 600\})$$

where "ENUMERATE" is one of the pre-defined predicates allowed in $\mathcal{BRL}$. The usual logic connectives, e.g., $\wedge, \vee, \neg$ and $\rightarrow$ may be used to express more complex constraints. For example, to express the constraint

```
BR3 The company is allowed to provide BTcellnet service
    for up to 10000 customers.
```

we use the following $\mathcal{BRL}$ formula

$$EQUAL(network, btcellnet) \rightarrow ATMOST(customer, 10000)$$

The full syntax of $\mathcal{BRL}$ is given in Appendix A. For a detailed description of $\mathcal{BRL}$, the reader is referred to [9].

In the rest of the paper, we will use the following set of constraints to explain our algorithm. These constraints are assumed to be extracted from a hypothetical application used by a mobile phone service promoter (Appendix B describes these constraints in English).

Example 1. A set of extracted constraints:

```
C1.  ENUMERATE(network,
           {Vodafone, Orange, one2one, Btcellnet})
C2.  ENUMERATE(freeTime, {300, 600})
C3.  EQUAL(service.freeTime, 600) ->
     EQUAL(service.network, Vodafone)
C4.  EQUAL(order.service.network, BTcellnet) ->
     ATMOST(order.customer, 10000)
C5.  ENUMERATE(customer.status,
           {current, temporary, historic, written-off})
C6.  ATMOST(order.customer, 1, order.recommender)
C7.  SUBSUME(customer, recommender)
C8.  ATMOST(order.customer, 3, order.service)
```

All the constraints are defined for the following structure:

```
order(customer(id, name, status),
      service(network, freeTime),
      recommender(id, name, status))
```

3 Related Constraints

We now explain what we mean by related constraints. Essentially, we are interested in a set of constraints that are related through shared structure(s), i.e. the constraints that involve the same structure(s) in their expressions. The ability to identify this type of related constraints is important in understanding the constraints extracted from legacy systems. This is because from any non-trivial applications, we can expect a large number of constraints to be extracted. Thus, it is essential that we are able to comprehend the extracted constraints selectively. The structure(s) involved in the constraints provide a plausible "point of focus" for such selective comprehension.

Conceptually, the relationship between a set of structures and a set of constraints can be illustrated as a *Structure-Constraint* (S-C) graph. For example, the S-C graph for Example 1 is shown in Figure 1, where a rectangle vertex stands for a structure and is labelled with the structure name, a round vertex stands for a constraint and is labelled with the constraint identifier. There are two types of arc in an S-C graph:

- A structure-structure arc (S_i, S_j) represents that structure S_j is a component of structure S_i. For example, the arc $(service, network)$ in Figure 1 specifies that `network` is a component of `service`.
- A constraint-structure arc (C_i, S_j) represents that the structure S_j is restricted by constraint C_i. Note that a single constraint may be connected to several structures, depending on the path expression(s) involved in the constraint. For example, C_5 is connected to both `customer` and `status` in Figure 1 because it involves `customer.status` in its expression. A constraint-structure arc (C_i, S_j) is also marked with a *weight* to indicate S_j's position in the path expression. For example, $(C_5, status)$ and $(C_5, customer)$ in Figure 1 are marked, respectively, with weights 0 and 1, indicating that `status` is the target structure and `customer` is the next structure (parent of `status`) in the path expression involved in C_5. The weights marked on constraint-structure arcs may be considered as representing the strength of constraints on structures: the smaller the weights, the stronger the relevance.

With an S-C graph, we can define and determine related constraints. Depending on how many structures are involved and whether their components are to be considered, we have three types of related constraint. The first type is the simplest and involves a single structure and does not consider its components. That is, given S_i, the set of related constraints, $Rel(S_i)$, are the constraints that are connected to S_i with 0-weighted arcs. For example, $Rel(customer) = \{C_4, C_6, C_7, C_8\}$ as they are the only constraints that are connected to `customer` with 0-weighted arcs. We call the constraints related in this way the 1-S-1-D related constraints.

The second type generalises the first by extending the relatedness to the structure's components. That is, given S_i and depth requirement n, the set of related constraints, $Rel(< S_i, n >)$, are the constraints that are connected to S_i with an arc whose weight is less than n. For example, $Rel(< customer, 2 >) = \{C_4, C_5, C_6, C_7, C_8, \}$ since they are the ones that are connected to `customer` with either 0- or 1-weighted arcs. We call the constraints related in this way the 1-S-n-D related constraints.

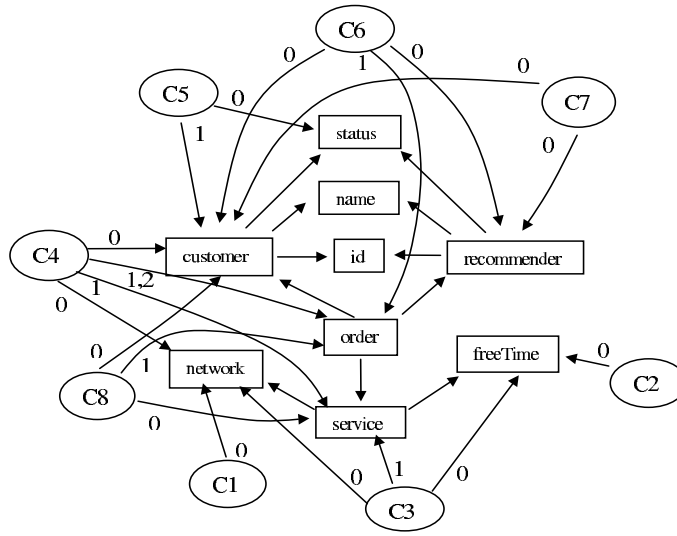


Figure 1. The S-C Graph for Example 1

The third type is a generalisation of the second by allowing multiple structures to be considered. Given $S_1, \ldots, S_n$ and a set of corresponding depth requirements, the set of related constraints, $Rel(< S_1, k_1 >, \ldots, < S_n, k_n >)$, is the intersection of $Rel(< S_1, n_1 >), \ldots, Rel(< S_k, n_k >)$. For example, $Rel(< customer, 2 >, < service, 2 >) = Rel(< customer, 2 >) \cap Rel(< service, 2 >) = \{C_4, C_5, C_6, C_7, C_8, \} \cap \{C_3, C_4, C_8\} = \{C_4, C_8\}$. We call the constraints related in this way the n-S-n-D related constraints.

>From the above definitions it can be seen that the task of determining a set of related constraints is equivalent to the traversing of the corresponding S-C graph for a subgraph that contains the relevant constraint vertices. In the following section, we introduce an algorithm to perform this task.

4 An Algorithm for Determining Related Constraints

We assume that the extracted constraints are already expressed in $\mathcal{BRL}$, and the S-C graph for the constraints and associated structures has already been constructed. [1] Our proposed algorithm consists of 3 steps. The first step normalises the S-C graph, the second step identifies the related constraints for a given criteria from the normalised S-C graph, and the last step derives additional related constraints by performing some logical deductions over the S-C graph.

[1] Constructing an S-C graph for a given set of constraints and structures is straightforward.

4.1 Normalising S-C Graph

The purpose of this step is to complete the original S-C graph, constructed from the set of constraints and structures extracted from legacy systems, with some "missing" constraint-structure arcs. This is necessary because in the original S-C graph, it is possible that a constraint that is related to a structure may not actually be connected to that structure. For example, according to Figure 1, $Rel(< order, 2 >) = \{C_4, C_6, C_8\}$. This is incomplete as C_7 is also related to $Rel(< order, 2 >)$. The reason for $(C_7, order)$ to be missing from the original S-C graph is that C_7 is expressed in terms of `recommender` and `customer`, and their parent structure (`order`) is not explicitly named in the path expression. The algorithm given below is used to normalise the original S-C graph, so that a constraint is always connected to the structure if it is related to that structure.

Algorithm 1. NORMALIZATION(G)

1. For every constraint-structure arc $(e \leftarrow (C, S)) \in G$
2. If S is the source of a path expression in C
3. $k \leftarrow \mathbf{WEIGHT}(e)$
4. $k \leftarrow k + 1$
5. P ← Find the parent structures of S
6. For every $S' \in P$
7. Add an arc (C, S') *to G with weight k*
8. $S \leftarrow S'$*; return to step 4*
9. Endfor
10. Endif
11. Endfor
12. Return G

For each constraint C, the algorithm looks for the *source* structure S of each path expression in C and assigns the weight of (C, S) to k. If S has parent structures, we connect C to those structures and mark the arcs with weight $k + 1$. This process is recursively performed until no parent structure can be found. For example, consider C_5 in Figure 1. The *source* of its path expression `customer.status` is `customer` and the weight of $(C_5, customer)$ is 1. Since `order` is the parent structure of `customer`, we add arc $(C_5, order)$ with weight 2 to the S-C graph. The normalised S-C graph for Example 1 is shown in Figure 2.

4.2 Identifying Related Constraints

This step of the algorithm is to traverse the normalised S-C graph to identify a subgraph which includes only the relevant structure(s) and constraints. Since the last step of our algorithm has extended the original S-C graph with all the missing constraint-structure arcs, looking for the related constraints for given structure(s) becomes straightforward. The following algorithm shows how 1-S-n-D related constraints are identified. The input of the algorithm is a normalised S-C graph G and a pair $< S, n >$, the output is a subgraph G_1 which includes related constraints for $< S, n >$.

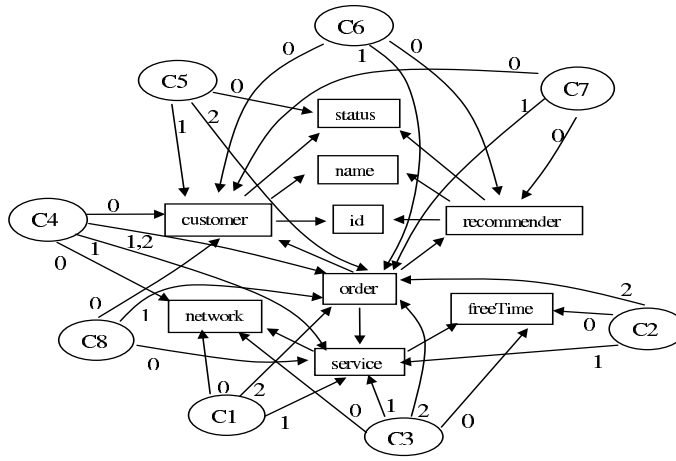


Figure 2. The Normalised S-C Graph for Example 1

Algorithm 2. IDENTIFICATION $(G, < S, n >, G_1)$

1. $G_1 \leftarrow \{S\}$
2. $CS \leftarrow$ *Find a set of constraint-structure arcs through which S*
/ *is connected to a constraint*
3. *For every* $(e \leftarrow (C, S)) \in CS$
4. *If* **WEIGHT***(e)* $< n$
5. $G_1 \leftarrow G_1 \cup \{C\} \cup \{e\}$
6. *EndIf*
7. *EndFor*
8. *Return* G_1

The following example illustrates how $Rel(< order, 2 >)$ is computed using the algorithm. We shade a vertex and dash an arc to indicate that they are included in the subgraph. The algorithm first locates vertex `order` in the S-C graph shown in Figure 2 and marks it. Then the algorithm looks for constraints that are connected to `order` and they are $C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8$. Among them, only C_4, C_6, C_7 and C_8 are connected to `order` with arcs whose weights are less than 2, so we mark them and the corresponding arcs. Figure 3 shows the result of performing IDENTIFICATION for $Rel(< order, 2 >)$.

The above algorithm directly applies to 1-S-1-D related constraints too. For n-S-n-D related constraints we can execute the above algorithm for n times, one for each structure, and then intersect the results.

4.3 Deducing Implicit Constraints

The set of related constraints derived by the previous steps of our algorithm may not be complete in a sense that there might be some other constraints which are not explicitly

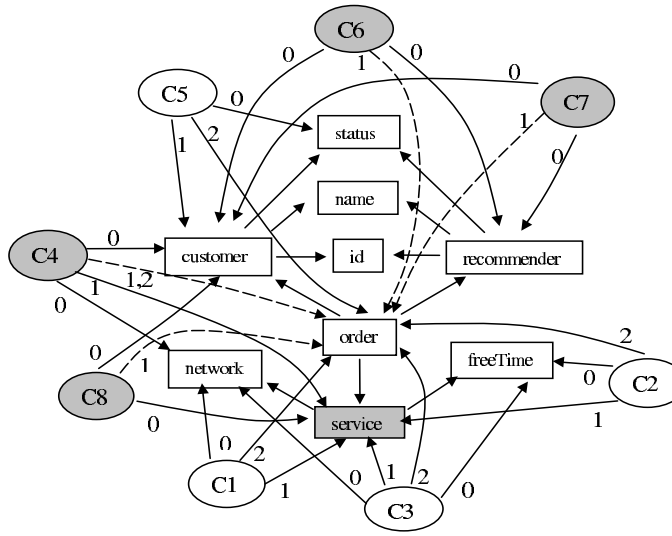


Figure 3. The Related Constraints for $< order, 2 >$

available in the S-C graph, but are deducible from the ones present in the graph. For example, from C_7 and C_8, it is possible to deduce the following:

$$ATMOST(order.recommender, 3, order.service)$$

which is clearly a member of $Rel(< order, 2 >)$. We argue that including implicit constraints in the computation of related constraints is beneficial. This is because while these implicit constraints may be seen as completely redundant from implementation point of view, they can help users, business users in particular, to comprehend the extracted constraints as a whole.

This third step of our algorithm is responsible for deducing the constraints that are implied by the known ones. What can be deduced depends on the type of constraint and the available axioms. For the constraints expressible in $\mathcal{BRL}$, we have introduced 14 axioms (see Appendix C). For example, the above constraint is derived by using Axiom 13. We classify these axioms into two categories: *transitivity axioms*, e.g. Axiom 2, which express the transitivity property of constraints, and *inheritance axioms*, e.g. Axiom 7, which express the fact that a structure inherits the constraints related to its super structures. These two groups of axioms need to be handled differently in deduction, though each individual axiom in the same group is handled similarly. We therefore need two algorithms: `TRANSITIVITY` for handling transitivity axioms and `INHERITANCE` for inheritance axioms.

In the following, we will give one algorithm, as an example, to show how to derive the constraints that are implicitly related to given structure(s) using Axiom 2. Algorithms used for handling other axioms are of the similar flavour, but will not be given here due to space limitation. One way to carry out the task is to apply the axioms to

all extracted constraints first and then use the algorithm IDENTIFICATION to determine which deduced constraints are actually relevant. However, this approach can be time-consuming because the set of constraints that might be deduced can be large. Our algorithms do not generate all the implicit constraints, but compute the ones that are relevant to the concerned structure(s) only. This is explained in the following algorithm for handling Axiom 2:

Algorithm 3. SUBSUME$(G, G_1 = Rel(< S, n >))$
1. For every constraint vertex $C_1 \in G_1$
2. If C_1 *is a* $SUBSUME(P_1, P_2)$ *constraint*
3. $S_1 \leftarrow$ **SUBSUMING**(S, n, P_2, G)
4. $S_2 \leftarrow$ **SUBSUMED**(S, n, P_1, G)
5. $G_2 \leftarrow$ **LCOMPOSE**(S, S_1)
6. $G_3 \leftarrow$ **RCOMPOSE**(S, S_2)
7. EndIf
8. $G_1 \leftarrow G_1 \cup G_2 \cup G_3$
9. EndFor
10. Return G_1

The input to the algorithm is a normalised S-C graph G and a set of related constraints $G_1 = Rel(< S, n >)$ which is derived by the `IDENTIFICATION` step. The output of the algorithm is G_1 augmented with the deduced constraint vertices and corresponding arcs. The algorithm works as follows. For every $C_1 \in G_1$ of the form $SUBSUME(P_1, P_2)$, the algorithm derives two sets of path expressions, S_1 and S_2, using the `SUBSUMING` and `SUBSUMED` functions respectively. S_1 contains the path expressions that transitively subsume P_2 and S_2 contains the path expressions that are transitively subsumed by P_1. It is necessary to create these two sets of path expressions because the structure (S) to which the deduced constraints must relate may appear in P_1, P_2 or both, and we must consider the two cases separately. Note the difference between a conventional transitive closure derivation and the one we described here. As we must consider the relevant structures involved in the constraints, our process is more complicated. The use of S in the `SUBSUMING` and `SUBSUMED` functions is to ensure that the constraints derived using Axiom 2 are always related to S. Once S_1 and S_2 are obtained, G_1 is augmented new constraints vertices and corresponding arcs composed with `LCOMPOSE` and `RCOMPOSE` functions.

4.4 Time Complexity of the Algorithm

In this section, we analyse the time complexity of the proposed algorithm. The size of input, i.e., the S-C graph, is characterised by the following

n_1 – the number of structure vertices
n_2 – the number of constraint vertices

The overall cost of the proposed algorithm is polynomial. The cost is divided into three parts, each of which is associated with each step of the algorithm. We show the cost in Table 1

Table 1. Time Complexity

Step		Time Cost
NORMALIZATION		$O(n_1 * n_2)$
IDENTIFICATION	1-S-1-D, 1-S-n-D	$O(n_1 + n_1 * n_2)$
	n-S-n-D	$O(n * (n_1 + n_1 * n_2 + n_2^2))$
DEDUCTION	1-S-1-D, 1-S-n-D	$O(n_1 * n_2^3)$
	n-S-n-D	$O(n * (n_1 * n_2^3 + n_2^2))$

To normalise an S-C graph, the main effort is to add missing constraint-structure arcs to an S-C graph. Since the maximum number of constraint-structure arcs is made by pairing every structure vertex with every constraint vertex, we can at most add $n_1 * n_2$ number of constraint-structure arcs to the S-C graph. Thus, the worst case for the algorithm NOAMALIZATION to execute is $O(n_1 * n_2)$.

To identify 1-S-n-D related constraints, we need a maximum of n_1 steps to locate the concerned structure in the normalised S-C graph, and maximal $n_1 * n_2$ steps to determine if a constraint is connected to the structure or not. Thus the worst case for IDENTIFICATION to execute is $O(n_1 + n1 * n_2)$. This result applies to identifying 1-S-1-D related constraints as well. To identify n-S-n-D related constraints, however, we need an extra $O(n * (n_2^2))$ effort to find the intersection of n sets of related constraints.

To compute implicit constraints using the transitivity axioms for 1-S-n-D problems, we need a maximum of $(n_1 * n_2^2)$ steps to construct a *transitive closure* of path expressions for every single constraint in $Rel(< S_i, n >)$. So the worst case is $(n_1 * n_2^3)$, which happens when $Rel(< S_i, n >)$ has n_2 number of constraints. We have not given algorithms for deriving implicit constraints using the inheritance axioms or involving n-S-n-D problems due to space limitation, but their complexity is also polynomial [8].

5 An Example

We have implemented the algorithm in Prolog. In this section, we show how Example 1 given in Section 3 is executed by our implementation. The construction of the S-C graph is straightforward. Structure vertices are represented as a set of $structure(S)$ facts. For example,

```
structure(order)
```

expresses that `order` is a structure. The arcs connecting the structures are represented as a set of $composite(S, L)$ facts. For example,

```
composite(order, [customer, service, recommender])
```

expresses that `order` has `customer, service` and `recommender` as components. The set of extracted constraints are represented in the form of

$$constraint(ID, cons\_expr)$$

where ID is the constraint identifier, and $cons\_expr$ is the clausal form of the constraint expression in $\mathcal{BRL}$. For example, constraint C_3 is represented in our program as follows:

```
constraint(C3, [neg(equal, [service,freeTime], 600),
           pos(equal, [service,network], vodafone)])
```

The constraint-structure arcs are expressed as a set of $CS(C, S, weight)$ facts. For example, the arcs connecting C_3 to the three structures are expressed as the following facts:

```
CS(C3, service,1)
CS(C3, freeTime,0)
CS(C3, network,0)
```

S-C Graph Normalization. The first step of our algorithm is to add the missing constraint-structure arcs to the S-C graph. Given the S-C graph for Example 1, the execution of

```
?- normalization.
```

asserts the following facts into the system:

```
CS(C1, service, 1)
CS(C1, order, 2)
CS(C2, service, 1)
CS(C2, order, 2)
CS(C3, order, 2)
CS(C5, order, 2)
CS(C7, order, 1)
```

Identifying Related Constraints. According to the type of related constraint requested by the user, the system will return the corresponding set of constraints. In the following, we show the results for 3 requests. The first asks for $Rel(network)$, a set of 1-S-1-D related constraints. The execution of

```
?- identification(network, 1, C).
```

results in the following set of constraints to be returned, which are all connected to `network` with 0-weighted arcs.

```
C=constraint(C1,[pos(enumerate,[order,service,network],
   [vodafone,orange,one2one,btcellnet])])
C=constraint(C3,[neg(equal,[order,service,talkTime],600),
   pos(equal,[order,service,network], vodafone)])
C=constraint(C4,[neg(equal,[order,service,network],
    btcellnet),pos(atmost,[order,customer],10000)])
```

The second request asks for $Rel(< order, 2 >)$, a set of 1-S-n-D related constraints.

```
?-identification(order, 2, C).
```

The constraints identified are those which are connected to `order` with 0- or 1-weighted arcs.

```
C = constraint(C4,[neg(equal,[order,service,network],
        btcellnet),pos(atmost,[order,customer],10000)])
C = constraint(C6,[pos(atmost,[order,customer],1,
    [order,recommender])])
C = constraint(C7,[pos(subsume,[order,customer],
    [order,recommender])])
C = constraint(C8,[pos(atmost,[order,customer],3,
    [order,service])])
```

The third request asks for $Rel(< recommender, 2 >, < service, 1 >)$, a set of n-S-n-D related constraints:

```
?-identification([recommender, service],[2,1], C).
```

Since there is no constraint that is connected to both `recommender` and `service`, the system returns nothing.

Deducing Implicit Constraints. This step is to infer the constraints implied by known ones. The system starts with related constraints computed by the last step, and looks for implicit constraints for the given structure(s) using the set of axioms given in Appendix C. For example, the execution of

```
?-deduction(order, 2, C).
```

computes the implicit constraints for $Rel < order, 2 >$ and returns nothing because no constraint can be deduced in this case. The execution of

```
?-deduction([recommender, service],[2,1], C).
```

on the other hand, derives one additional constraint for $Rel(< recommender, 2 >, < service, 1 >)$

```
C = constraint(C9, or([pos(atmost,[order,recommender],3,
    [order,service])]))
```

6 Conclusion and Future Work

In this paper we have described an algorithm for determining the set of constraints that are related to each other with respect to a given class or classes of business objects. We assume that the constraints are extracted from legacy systems, and our method is to represent such constraints in a meta-level predicate language, and to derive the related constraints by a mixture of logical deduction and path expression manipulation. We have also shown that the proposed algorithm has a polynomial time complexity, and

thus it can be expected to scale to constraints extracted from non-trivial applications. The analysis technique we proposed here can help users to understand the extracted constraints selectively by localising the constraints of interests.

While the paper is focused on the analysis of the relatedness property for constraints, it is possible to analyse other properties too. For example, we may wish to determine whether there exists a conflict among the constraints. We plan to investigate other forms of analysis in the future. We will also test the algorithm on the constraints extracted from real-world applications.

Acknowledgement

This work is part of the BRULEE project which is funded by Grant GR/M66219 from the U.K. Engineering and Physical Sciences Research Council.

References

1. N. Bassiliades and P. M. D. Gray. CoLan: A Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering(DKE)*, 14(3):203–249, 1994.
2. A. Borgida. Description Logics in Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.
3. A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. Classic: A Structural Data Model for Objects. In *SIGMOD Conference*, pages 58–67, 1989.
4. R. J. Brachman. An Overview of KL-ONE knowledge System. *Cognitive Science*, 9(2):171–216, 1985.
5. R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database. *Data and Knowledge Engineering*, 12:107–142, 1994.
6. J. A. Chudziak, H. Rybinski, and J. Vorbach. Towards a Unifying Logic Formalism for Semantic Data Models. In *Proceedings of 12th International Conference on the Entity-Relationship Approach*, pages 492–507, 1993.
7. S. Embury and J. Shao. Assisting the Comprehension of Legacy Transactions. In *Proceedings of Second International Workshop of Data Reverse Engineering*, 2001.
8. G. Fu. Comprehension of Constraint Business Rule Extracted from Legacy Systems. Brulee project techniqual report, Department of Computer Science, Cardiff University, 2000.
9. G. Fu, J.Shao, S.M. Embury, and W.A. Gray. Representing Constraint Business Rules Extracted from Legacy Systems. Submitted for Publication, Department of Computer Science, Cardiff University, 2002.
10. P. Grefen and P. Apers. Integrity Control in Relational Database System - An Overview. *Data and Knowledge Engineering*, 10:187–223, 1993.
11. R.M. Gregor. The Evolving Technology of Classification-Based Knowledge Representation Systems. In J. Sowa, editor, *Principle of Semantic Networks*. 1991.
12. D. Hay and K. Healy. Defining Business Rules - What are They Really. Guide Business Rule Project Report, 1996.
13. H. Huang, W. Tsai, and et al. Business Rule Extraction from Legacy Code. Technical Report TR 95-034, Department of Computer Science and Engineering, University of Minnesota, 1995.
14. P. McBrien, A.H. Seltveit, and B. Wangler. Rule Based Specification of Information Systems. In *Proceedings of the International Conference on Information Systems and Management of Data*, pages 212–228, 1994.

15. M. Morgenster. Constraint Equations: Declarative Expression of Constraints with Automatic Enforcement. In *Proceedings of 10th International Conference on Very Large Data Bases*, pages 291–300, 1984.
16. ORM. UML 2.0 OCL RFP. Available online `http://www.omg.org/techprocess/meetings/schedule/uml_2.0_ocl_rfp.html`, 2002.
17. P. Ruttan. Recovering Business Rules to Enable Legacy Transformation. Technical report, Netron Inc., 1999.
18. J. Shao, X. Liu, G. Fu, S. Embury, and W. Gray. Querying Data-Intensive Programs for Data Design. In *proceeding of 13th International Conference for Advanced information System Engineering (CAiSE'01)*, 2001.
19. J. Shao and C. Pound. Reverse Engineering Business Rules from Legacy System. *BT Journal*, 17(4), 1999.
20. V. C. Storey, H. L. Yang, and R. C. Goldstein. Semantic Integrity Constraints in Knowledge-Based Database Design Systems. *Data and Knowledge Engineering(DKE)*, 20(1):1–37, 1996.
21. S. Urban. ALICE: An Assertion Language for Integrity Constraint Expression. In *Proceedings of the 13th Annual Conference on Computer Software and Application*, pages 292–299, 1989.

Appendix A $\mathcal{BRL}$ Grammar

$$
\begin{aligned}
<cons\_expr> ::= {} & <cons\_expr> \wedge <cons\_expr> \mid <cons\_expr> \vee <cons\_expr> \mid \\
& \neg <cons\_expr> \mid <cons\_expr> \rightarrow <cons\_expr> \mid \\
& \forall x <cons\_expr> \mid \exists x <cons\_expr> \mid \\
& TYPEOF(<path\_expr>, <built\_in>) \mid \\
& ENUMERATE(<path\_expr>, \{<Object> [, <Object>]*\}) \mid \\
& GREATHRAN(<path\_expr>, <arith\_expr>) \mid \\
& EQUAL(<path\_expr>, <arith\_expr>) \mid \\
& SUBSUME(<path\_expr>, <path\_expr>) \mid \\
& DISJOINT(<path\_expr>, <path\_expr>) \mid \\
& ONEOF(<path\_expr>, \{<path\_expr> [, <path\_expr>]*\}) \mid \\
& ATMOST(<path\_expr>, <positive\_integer>) \mid \\
& ATLEAST(<path\_expr>, <positive\_integer>) \mid \\
& ATLEAST(<path\_expr>, <positive\_integer>, <path\_expr>) \mid \\
& ATMOST(<path\_expr>, <positive\_integer>, <path\_expr>) \\
<arith\_expr> ::= {} & <path\_expr> \mid <object> \mid sum(<path\_expr>) \mid \\
& count(<path\_expr>) \mid ave((<path\_expr>) \mid max((<path\_expr>) \mid \\
& min(<path\_expr>) \mid <arith\_expr> + <arith\_expr> \mid \\
& <arith\_expr> - <arith\_expr> \mid <arith\_expr> \\
& * <arith\_expr> \mid <arith\_expr> / <arith\_expr> \\
<path\_expr> ::= {} & <structure> [. <structure>] * \\
<built\_in> ::= {} & real \mid integer \mid boolean \mid string \mid char
\end{aligned}
$$

$< structure > ::= < symbol >$
$< object > ::= < symbol >$

Appendix B Constraints in English (for Example 1)

C1. The company uses the following networks: Vodafone, Orange, one2one and BTcellnet.
C2. The company offers 2 categories of free talk: 300 minutes and 600 minutes.
C3. Only Vodafone customers are entitled to 600 minutes free talk.
C4. The company is allowed to provide BTcellnet service to up to 10000 customers.
C5. The status of a customer is one of following: current, temporary, historic or written-off.
C6. Every customer has no more than one recommender.
C7. A recommender must be an existing customer.
C8. Every customer can subscribe to at most 3 services.

Appendix C: Axioms for Deriving Implicit Constraints

Axiom 1 $ONEOF(P, \{P_1, ..., P_m\}) \Rightarrow SUBSUME(P, P_1) \wedge \ldots \wedge SUBSUME(P, P_m)$
Axiom 2 $SUBSUME(P_1, P_2) \wedge SUBSUME(P_2, P_3) \Rightarrow SUBSUME(P_1, P_3)$
Axiom 3 $ATLMOST(P_1, n, P_2) \wedge ATMOST(P_2, m, P_3) \Rightarrow ATMOST(P_1, n * m, P_3)$
Axiom 4 $EQUAL(P_1, P_2) \wedge EQUAL(P_2, P_3) \Rightarrow EQUAL(P_1, P_3)$
Axiom 5 $GREATHAN(P_1, P_2) \wedge GREATHAN(P_2, P_3) \Rightarrow GREATHAN(P_1, P_3)$
Axiom 6 $GREATHAN(P_1, P_2) \wedge EQUAL(P_2, P_3) \Rightarrow GREATHAN(P_1, P_3)$
Axiom 7 $SUBSUME(P_2, P_1) \wedge ENUMERATE(P_2, \{O_1, \ldots O_m\}) \Rightarrow ENUMERATE(P_1, \{O_1, \ldots, O_m\})$
Axiom 8 $SUBSUME(P_2, P_1) \wedge GREATHAN(P_2, A) \Rightarrow GREATHAN(P_1, A)$
Axiom 9 $SUBSUME(P_2, P_1) \wedge EQUAL(P_2, A) \Rightarrow EQUAL(P_1, A)$
Axiom 10 $SUBSUME(P_2, P_1) \wedge TYPEOF(P_2, b) \Rightarrow TYPEOF(P_1, b)$
Axiom 11 $SUBSUME(P_2, P_1) \wedge ATMOST(P_2, m) \Rightarrow ATMOST(P_2, m)$
Axiom 12 $SUBSUME(P_2, P_1) \wedge ATLEAST(P_2, n, P_3) \Rightarrow ALTEAST(P_1, n, P_3)$
Axiom 13 $SUBSUME(P_2, P_1) \wedge ATMOST(P_2, m, P_3) \Rightarrow ATMOST(P_1, m, P_3)$
Axiom 14 $SUBSUME(P_2, P_1) \wedge DISJOINT(P_2, P_3) \Rightarrow DISJOINT(P_1, P_3)$

Retrieval Performance Experiment with the Webspace Method

Roelof van Zwol and Peter M.G. Apers

University of Twente,Department of Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
{zwol,apers}@cs.utwente.nl

Abstract Finding relevant information using search engines that index large portions of the World-Wide Web is often a frustrating task. Due to the diversity of the information available, those search engines will have to rely on techniques, developed in the field of information retrieval (IR).
When focusing on more limited domains of the Internet, large collections of documents can be found, having a highly structured and multimedia character. Furthermore, it can be assumed that the content is more related. This allows more precise and advanced query formulation techniques to be used for the Web, as commonly used within a database environment. The Webspace Method focuses on such document collections, and offers an approach for modelling and searching large collections of documents, based on a conceptual schema.
The main focus in this article is the evaluation of a retrieval performance experiment, carried out to examine the advances of the webspace search engine, compared to a standard search engine using a widely accepted IR model. A major improvement in retrieval performance, measured in terms of recall and precision, up to a factor two, can be achieved when searching document collections, using the Webspace Method.

1 Introduction

Over time the Internet has grown into an ever more tangled resource of information. The state-of-the-art means for finding information are text-based search engines like Alta-Vista and Google, hierarchical indexes and directories like Yahoo!. These search engines have to base their tools solely on information retrieval techniques, due to the unstructured nature of the Internet. The diversity, irregularity, and incompleteness of the data involved, make it impossible to use database technology at this global level. Besides that the data has the tendency to change rapidly over time.

However, when focusing on smaller domains of the WWW, database techniques can be invoked successfully to enhance the precision of the search process. On such domains, large collections of documents can be found, containing related information. Although the conditions are better, one still has to deal with the semi-structured and multimedia character of the data involved. The *Webspace Method* [13] focuses on such domains, like Intranets, digital libraries, and large web-sites.

The goal of the *Webspace Method* is to provide sophisticated search facilities for web-based document collections, using existing database techniques. To achieve this,

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 150–165, 2002.

three stages are identified for the Webspace Method. The first stage deals with conceptual modelling of web-data. During the second stage, both conceptual and multimedia meta-data is extracted from the document collection. Finally, in the last stage, the Webspace Method introduces a new approach to query a collection of web-based documents.

The key behind the Webspace Method is the introduction of a *semantical level*, which provides a conceptual description of the content of a webspace. This semantical level consist of a collection of concepts, which are defined in an *object-oriented* schema. The object-oriented schema is also referred to as the *webspace schema*. The second level of the Webspace Method is called the *document level*. This level is formed by the web-based document collection. Each document on the document level of a webspace consists of a view, corresponding to (a part of) the webspace schema, to assure the relation between the semantical and physical levels of the Webspace Method. The webspace schema defined during the first stage, is also used to extract the relevant meta-data during the second stage, and to formulate queries over a *webspace*. A webspace is formed by both the semantical and the document level.

Revolutionary within the scope of search engines and query formulation over document collections is that the Webspace Method allows a user to integrate (combine) information stored in several documents in a single query. At this point traditional search engines, e.g. Alta-Vista and Google, are only capable of querying a single document at a time. As result, the query will return the requested conceptual information as a view on the webspace schema, rather than returning a collection of relevant document URLs.

The main focus of this article forms the retrieval performance experiment. To evaluate the retrieval performance of the webspace search engine, and in particular the contribution of conceptual modelling to the retrieval process, a retrieval performance experiment is carried out. It measures the increase in performance of the webspace search engine, compared to the traditional way of querying a collection of documents, using a standard search engine. For this purpose, the standard search engine is equipped with the same IR model for text retrieval as the webspace search engine. A third search engine is also evaluated, which implements a fragmented variant of the webspace search engine. The motivation for this fragmented variant, along with a discussion of the implementation is given in Section 4.

The evaluation method used for the experiment is related to the *TREC test collection*, as described in Section 3.2. The second part of this article describes the experimental setup, which provides insight in the test collection, setup for this experiment. It is based on the 'Lonely Planet' web-site. The original web-site can be found at [12], and contains 6500 documents with structured information about destinations all over the world, divided in regions, countries, and cities. Furthermore, it contains a large collection of postcards, send in by travellers, containing the experiences of those travellers, while staying at a certain destination.

The results of the experiment show a high increase in retrieval performance, measured in terms of *recall* and *precision*. On average, both webspace search engines perform up to a factor two better, than the standard search engine. Furthermore it proves that the search engines based on the Webspace Method are capable of finding information that can not be found by standard search engines, due to the conceptual model introduced by the Webspace Method.

Related Work

Modelling data on the web is an ongoing research area, where many research projects have been positioned. Closely related to our approach is the Araneus project [10] where also existing database technology is applied to the WWW. This project is also concerned with handling both structured and semi-structured documents. The main difference with our approach is that we aim at combining concept-based search with content-based information retrieval, to come up with more precise query formulation techniques for data on the web. Others [1,3], like in XQuery [4] use the structure of an XML document as their model. This allows them to search for patterns and structure in the XML data, but does not allow them to formulate content based (IR) queries. In [7,8], about XIRQL and searching text-rich XML documents with relevance ranking, the focus is on XML and information retrieval. But these approaches do not use a conceptual model, and integrate the IR model only partially. Of course in the field of information retrieval, and multimedia databases many sophisticated models are proposed. We do not aim to come up with better IR techniques, but aim to combine existing IR techniques with conceptual modelling, using a database-oriented approach. For those interested in information retrieval and MM-DBMS, we refer to [2,5,6], where these matters are discussed.

Organisation

In the remainder of this article, the ideas behind the Webspace Method are explained in Section 2. Before discussing the experiment a short discussion of retrieval performance evaluation is presented in Section 3 to explain some of the backgrounds of the experimental setup, presented in Section 4. The results of the retrieval performance experiment are discussed in Section 5. Finally, we will come to the conclusions in Section 6.

2 Ideas behind the Webspace Method

In the introduction we already argued that database techniques cannot be applied straightforwardly to search for data on the Web, since the Web is not a database [11]. Therefore the Webspace Method aims at smaller domains of the Internet, where large collections of documents can be found containing related information. Such data can typically be found on large Intranets, web-sites and digital libraries. The Webspace Method for modelling and querying web-based document collections is developed for such collections. It combines conceptual search with content-based information retrieval to obtain more precise and advance query formulation techniques for web-data.

The Webspace Method consists of three stages: a modelling stage, a data-extraction stage, and a query stage. The modelling stage is used to define a webspace at both the semantical and document level. Next, during the indexing stage, meta-data is extracted from a webspace, and stored into the object server used by the webspace search engine. Finally, during the query stage, a webspace can be queried using the conceptual schema (webspace schema), which allows a user to formulate his information needs in terms of concepts defined in the webspace schema.

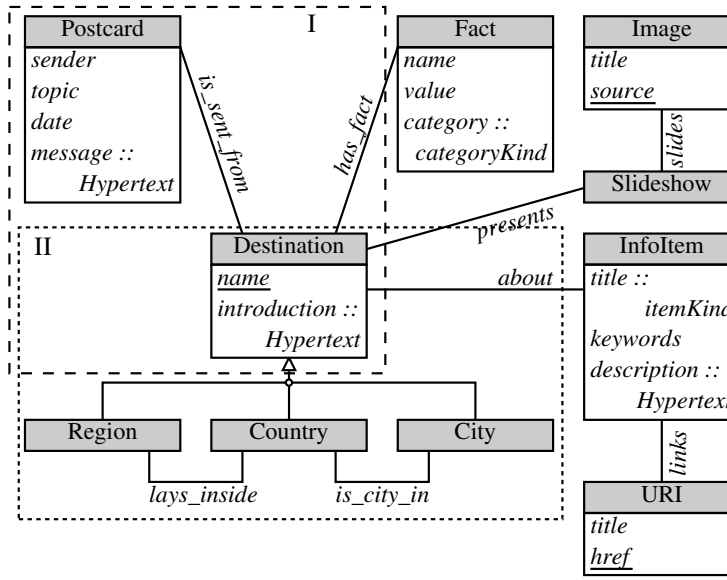


Figure 1. Webspace schema for the 'Lonely Planet' case-study.

2.1 Modelling a Webspace

For each webspace, it is possible to identify a finite set of concepts, which adequately describe the content of a webspace at a semantical level [14]. During the modelling stage of the Webspace Method, these concepts are modelled in an object-oriented schema, called the webspace schema. At the document level the content is stored in XML documents, or in a content DBMS. In either case the representation of the document, as presented to the end user, forms a materialised view. To create such a view, it requires that the author specifies a structure of the document, provides the document's content, as well as the default presentation.

Webspace Schema. Each concept defined for a webspace is referred to by a unique name. Furthermore, each concept should be defined in the webspace schema. Going towards the test collection used for the retrieval performance experiment, the webspace schema presented in Figure 1 is setup to semantically describe the content of the 'Lonely Planet' webspace.

It contains ten class definitions with their attributes and the associations between the classes. The underlined attributes form the uniques keys of a web-class. The attributes having their type displayed are of a multimedia class, which realises the integration of multimedia objects into the conceptual framework The dotted boxes, illustrate how the information contained in the XML documents at the document level of a webspace are spread over the different conceptual classes.

Materialised Views. Each document found on the document level forms a *materialised view* on the webspace schema. Thus, the dotted box containing the classes *Postcard* and *Destination* (Figure 1) forms a view on the webspace schema, since it describes only a part of the schema. All documents that contain information, related to (exactly) this part of the schema are materialising this view. Other documents might contain different materialised views, while the collection of materialised views is likely to have some overlap, as is the case for boxes I and II of Figure 1.

To derive a materialised view from the webspace schema, i.e. an XML-document, several steps will have to be taken in order to define the structure of the document explicitly. This procedure is described in more detail in [14] and implemented by the *webspace modelling tool*.

2.2 Indexing a Webspace

During the extraction stage of the Webspace Method the object server needed for querying a webspace is set-up. Depending on the webspace schema and the types of multimedia used for a webspace, daemons need to be administered in the Daemon Data Dictionary. A *daemon* is a program performing a single task on a web object. In general, when the object is a multimedia object, the daemon will try to extract relevant meta-data and store it in the object server. But also for the storage of conceptual objects daemons are used. A detailed description of the DDD and its role in the extraction stage is given in [15].

2.3 Querying a Webspace

The main contribution of the Webspace Method is that it provides a new approach for querying web-based document collections. The basis for this is formed by the webspace schema, which allows powerful query formulation techniques, as known within a database environment, to be invoked on the content of a webspace. The query formulation techniques, implemented by the webspace search engine, allow queries to be formulated, by using the webspace schema, over the content stored in one or more documents. Secondly, it allows users to directly extract the relevant parts of one or more documents as the result of a query, instead of a document's URL.

The webspace schema developed during the modelling stage of the Webspace Method is provided to the user, to specify his information need. To allow a user to formulate these relatively complex queries over a webspace a graphical user interface (GUI) is used, which guides the user through the query formulation process. The GUI provides a visualisation of the classes contained in the webspace schema, which is used to determine the query skeleton. Next, the constraints are formulated, and the user specifies the information that is returned as the result of the query.

Since the user specifies his information need by selecting concepts from the webspace schema, it can formulate queries that combine information originally stored in several documents, i.e. materialised views. Using the overlap between the materialised views, allows the webspace search engine to find information that can not be found by search engines that do not rely on a conceptual schema. Crucial for the success of the Webspace Method is the schema-based approach, but also the integration with the

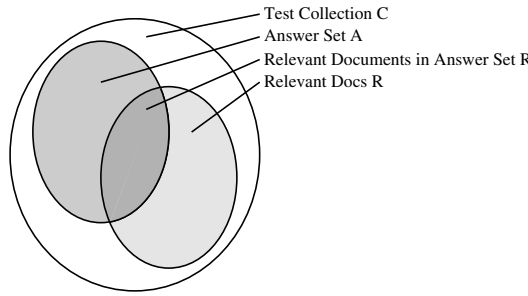


Figure 2. Information request

information retrieval, which allows us to benefit from the existing knowledge in that area.

3 Retrieval Performance Evaluation

Measuring the response time and usage of space of a system are normally the objectives, when carrying out a performance evaluation. In information retrieval, other metrics, besides time and space are of interest. A user's information need is formulated using a set of terms, and the result of a query contains inexact answers, where, based on the assumed relevance of a document, a ranking is produced. Therefore the main objective of a *retrieval performance evaluation* in the information retrieval is to measure the precision of the answer set given a specific query.

3.1 Retrieval Performance Measures

In the information retrieval a retrieval performance experiment is based on a *test reference collection* and one or more *evaluation measures*. The test reference collection consists of a collection of documents, a set of information requests (queries), and a set of relevant documents for each information request, provided by specialists.

The performance of a retrieval strategy, implementing a specific IR model, is measured by calculating the similarity between the set of documents found by the retrieval strategy, for each information request, and the set of relevant documents of the test reference collection.

The measures *recall* and *precision* are most frequently used, to measure the retrieval performance of an experiment, evaluating an IR system in batch mode. Figure 2 shows how these measures can be calculated for a specific information request.

Given an information request I, taken from the test reference collection and the set of relevant documents R for that specific information request, then $\mid R \mid$ refers to the number documents in this set. Let A be the answer set of documents found by the retrieval strategy for information request I. Then $\mid A \mid$ is the number of documents in the answer set. Further, let $\mid R_a \mid$ be the number of documents in the intersection of the sets R and A. The measures for recall and precision are then calculated as:

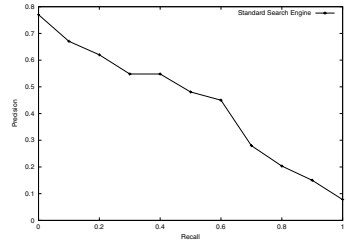


Figure 3. Precision at 11 standard recall levels

$$Recall = \frac{\mid R_a \mid}{\mid R \mid} \tag{1}$$

$$Precision = \frac{\mid R_a \mid}{\mid A \mid} \tag{2}$$

The 'perfect' IR system, would both have a recall and precision of 1. Unfortunately such systems do not exist (yet). Furthermore, basing the retrieval performance solely on these two measures would neglect the fact that the answer set of a specific information request is ranked. Retrieval strategies that rank the relevant documents higher should be rewarded for that. This is normally expressed in a *precision versus recall curve*. This curve is calculated by introducing 11 standard recall levels, for which the precision is calculated. This results in a curve, as shown in Figure 3.

To illustrate how the precision curve at 11 standard recall levels of the figure should be interpreted, consider the following. When the *top* 30% of the documents in the answer set is evaluated, the retrieval strategy has a precision of 0.5480. The graph shows the typical behaviour of IR systems. Increasing the recall of the answer set, will gradually cause a decrease in precision. Interpolation is used to determine the precision at recall level 0.

Alternatively, the average precision at given *document cut-off values* can be computed. This approach provides additional information with respect to the retrieval performance of the ranking algorithm. The result is a curve, which shows the average precision of the retrieval strategy when 5, 10, . . . , 100 documents are seen.

The recall-precision curves explained in this section are usually not computed over a single query, but over the entire set of information requests. This provides a good estimate of the overall performance of a retrieval strategy. In [2] it is pointed out that it is equally important to investigate the behaviour of a retrieval strategy for the individual queries.

Two reasons are brought up. First, the average precision might disguise important anomalies of the retrieval strategy. Second, when two retrieval strategies are compared it is interesting to see whether one of them outperforms the other for each query. In literature evaluation measures that provide this kind of information are referred to as *single value summaries*.

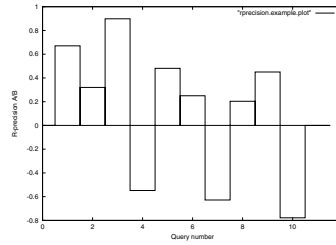


Figure 4. R-precision histogram, comparing the retrieval strategies A and B

The *average R-precision* is an example of such a measure. A single value of the ranking is computed by taking the precision at the R-th position in the ranking, where R is the total number of relevant documents for the current query. These measures can be plotted in R-precision histograms using the following rule to compare the retrieval history of the retrieval strategies A and B. Let $RP_A(i)$ and $RP_B(i)$ be the R-precision values of the corresponding retrieval strategies for the i-th query. The difference between these strategies is then simply calculated as:

$$RP_{A/B}(i) = RP_A(i) - RP_B(i) \tag{3}$$

An example R-precision histogram is given in figure 4. A positive precision means that strategy A outperformed strategy B and vice versa if the precision is negative.

Other measures which are used to emphasis certain aspects of the retrieval performance evaluation are discussed in [2,5].

3.2 Text REtrieval Conference

The Text REtrieval Conference, or shortly TREC, is an initiative of the National Institute of Standards and Technology (NIST) and the Information Technology Office of the Defense Advanced Research Projects Agency (DARPA). The goal of the conference is 'to encourage research in information retrieval from large text applications by providing a large test collection, uniform scoring procedures, and a forum for organisations interested in comparing their results'.

The ad-hoc task of the TREC test reference collection uses the following four basic types of evaluation measures: (1) *summary table statistics*, (2) *recall-precision averages*, (3) *document level averages*, and (4) *average precision histogram*

4 Experimental Setup

Evaluation of retrieval strategies for information retrieval used to be a difficult task, since good benchmarks and large test collections were missing. But today, the TREC collection among others, provides good experimental platforms for most IR systems. Unfortunately, the techniques used for the Webspace Method cannot be applied on such

test collections, due to the requirements of the conceptual model. To evaluate the retrieval performance of the Webspace Method a new test collection is built, which satisfies the conceptual constraints of the Webspace Method. For this purpose the document collection originally used for the 'Lonely Planet' case-study is used.

4.1 'Lonely Planet' Test Reference Collection

Like the test reference collections described in Section 3 the Lonely Planet test reference collection consists of a collection of documents, a set of information requests (topics), and a set of relevant documents for each information request, provided by specialists.

- **Document collection**. The Lonely Planet document collection consists of approximately 6500 documents, which describe destinations all over the world. For each destination, several documents exist, containing items like 'history', 'culture', 'activities', 'facts', and much more destination related information. A large subset of the document collection is formed by the postcard collection. A postcard contains some information about a destination, which is send in by a traveller, containing personal experiences with respect to the destination. Furthermore, for nearly every destination a slideshow is available, containing a series of pictures, which illustrate the typical characteristics of that destination. To get an impression of the document collection, please visit the original website, located at `http://www.lonelyplanet.com/`.
- **Topics**. The test collection contains 20 topics (queries) related to the Lonely Planet document collection. In Table 1 the description of such a topic is given.

Table 1. Description of Topic 5

TOPIC 5:
Search for cultural information about destinations, where the description contains terms like:"skin colour racism politics church".

Query terms	*Stemmed terms*
culture skin colour racism politics church	skin colour racism polite church

- **Relevance judgements**. The relevance judgements, i.e. the set with relevant documents, for each information request (topic), is determined by a *blind review pooling method*. For each topic, the set of documents found by the different search engines were collected and mixed. Twenty experts were asked to evaluate the (combined) retrieved document set for each topic.

4.2 Evaluation Measures

The primary evaluation of the experiment is based on the evaluation measures used for TREC. As discussed in Section 3.2, four classes of evaluation measures are used: (1)

the summary table statistics, (2) recall-precision averages, (3) document level averages, and (4) the average precision histogram. In the next section a discussion of the results of the experiments is given, based on this classification.

4.3 Runs

Last but not least, three runs were carried out on the 'Lonely Planet' test collection. Below a short description of each run (search engine) is given. For each topic, only the fifteen highest ranked documents were evaluated as the answer set of the search engine for a given topic. As a result the experts have to examine a set of documents, with the minimal size of 15 documents, if the three search engine produce the same answer set for a topic, and a maximum of 45 documents, if the answer sets are completely disjoint.

- **Standard search engine**. The standard search engine (SSE) is based on the well-known vector-space model. This model forms the basis for commonly used search engines on the WWW, like Google and Alta-Vista. Its interface allows a sequence of query terms to be formulated, which is used to find a ranking of relevant documents. For each topic, a set of query terms is defined, which represents that topic. The query terms for Topic 5 are given in Table 1.
- **Webspace search engine**. The webspace search engine (WSE), is of course based on the Webspace Method, and uses the webspace schema for formulation of the queries (topics) over the document collection. The text retrieval component of the WSE is based on exactly the same vector-space model as used for the standard search engine.
  Instead of indexing the entire document, only the Hypertext-fragments are indexed by the TextDaemon. The stemmed terms, given in Table 1 are the (stemmed) terms used by the WSE, to evaluate the relevance of the given Hypertext-objects. To be able to make a comparison between the search engines, the WSE only returns the document URLs containing the relevant information, instead of using the option of composing user-defined views.
- **Fragmented webspace search engine**. The fragmented webspace search engine (FWSE) is a variant of the webspace search engine, which uses a horizontally fragmented TextDaemon.
  Instead of building one term index for all Hypertext-objects, a separate term index is built for each conceptual attribute of type Hypertext. For instance, in case of the webspace schema presented in Figure 1, three different term indexes are built using the triggers: (1) Postcard.message.Hypertext, (2) Destination.introduction.Hypertext, and (3) InfoItem.description.Hypertext. The motivation behind the fragmentation is the following:
  a) The Webspace Method introduces a conceptual index, which exploits the *semantical structure* of a document. Thus the textual descriptions associated with a concept, will probably also contain semantically different terms.
  b) If no fragmentation is used, the less frequently occurring index terms related to a specific conceptual class and attribute have a relatively low *tfidf* if the same terms occur frequently in the Hypertext-objects associated with a different class and attribute. The fragmentation causes a correction in the *tfidf*.

c) This correction is only useful, if the query uses more than one query term when searching *Hypertext*-fragments, because a difference in the final ranking of the *Hypertext*-objects, can only occur if the difference between the *idf*-values of the terms, with and without fragmentation is large enough. The more terms are specified, the more likely it is that a difference in ranking will occur. In [9] some tests, also based on the 'Lonely Planet' case-study, are described, which provide detailed information on the implementation issues, and show that there is actually a change in ranking, when comparing the results of the FWSE with the WSE.

5 Experimental Results

The evaluation method provides statistics, which to a certain degree, are comparable with trends obtained from the statistical results from TREC. Below the four basic measures are discussed.

5.1 Summary table statistics

In Table 2 the *summary table statistics* for the three runs of the experiment are given. They provide a first indication of the performance of the different search engines.

Table 2. Summary table statistics

Total number of documents over all queries			
	SSE	**WSE**	**FWSE**
Answer set (A)	300	287	287
Relevant document set (R)	222	222	222
Relevant doc. in answer set(R_a)	104	187	188

The first row of the table provides information about the size of the answer set containing the retrieved documents for each of the search engines. The answer set of the SSE contains the maximum of 300 documents over the 20 given topics, while both the webspace search engines did not always return 15 relevant documents, due to the conceptual constraints of the query. This has a positive influence on the precision but reduces the recall, if the webspace search engines do not find all the relevant documents.

The total number of relevant documents (R), over the entire set of documents, for all topics is 222. The third row shows the set R_a, which gives a first indication of the performance of the search engines. The standard search engine found 104 relevant documents, where the webspace search engines have found 187 and 188 documents. This is almost twice the number of relevant documents found by the standard search engine.

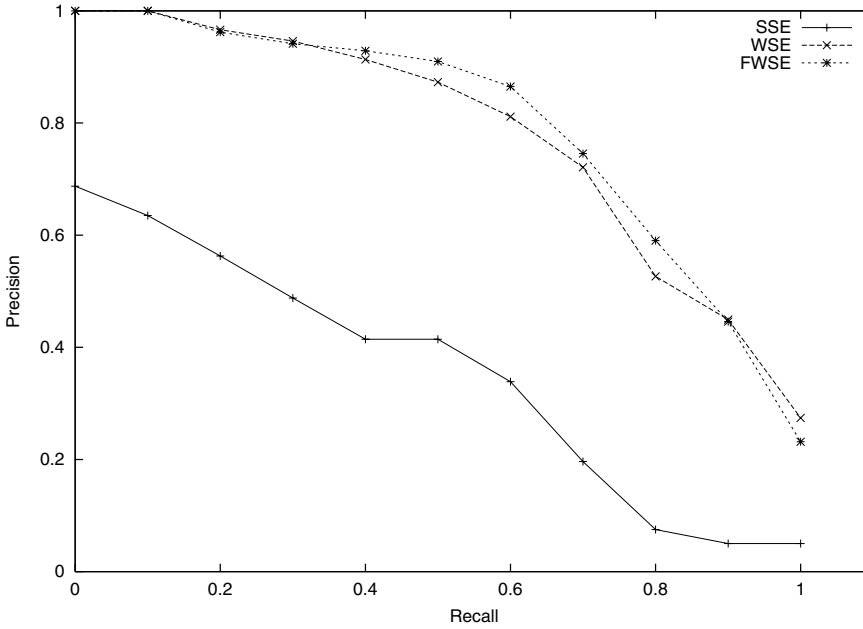


Figure 5. Interpolated recall-precision curve

5.2 Recall-Precision Averages

The summary table statistics only supply average statistics of a run, due to their set-based nature. More detailed information on the retrieval performance is normally expressed in precision versus recall. This also takes into account the evaluation of the ranking. Figure 5 shows the recall-precision curve. This curve computes the average performance over a set of topics, interpolated over 11 standard recall levels (0.0 to 1.0, with increments of 0.1). At standard recall level *i*, it shows the maximum precision of any recall level greater than or equal to *i*. The optimal IR system, will have a curve, equal to a horizontal line, at a precision of 1.

The recall-precision curve of the standard search engine starts at a precision of 0.69 and ends with a precision of 0.05 (see Table 3). Both webspace search engines start at the optimal precision (1.00), and end with a precision of 0.27, and 0.23. The resulting curves show that the performance of the search engines that use the Webspace Method have a much higher performance, than the standard search engine, using the same IR model. There is only a small increase in performance when the fragmented IR model is used in combination with the Webspace Method. This small increase in performance is caused by a better ranking of the relevant documents, rather than that different documents were found.

The values for the *non-interpolated average precision* for the three search engines show that the WSE and the FWSE, with their precisions of 0.7687 and 0.7806, respectively, cause a large improvement in the precision by almost a factor 2. At the same

Table 3. Precision measures

Recall versus precision				
Run	Average Precision	Initial Precision	Precision @ 100	Recall
SSE	0.3389	0.6873	0.0500	0.4684
WSE	0.7687	1.0000	0.2740	0.8423
FWSE	0.7806	1.0000	0.2316	0.8468

time, the average recall of the webspace search engines is also increased by a factor 1.8, compared to the standard search engine. Note that the non-interpolated average precision measure especially rewards systems, that rank relevant documents at a high position.

From these measures it is clear that the schema-based approach for querying document collections, in combination with the integration with information retrieval, as introduced by the Webspace Method, is responsible for the increase in retrieval performance.

5.3 Document Level Averages

The document level averages provide more insight in the quality of the ranking mechanism. Based on pre-determined document cut-off values, the average precision of the search engine is given after seeing N documents. In Table 4 document precision averages are given for all three runs, after retrieving x documents. It reflects the actual measured retrieval performance, from the user's point of view. The document precision average is calculated by summing the precisions at the specified document cut-off value, divided by the number of topics (20). Again the precision of the webspace search engines is much better than for the standard search engine. The table also shows that the ranking computed by the FWSE is better than for the WSE.

Table 4. Document level averages

Results of the standard evaluation method			
	SSE	WSE	FWSE
At 5 docs	0.4400	0.8200	0.8500
At 10 docs	0.3650	0.6950	0.7100
At 15 docs	0.3476	0.6233	0.6267
R-precision	0.3686	0.7363	0.7419

The *R-precision* is the precision after R documents have been retrieved. In this case R equals the number of relevant documents. It de-emphasises the effect of the ranking of a run. It shows that the R-precision for both webspace search engines is 0.74.

5.4 Average Precision Histogram

The average precision histogram of Figure 6 shows the average non-interpolated precision for each of the three runs per topic. This measure gives a good indication of

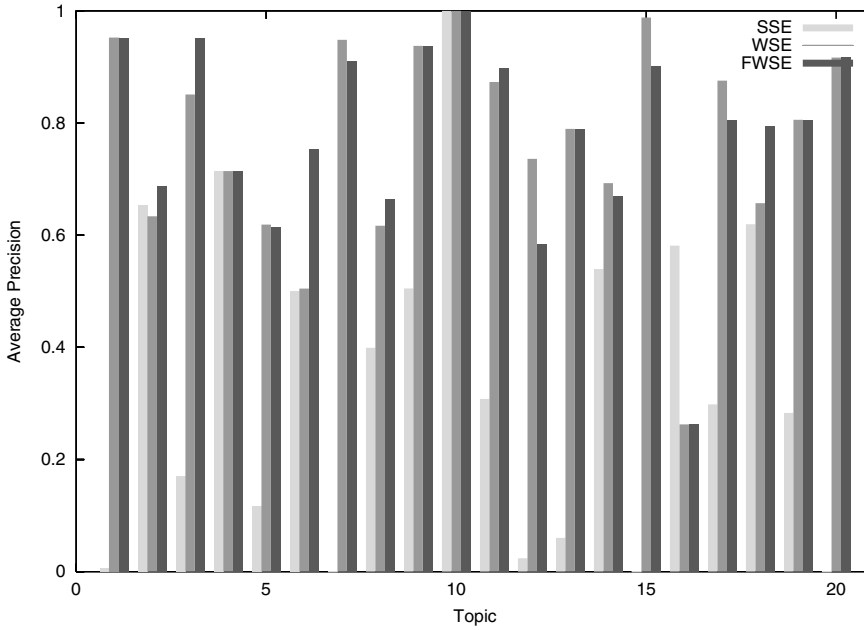


Figure 6. Average precision histogram

the performance of a run on specific topics. The measured precisions for the topics 1, 3, 5, 7, 8, 9, 11, 12, 13, 15, 17, 19, and 20 show a huge increase in retrieval performance, when the topic is evaluated using the Webspace Method. In some cases, where the standard search engine performed really poorly, this was caused by the conceptual constraints, as is the case for Topic 15: '*Find postcards, send from destinations within North East Asia containing information about 'taxi shuttles airport or public transportation'.*'. The constraint '*destinations within North East Asia*' cannot be handled by normal search engines.

Only on Topic 16 the standard search engine performed better than the webspace search engines. While the performance of the search engines was nearly the same for topics 2, 4, 10, and 18.

When comparing both webspace search engines, it turns out that the performance of the FWSE is the same, or slightly better for most queries than the WSE, with the exception of topics 12, 14, 15, and 17. While for topics 2, 3, 6, 8, and 18 the ranking produced by the FWSE is clearly better than the one produced by the WSE.

6 Conclusions

The results of the retrieval performance experiment show a huge increase in performance, when searching for documents, using the Webspace Method. Although the setup and results of the experiment are good and reliable, more experiments should be carried out to validate the conclusions at this point.

For instance, a relatively low document cut-off point (15 documents per topic) is used, compared to the TREC-benchmark. This might lead to imprecise precisions, at the high recall levels (>0.8). On the other hand, the size of the Lonely Planet test collection is also a lot smaller, which eliminates the necessity of evaluating the relevance of documents which are not highly ranked. Therefore, it is expected that the document cut-off point is not chosen too low.

Both the recall and precision values, found for both webspace search engines are approximately a factor of two higher than the values found for the standard search engine. This typically illustrates the impact of the Webspace Method on the retrieval performance. When comparing the results of the standard search engine with the recall-precision curves of the search engines that participated in TREC, a similar trend is found. Unusual are the (extremely) high precision values of the webspace search engines at the lower recall-levels, caused by the conceptual model of the Webspace Method. These values show the contribution of the Webspace Method to the retrieval process.

From the average precision histogram it can be concluded that on average the fragmented webspace search engine produces a better ranking than the standard webspace search engine. Furthermore, both webspace search engines are clearly capable of answering queries that cannot by answered by the standard search engines which solely have to rely on text retrieval techniques especially, if the information requested, is dispersed over more that one document.

References

1. G. O. Arocena and Mendelzon O. WebOQL: Exploiting document structure in web queries. In *proceedings of the International Conference on Data Engineering (ICDE)*, pages 24–33, 1998.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999. ISBN_ISSN: 0-201-39829-X.
3. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: A graphical language for querying and restructuring XML documents. In *proceedings of the International World Wide Web Conference (WWW*, pages 1171–1187, Canada, 1999.
4. D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A query language for XML. Technical report, World Wide Web Consortium (W3C), `http://www.w3.org/TR/xquery`, Februar 2001.
5. D.A.Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 1998. ISSN_ISBN: 0-7923-8271-4.
6. A.P. de Vries and A.N. Wilschut. On the integration of IR and databases. In *proceedings of the IFIP 2.6 Working Conference on Data Semantics 8*, 1999.
7. N. Fuhr and K. Grossjohan. XIRQL: An extension of XQL for information retrieval. In *proceeding of ACM SIGIR Workshop On XML and Information Retrieval*, Athens, Greece, July 2000.
8. Y. Hayashi, J. Tomita, and G. Kikui. Searching text-rich xml documents with relevance ranking. In *proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

9. I.A.G.H. Klerkx and W.G.Tijhuis. Concept-based search and content-based information retrieval. Master's thesis, Saxion Hogeschool Enschede, in cooperation with the department of Computer Science, University of Twente, Enschede, The Netherlands, march 2001. (in Dutch).
10. G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of xml. *IEEE Data Engineering Bullettin, Special Issue on XML*, September 1999.
11. Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the world wide web. *Int. Journal on Digital Libraries*, 1(1):54–67, 1997.
12. Lonely Planet Publications. Lonely planet online, March 2001, `http://www.lonelyplanet.com/`.
13. R. van Zwol and P.M.G. Apers. Searching documents on the intranet. In *proceedings of Workshop on Organizing Webspace (WOWS'99), in conjunction with Digital Libraries 1999*, Berkeley (CA), USA, August 1999.
14. R. van Zwol and P.M.G. Apers. Using webspaces to model document collections on the web. In *proceedings of Workshop on WWW and Conceptual Modelling (WCM 2000), in conjunction with ER 2000*, Salt Lake City (USA), October 2000.
15. R. van Zwol and P.M.G. Apers. The webspace method: On the integration of database technology with information retrieval. In *proceedings of Ninth International Conference on Information and Knowledge Management (CIKM 2000)*, Washington DC., USA, November 2000.

Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model*

Dimitri Theodoratos

Department of Computer Science, New Jersey Institute of Technology
University Heights, Newark, NJ 07102, USA
dth@cs.njit.edu

Abstract Information integration in the World Wide Web has evolved to a new framework where the information is represented and manipulated using a wide range of modeling languages. Current approaches to data integration use wrappers to convert the different modeling languages into a common data model. In this work we use a nested hypergraph based data model (called HDM) as a common data model for integrating different structured or semi-structured data. We present a hypergraph query language (HQL) that allows the integration of the wrapped data sources through the creation of views for mediators, and the querying of the wrapped data sources and the mediator views by the end users. We also show that HQL queries (views) can be constructed from other views and/or source schemas using a set of primitive transformations. Our integration architecture is flexible and allows some (or all) of the views in a mediator to be materialized.

1 Introduction

The advent of the World Wide Web, has evolved information integration from a traditional multidatabase architecture to a new framework where the information is available in diverse formats and structures. The integration process involves data sources that are heterogeneous and range from object-oriented and traditional relational database management systems to semistructured data repositories. In this context, exploiting the semantics of the data sources, greatly improves the integration process.

A typical approach for dealing with the heterogeneity of the data sources chooses one model as a common data model [22] and converts the modeling languages of the data sources into this model. An integration architecture is shown in Fig. 1. A wrapper [21,9] is software that translates between the source data models and local languages and the common data model and language. It transforms subqueries sent to the data sources and answers returned from the data sources. The mediators [28] are components that obtain information from wrapped data sources or other mediators. They provide information to other mediators above them or to the users of the system. The mediators export a mediator schema which is an integrated representation of data sources. The users can address their queries to the mediators or to the wrapped sources using the language of the common data model or using their own local language though a wrapper. A

* Research supported by EPSRC under the project "Automatic Generation of Mediator Tools for Heterogeneous Database Integration – AutoMed".

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 166–182, 2002.

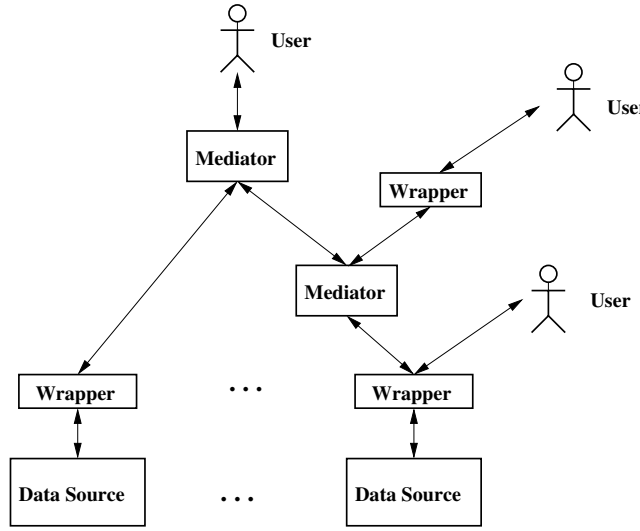


Figure 1. An integration architecture

mediator can be seen as a view of the data found in one or more data sources [25]. Traditionally data are not stored in a mediator. The users query the mediator schema, and the views transform these queries between the mediator schema and the data sources [5]. More recently, the data warehousing approach for data source integration [26,27] chooses to materialize the views in the mediators. In this case user queries are evaluated locally without accessing the data sources. A framework for supporting integrated views using a hybrid of these approaches is also possible [11,10].

The common data model used in this paper for data source integration is based on the Hypergraph Data Model (HDM) [20]. An advantage of the HDM is that it is characterized by a conceptual simplicity. All conceptual objects are modeled using the notions of nodes and hyperedges. Another advantage of the HDM is that the definition of these constructs is separated by the definition of constraints on their values. In this sense, the HDM is a low level model that underpins other data models. Higher level models either structured (e.g. E/R, Relational, UML) or semi-structured (e.g. XML) can be defined in terms of the HDM [14,15]. Thus, the HDM provides unifying semantics for higher level modeling constructs.

Contribution. The main contributions of the paper are the following.

- We present a variation of the HDM for data source integration. The new model distinguishes between two kinds of nodes that model entities and property values. Database schemas are hypergraphs. Instances are associated only with the hyperedges of a schema. Hyperedge instances may be either n-ary, $n \geq 2$, relations as-

sociating entities, or binary relations assigning property values to entities and to associations of entities. Relation attributes are hypergraph nodes. Node renaming functions are used to name the multiple occurrences of entity nodes in hyperedges. Node extensions are captured through the introduction of manifest constraints.
- We provide a hypergraph based query language (HQL). HQL is homogeneous with the HDM: queries in HQL are hypergraphs. In a query hypergraph, hyperedges are associated with relational algebra expressions. A query answer is a relation. Outerjoin based semantics for queries are used to deal with missing data, a common situation when integrating heterogeneous distributed data sources.
- We offer an integration environment through the definition of HQL views (mediators) over multiple database schemas. The concept of schema edge instance is extended to view edges in order to allow the definition of views using other views. Our approach to data source integration is a hybrid one since views can be materialized or virtual.
- We show that an HQL view defined over an integration environment can be constructed from another view using view transformations. These transformations allow us also to deal with view evolution issues. We present a set of two primitive transformations based on which composite transformations can be defined. The primitive transformations are shown to be complete and reversible.

Outline. The next section presents related work. Section 3 introduces the HDM: hypergraphs, schemas, instances and databases. The Hypergraph query language HQL is presented in Section 4. Syntax, semantics, views and data source integration issues are discussed there. Section 5 deals with view transformations, primitive transformations, and their properties. Conclusions and directions for further work are provided in Section 6.

2 Related Work

Although the World Wide Web has brought new dimensions to the problem of data source integration, this is an issue that has been addressed for many years and the bibliography is very extensive. There are several research projects and a considerable number of mediated query systems that assume a mediator environment based on a common data model. They adopt different modeling and querying languages [5]. The Tsimmis project [17] focuses on the integration of structured and unstructured data sources, techniques for the rapid prototyping of wrappers [9] and techniques for implementing mediators. Its common data model is a simple object-based information exchange model, while its query language uses a form of object-logic. The Garlic project implements a dynamic source capability solution [21]. The Information Manifold project [13] uses views to map from the source local schemas to the mediator schema. Its query language is a dialect of description logic. The evaluation of mediator queries is performed by rewriting queries using views [12]. The DISCO project [24] focuses on the problems encountered when the number of heterogeneous distributed data sources scales. Its mediator query language is a subset of ODMG corresponding to the relational algebra. Some data warehousing systems use the concept of mediation for the task of data

integration. This is the case of the H2O project [29,11]. However, even though data warehousing is also an approach for data source integration, the focus of the research in this area is on the materialized view maintenance problem [30], and the view selection problem [23].

A number of visual graphical query languages have been proposed in the past [2,8,19,3,4,18] some of them supporting also nesting [3,4]. They aim at providing a uniform approach to representing, querying and, possibly, updating both schema and data. In contrast to these approaches, our goal here is to use a hypergraph data model as a low-level common data model for heterogeneous data sources and a hypergraph query language for integrating these data sources through the definition of views (mediators). [1] presents a graph model and shows how schemas expressed in different data models can be translated in terms of the graph model. It also provides a set of graphical primitives in terms of which query operations may be visually expressed. However, it does not provide a mechanism for defining and querying views.

3 The Hyper-Graph Data Model for Heterogeneous Data Source Integration

The Hypergraph Data Model [20] is based on the notion of a nested hypergraph. For the needs of this paper we present in this section a variation of the original Hypergraph Data Model. In the presentation below, we borrow terms from the E/R model [6] in order to provide some intuition to the reader.

We assume two infinite sets *Names* and *Values* whose elements are called names and values respectively.

Database Schema. A *node* is an element of *Names*. With every node N a domain $dom(N) \subseteq Values$ is associated. We distinguish two kinds of nodes: *attribute nodes* and *entity nodes*. Entity nodes have a common domain. The domain of an attribute node is disjoint from the domain of the entity nodes. Intuitively, entity nodes represent collections of entities with common properties, and attribute nodes represent collections of property values.

A *hyper-edge* (henceforth simply *edge*) is a pair $\langle E, M\rangle$ where E is an element of *Names* and M is a finite multiset. E is the name of edge $\langle E, M\rangle$. Edges can be of two types: *relationship edges* and *attribute edges*. The multiset M of a relationship edge $\langle E, M\rangle$ contains entity nodes. The multiset M of an attribute edge $\langle E, M\rangle$ is a set of the form $\{C, A\}$, where C is an entity node or a relationship edge and A is an attribute node. An attribute edge between an entity and an attribute node or a relationship edge $\langle E, M\rangle$ is associated with a function that maps every occurrence of an entity node in M to a distinct name. This function is called *node renaming function*. Let f be the node renaming function of an edge $\langle E, M\rangle$. If $f(N)$ is N, name $f(N)$ is not mentioned explicitly. If an entity node N occurs more than once in M, $f(N)$ must be explicitly specified for at least one occurrence of N in M. Intuitively, relationship edges represent collections of associations among entities, while attribute edges represent collections of associations between entities and property values, or between associations of entities and property values. We use the term *construct* to mean either a node or an edge.

A *hypergraph* is a pair $\langle Nodes, Edges\rangle$, where $Nodes$ is a non-empty set of nodes, and $Edges$ is a non-empty set of edges such that:

(a) Every node in $Nodes$ has an incident edge in $Edges$.

(b) Set $Edges$ does not contain more than one attribute edge between an entity node (or a relationship edge) and an attribute node. $Edges$, however, can contain more than one relationship edge involving the same multiset of entity nodes.

(c) If $L \not= N$ is a name assigned by the node renaming function of a relationship edge to an entity node N in $Nodes$, there is a sequence of name pairs $(L_0, L_1), \ldots, (L_{n-1}, L_n)$, $n \geq 1$, such that $L_0 = N$, $L_n = L$ and L_k, L_{k+1}, $k \in [0, n-1]$, are assigned to N by the node renaming function of a relationship edge in $Edges$. One consequence of this is that if an entity node N in $Nodes$ has at least one incident relationship edge in $Edges$, N is assigned to at least one occurrence of N by the node renaming function of a relationship edge.

(d) If $L \not= N$ is a name assigned by the node renaming function of an attribute edge in $Edges$ to an entity node N in $Nodes$, L is also assigned to an occurrence of N by the node renaming function of a relationship edge in $Edges$. That is, new names for an entity node are created by the node renaming functions of relationship edges and are used by the node renaming functions of the attribute edges.

Such a hypergraph is nested because an attribute node can be linked not only to an entity node but also to a relationship edge (multiset of entity nodes). Note that a hypergraph is not necessarily a connected graph. Name E uniquely determines M in a hypergraph $\langle E, M\rangle$. Therefore, in the following, we identify an edge with its name.

Constraints are Boolean expressions involving constructs. Different kinds of constraints can be expressed in the generic hypergraph model as for instance cardinality constraints, ISA constraints, disjointness constraints, and completeness constraints [6]. Here we focus on a hypergraph query language and therefore we do not explicitly deal with constraints. We only mention *manifest constraints* and *uniqueness constraints*. Let E be the attribute edge between an entity node N and an attribute node A. A manifest constraint is denoted $manifest(E)$. Node A is called manifest attribute node of N. Intuitively, a manifest constraint states that every entity of N is associated in E with a property value from the domain of the manifest attribute node A (and thus every entity of N is manifested). The entities of node N are those entities of $dom(N)$ that are associated with an entity or a property value in an edge incident on N. Let now E be the attribute edge between an entity node or a relationship edge C and an attribute node A. A uniqueness constraint is denoted $unique(E)$. Intuitively, a uniqueness constraint states that an entity (if C is an entity node) or association of entities (if C is a relationship edge) is associated in E with at most one property value from the domain of attribute node A. Note that even though constraints are not involved in the definition of a query, their presence affects both the formulation of queries by the user, and the query optimization process.

A *database schema* is a triple $\langle Nodes, Edges, Cons\rangle$, where $\langle Nodes, Edges\rangle$ is a hypergraph H and $Cons$ is a set of constraints over constructs in H. In particular, for every entity node N in $Nodes$, $Cons$ contains a manifest constraint involving an attribute edge incident on N.

Example 1. Figure 2 shows two database schemas S_1 and S_2 that describe two different sections of a company. Schema S_1 deals with employees, their supervisors – who are also employees –, the department where they work and their dependent persons. Schema S_2 deals with departments, their manager – who is an employee –, and the products the departments have created. Schema S_1 describes mainly employees, while schema S_2 describes mainly departments. In Fig. 2, a big black circle denotes an entity node, while a smaller white one denotes an attribute node. The name of a construct is depicted close to the respective construct. An attribute edge between a relationship edge and an attribute node is distinguished by a black bullet in the intersection of the attribute edge and the relationship edge, e.g. attribute edge E_6 in S_1. The name of a manifest attribute node of an entity node is shown in bold, e.g. attribute nodes ENO and $DENAME$ in schema S_1. Unique constraints are denoted by underlying the name of the involved attribute node, e.g. attribute nodes DNO and $DNAME$ in schema S_1. An alternative name for an entity node provided by the renaming function of a relationship edge is shown in typewriter font by the node and the edge, e.g. name $SEMP$ (standing for supervisor employee) assigned to one occurrence of node EMP by the renaming function of the edge $SUPERVISES$ in schema S_1. The other occurrence of node EMP in $SUPERVISES$ is assigned the name EMP but is not displayed in the figure. □

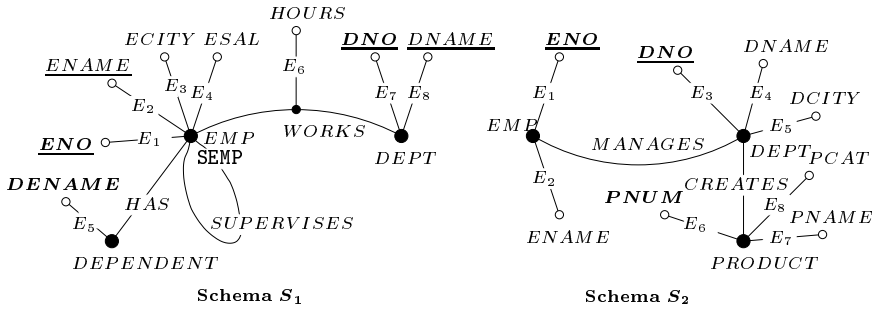


Figure 2. Two database schemas

Database Instance. A database instance is defined using the notion of a relation. We follow the approach to the relational model that uses relation attributes. The attributes of the relations are node names.

We first define the *schema of an edge*. If E is an attribute edge between an entity node N and an attribute node A, and L is the name assigned to N by the node renaming function of E, the schema of E is the set $\{L, A\}$. The schema of a relationship edge is the range of the node renaming function associated with this edge (the set of node names assigned by the renaming function to the occurrences of nodes in the edge). The

schema of an attribute edge between a relationship edge having schema $\{L_1, \ldots, L_k\}$, and an attribute node A is the set $\{L_1, \ldots, L_k, A\}$.

An *instance* $I(E)$ of an edge E is a relation such that:

1. The schema of $I(E)$ is the schema of E.
2. The content of $I(E)$ is a finite set of tuples. If L is an attribute in the schema of $I(E)$ and a name for a node N, and t is a tuple in the content of $I(E)$, the value in t for the attribute L belongs to the domain of N ($t[L] \in dom(N)$).

An *instance I of a hypergraph* $H = \langle Nodes, Edges \rangle$ is a function that maps each edge in $Edges$ to an edge instance such that if $\{L_1, \ldots, L_k, A\},\ k \geq 2$, is the schema of the interpretation $I(E)$ of an attribute edge E between a relationship edge E' and an attribute node A then $\Pi_{L_1,\ldots,L_k}(I(E)) \subseteq I(E')$. ($\Pi_X$ denotes the relational projection operation over an attribute set X.) In other words, a property value is associated with an existing association of entities.

The association of the entities of an entity node with property values from the adjacent attribute nodes in a hypergraph instance need not be complete in the sense that an entity may be associated with a property value of one adjacent attribute node but with no property value of another adjacent attribute node.

Let E be the attribute edge in a hypergraph H between an entity node N and an attribute node, and L be the name assigned to N by the node renaming function of N. An instance I of H *satisfies* the manifest constraint $manifest(E)$, if for every edge E' incident on node N and for every name L' assigned to N by the renaming function of E', $\Pi_{L'}(I(E')) \subseteq \Pi_L(I(E))$. Let now E be the edge in H between an entity node or a relationship edge and an attribute node A. Let also $\{L_1, \ldots, L_k, A\},\ k \geq 1$, be the schema of E. An instance I of H *satisfies* the uniqueness constraint $unique(E)$, if $I(E)$ satisfies the key constraint $L_1, \ldots, L_k$ (in the sense of the relational model) on the schema of $I(E)$.

An *instance of a database schema* $S = \langle N, E, C \rangle$ is an instance of the hypergraph $\langle N, E \rangle$ that satisfies all the constraints in C. In the following, edge names are used to denote both: edges and edge instances. The name of a schema can precede an edge name separated by a period to denote the schema the edge belongs to. □

Example 2. Figure 3 shows the instances of some edges in the instances I_1 and I_2 of the respective schemas S_1 and S_2 of Fig. 2. Here we do not deal with inconsistency and conflict issues in heterogeneous distributed data sources. We assume that there is a mapping that identifies the same entities in the two instances, and distinguishes different entities. Thus, the same value in the instances of the two schemas correspond to the same entity, and conversely, and the same holds for property values. Further, we assume that I_1 has more information about employees than I_2. In this sense any property value assigned to an employee entity in S_2 is also assigned to the same employee entity in I_1. □

Database. A database is a pair $\langle S, I \rangle$, where S is a database schema and I is an instance of S.

$S_1.E_1$

EMP	ENO
e_1	1
e_2	2
e_3	3

$S_1.E_2$

EMP	$ENAME$
e_1	$Smith$
e_2	$Brown$
e_3	$Jones$

$S_1.E_3$

EMP	$ECITY$
e_1	NY
e_3	LA

$S_1.E_4$

EMP	$ESAL$
e_1	$10,000$
e_2	$50,000$

$S_1.SUPERVISES$

EMP	$SEMP$
e_2	e_1
e_3	e_1

$S_1.WORKS$

EMP	$DEPT$
e_1	d_1
e_2	d_2
e_3	d_2

$S_2.E_3$

$DEPT$	$CITY$
d_1	SF
d_2	LA

Figure 3. Edge instances

4 The Hypergraph Query Language HQL

We introduce in this section the hypergraph query language HQL.

Syntax. A *query* over a database schema S is a hypergraph H such that:

1. An attribute node in H has at most one incident edge.
2. The nodes in H are marked as hidden or shown. In particular, entity nodes are marked as shown.
3. Every edge E in H is associated with a relational algebra expression that involves edge names from S. Given an instance for S, this expression evaluates to a relation R. The schema of R is the schema of E. If L is an attribute in the schema of R and a name for a node N, and t is a tuple in the content of R then $t[L] \in dom(N)$.

An edge expression is built using the usual relational algebra operators: selection (σ_c, where c is a Boolean combination of selection predicates), projection (Π_X, where X is a set of attributes), join ($\bowtie_c$, where c is a conjunction of join predicates), union ($\cup$), intersection ($\cap$), difference ($-$), and attribute renaming ($\rho_{A \to B}$, where A and B are node names). In particular, the expression of an attribute edge does not involve join operators, while the expression of a relationship edge does not involve selection operators.

Semantics. Let Q be a query over a database schema S. The *answer* of Q over an instance I of S is a relation resulting by evaluating over I an expression e defined as follows:

1. If Q contains only one edge E_1 associated with an expression e_1, e is $P_X(e_1)$, where P_X denotes the duplicate and null preserving projection operator, and X is the set of names (attributes) in the schema of E_1 that correspond to shown nodes. If E_1 is a relationship edge, all the names in the schema of E_1 correspond to shown nodes since the nodes of E_1 are entity nodes. If E_1 is an attribute edge, one name in

the schema of E_1 corresponds to an attribute node that can be a shown or a hidden node.

2. If Q contains only attribute edges $E_1, \ldots, E_n$ associated with the expressions e_1, $\ldots, e_n$, respectively, that involve only attribute renaming operators (if any), e is the expression $P_X(e_1 \sqsupset\!\bowtie\!\sqsubset \ldots \sqsupset\!\bowtie\!\sqsubset e_n)$, where $\sqsupset\!\bowtie\!\sqsubset$ denotes the natural full outer join operation, and X is the set of names in the schemas of $E_1, \ldots, E_n$ that correspond to shown nodes.
3. If Q does not fall in any of the previous two cases, it contains edges $E_1, \ldots, E_n$, $n \geq 2$, associated with the expressions $e_1, \ldots, e_n$, respectively, such that E_1 is a relationship edge, or e_1 involves operators other than attribute renaming operators. Then e is the expression $P_X(e_1\ \theta_1 \ldots \theta_{n-1}\ e_n)$, where θ_i is: (a) the natural join operator, $\bowtie$, if E_{i+1} is a relationship edge or e_{i+1} involves operators other than attribute renaming operators, and (b) the natural left outer join operator, $\sqsupset\!\bowtie$, otherwise (in this case E_{i+1} is an attribute edge involving only attribute renaming operators, if any). X is again the set of names in the schemas of $E_1, \ldots, E_n$ that correspond to shown nodes.

Notice that even though the instances of the edges of a schema are sets of tuples and do not contain null values, the answer of a query is a multiset of tuples and can contain null values. A null value can appear in a tuple in the answer of a query only for an attribute corresponding to an attribute node. Null values may occur in the answer of a query only as a result of the incompleteness of the association of the entities of an entity node with property values from an adjacent attribute in a database instance. These semantics are appropriate for queries because, in the framework of data source integration, we often deal with missing data.

Views. A *view* is a named query. When the answer of a view is stored, the view is called *materialized*. Otherwise, it is called virtual. In our approach, a mediator can contain a virtual or a materialized view.

Example 3. Consider the database schema S_1 of Fig. 2.

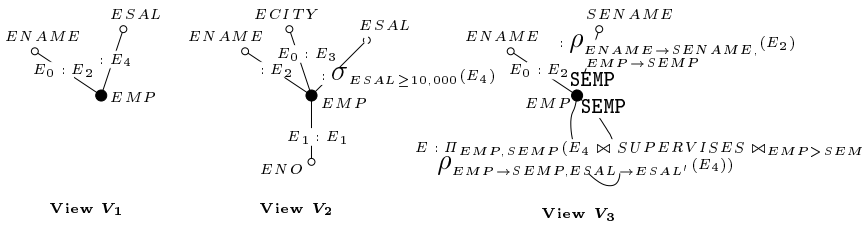


Figure 4. Views over schema S_1

EMP	$ENAME$	$ESAL$
e_1	$Smith$	$10,000$
e_2	$Brown$	$50,000$
e_3	$Jones$	null

(a)

EMP	ENO	$ENAME$	$ECITY$
e_1	1	$Smith$	NY
e_2	2	$Brown$	null

(b)

EMP	$SEMP$	$ENAME$	$SENAME$
e_2	e_1	$Brown$	$Smith$

(c)

EMP	$ENAME$	$DEPT$
e_3	$Brown$	d_2

(d)

EMP	$ECITY$
e_1	NY
e_2	null

(e)

Figure 5. View answers

View V_1 (Figure 4) over S_1 retrieves the names of the employees and their salaries. The expression of an edge is shown in the Figures by the corresponding edge preceded by a colon. The name of an edge is often omitted in the Figures (e.g. the name of the edge between nodes EMP and $ESAL$ in view V_1). When it is not omitted, it precedes the colon (e.g. the name E_0 of the edge between nodes EMP and $ENAME$ in view V_1). The expression that computes the answer of V_1 is $E_2 \supset\!\!\!\bowtie\!\!\!\subset E_4$. The answer of V_1 over the instance I_1 of S_1, partially shown in Fig. 3, is displayed in Fig. 5(a).

View V_2 (Fig. 4) over S_1 retrieves the numbers and the names of the employees and the cities where they live but only for the employees whose salary is greater than or equal to $10,000$. Attribute node $ESAL$ is a hidden node in V_1. Hidden nodes in the Figures are shown with dashed lines. The answer of V_2 is computed by the expression $P_{EMP,ENO,ENAME,ECITY}(\sigma_{ESAL \geq 10,000}(E_4)$ $\supset\!\!\!\bowtie E_1 \supset\!\!\!\bowtie E_2 \supset\!\!\!\bowtie E_3)$. The result of the computation over the instance I_1 of S_1 is shown in Fig. 5(b).

View V_3 (Figure 4) over S_1 retrieves the names of the employees whose salary is greater than the salary of their supervisor and the names of their supervisors. The supervisor's name is provided under the attribute $SENAME$. Notice that the node name $SEMP$ can be assigned to the node EMP by the node renaming function of the attribute edge between EMP and $SENAME$ because $SEMP$ has been assigned to EMP by the node renaming function of the relationship loop edge on EMP. The answer of V_3 is computed by the expression $E_2 \bowtie\!\!\!\subset \Pi_{EMP,SEMP}(E_4 \bowtie SUPERVISES \bowtie_{ESAL>ESAL'} \rho_{ESAL \rightarrow ESAL'}(E_4))$ $\supset\!\!\!\bowtie \rho_{ENAME \rightarrow SENAME}(E_2)$. The result of the computation over the instance I_1 of S_1 is shown in Fig. 5(c). □

Data Source Integration. New edges that are not restrictions of existing schema edges can appear in a view. In other words, an edge among some nodes in a view may not be associated with an expression that involves an edge among the corresponding schema nodes in the schema. Views can also be defined over multiple database schemas. These

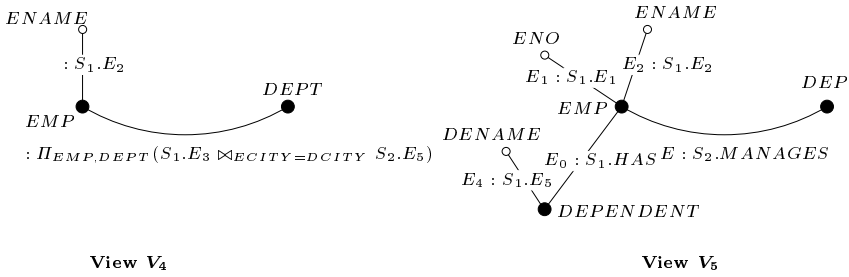


Figure 6. Views over schemas S_1 and S_2

views involve edge names from different schemas, and provide an integration of multiple data sources.

Example 4. View V_4 (Figure 6) retrieves the names of the employees who work for a department located in the city where they live. Schema names precede edge names in the view. The answer is computed by the expression $S_1.E_2 \bowtie\sqsubset \Pi_{EMP,DEPT}(S_1.E_3 \bowtie_{ECITY=DCITY} S_2.E_5)$. The result of the computation over the instance I_1 of schema S_1 and the instance I_2 of schema S_2 (Fig. 3) is shown in Fig. 5(d). View V_5 (Fig. 6) retrieves the numbers and the names of the managers who have at least one dependent and the names of their dependents. The answer is computed by the expression $S_1.E_5 \bowtie\sqsubset S_1.E_1 \bowtie\sqsubset S_1.HAS \bowtie S_2.MANAGES \sqsupset \bowtie S_1.E_2$. □

Queries and views can be defined using other views. A view can be seen as a schema. Given database instances, instances for the view edges are defined as follows: let $L_1, \ldots, L_n$ be the schema of an edge E in a view V, and R be the answer of V. The instance of E is the relation $\Pi_{L_1,\ldots,L_n}(R)$. Π is the relational algebra projection operator and it removes tuples that contain null values. As with schema edges, the name of a view edge is used to name its instance. Note the difference between the instance of a schema edge and the instance of a view edge. If E is a schema edge and also the expression associated with a view edge E', the instance of view edge E' is a subset of the instance of schema edge E. As with schemas, view names precede edge names separated by a period in a view to indicate the view they belong to. Conversely, a schema edge can be seen as a "view". The expression associated with this edge is a constant relation (the instance of the edge). This relation is also the answer of the view.

Example 5. View V_6 (Fig. 7) retrieves the cities where the employees having a salary greater than or equal to $10,000$ live. It is defined using exclusively view V_2. The edge between node EMP and node ENO (the latter being the manifest attribute node of node EMP in the schema S_1) has to be included in the view if null values need to be retrieved. The answer of view V_6 over the instances I_1 and I_2 of schemas S_1 and S_2, respectively, is provided in Fig. 5(e). View V_7 (Fig. 7) retrieves the names of the managers who have no dependents. This view is defined using schemas S_1 and S_2 and

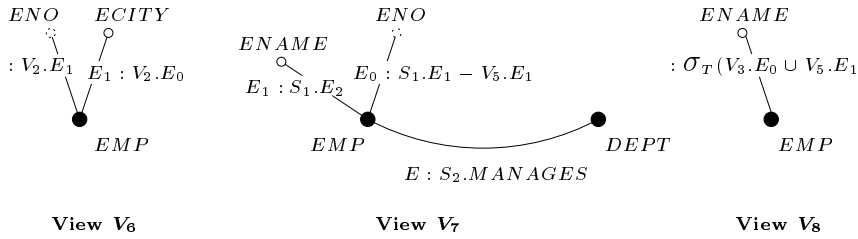


Figure 7. Views defined using other views

view V_5. Its answer is computed by the expression $S_1.E_2 \ltimes\!\sqsubset (S_1.E_1 - V_5.E_1) \bowtie S_2.MANAGES$. View V_8 (Fig. 7) retrieves the names of the employees who are managers and have dependents or whose salary is greater than the salary of their supervisor, but only of employees whose salary is known. View V_8 is defined using exclusively views V_3 and V_5. The expression that computes the answer is the expression associated with the single edge of the view. T in the selection condition stands for a valid formula (that is, a formula that is true for any instance of the attributes that occur in it, e.g. $ENAME = ENAME$). □

HQL and Relational Languages. The expressions associated with the edges of an HQL query are relational algebra expressions, while those that compute the answer of an HQL query are extended relational algebra expressions. There are two advantages related to this feature: (a) HQL queries over HDM schemas can be easily translated to queries in a popular relational query language (e.g. SQL) over relational schemas and conversely, and (b) Well known query optimization techniques for centralized [7] and distributed [16] database systems can be employed for evaluating HQL queries.

5 View Transformations

A view defined over database schemas and/or other views can also be constructed, in a step-by-step way, using view transformations. The transformations formalize changes to a view and thus, they allow us to deal with view evolution issues. A transformation is applied to a *source view* and returns a *target view*. The application of a transformation assumes, besides the source view, a set of database schemas and/or views over which the source view is defined. This set is called *integration environment*. Here we concentrate mainly on primitive (view) transformations.

Primitive Transformations. We define two primitive transformations: $InsertEdge$ and $DeleteEdge$. The primitive transformation $InsertEdge$ inserts an edge in the source view. The source view can be absent. In this case a new view is

created. If the source view is not absent, those of the new edge nodes that already exist in the source view are merged with the corresponding existing nodes of the source view. The rest of the nodes of the new edge are inserted as new nodes in the source view. $InsertEdge$ has one of the following forms depending on whether the edge to be inserted is a relationship edge or an attribute edge:

$InsertEdge(R,\ View\_name,\ Edge\_name,\ Entity\_node\_1,\ Renamings\_1,\ \ldots,\ Entity\_node\_n,\ Renamings\_n,\ Edge\_expression)$ or
$InsertEdge(A,\ View\_name,\ Edge\_name,\ Construct,\ Renaming,\ Attribute\_node,\ Attribute\_node\_status,\ Edge\_expression).$

The first parameter in both forms of the transformation determines the type of edge to be inserted. It is A, if the edge to be inserted is an attribute edge and R, if it is a relationship edge. $Edge\_name$ is the name of the edge to be inserted. $View\_name$ is the name of a view where $Edge\_name$ is to be inserted. If $View\_name$ is not specified, a new view is to be created. $Entity\_node\_1,\ \ldots,\ Entity\_node\_n$ are the nodes of the relationship edge $Edge\_name$, and $Renamings\_1, \ldots, Renamings\_n$ are sets of names assigned to these nodes, respectively, by the node renaming function of $Edge\_name$. There are as many distinct names in $Renamings\_i$ as are the occurrences of $Entity\_node\_i$ in $Edge\_name$. $Construct$ is the entity node or the relationship edge of the attribute edge $Edge\_name$. If $Edge\_name$ is an attribute edge between an entity node and an attribute node, $Renaming$ is a singleton containing the name assigned to node $Construct$ by the node renaming function of $Edge\_name$. If $Edge\_name$ is an attribute edge between a relationship edge and an attribute node, $Renaming$ is an empty set. $Attribute\_node$ is the attribute node of $Edge\_name$. Parameter $Attribute\_node\_status$ takes values H or S depending on whether $Attribute\_node$ is to be a hidden or a shown attribute node, respectively, in the target view. $Edge\_expression$ is the expression associated with $Edge\_name$, and involves edge names from the integration environment.

The primitive transformation $DeleteEdge$ removes an edge from the source view. Those of the nodes of the edge that do not have other incident edges besides this edge are also removed from the source view. The target view can be absent. In this case the source view is deleted. $DeleteEdge$ has the following form:

$DeleteEdge(View\_name,\ Edge\_name).$

$View\_name$ is the name of the view from which the edge is to be deleted and $Edge\_name$ is the name of the edge in view $View\_name$ to be deleted.

Example 6. Suppose that the integration environment comprises the database schemas S_1 and S_2 of Fig. 2 and the view V_7 of Fig. 7. A view identical to view V_5 of Fig. 6 can be constructed by the following sequence of primitive transformation applications to V_7. We assume that when primitive transformations are applied in sequence to a view V_x, the resulting views are named $V_{x1}, V_{x2}, \ldots$.

$DeleteEdge(V_7, E_1)$
$InsertEdge(A,\ V_{71},\ E_2,\ EMP,\ \{EMP\},\ ENAME,\ S,\ S_1.E_2)$
$DeleteEdge(V_{72},\ E_0)$
$InsertEdge(A,\ V_{73},\ E_1,\ EMP,\ \{EMP\},\ ENO,\ S,\ S_1.E_1)$

$InsertEdge(R,\ V_{74},\ E_0,\ EMP,\ \{EMP\},\ DEPENDENT,$
$\{DEPENDENT\},\ S_1.HAS)$
$InsertEdge(A,\ V_{75},\ E_4,\ DEPENDENT,\ \{DEPENDENT\},\ DENAME,$
$S,\ S_1.E_5).$ □

Application Constraints. The primitive transformations are not always applicable. Application constraints result by the fact that $edge\_name$ cannot be inserted in or deleted from the source view $view\_name$ if the graph resulting by this insertion or removal is not a view.

Example 7. Consider the sequence of transformation applications of Example 6. In general, the order of application of the transformations matters. For instance, the third and the fourth transformation applications cannot precede the first two because an edge named E_1 already exists in V_7.

Transformation $DeleteEdge(V_3,\ E)$ cannot be applied to view V_3 of Fig. 4 to delete the loop edge on EMP: if the loop edge on EMP is removed from V_3, the name assigned to the entity node EMP by the node renaming function of the attribute edge between nodes EMP and $SENAME$ in the resulting graph has not been assigned to EMP by the node renaming function of a relationship edge in this graph. □

Composite Transformations. A number of additional transformations are useful for constructing a view from a source view. These transformations modify any of the following elements of the source view: the name of an edge, the nodes of an edge, the names assigned to a node of an edge by the node renaming function of this edge, the status (hidden or shown) of an attribute node in an attribute edge, the expression associated with an edge. Other transformations combine multiple such modifications. For instance, the modification of the expression associated with an edge may require the modification of the nodes of the edge and the names assigned to them by the node renaming function of the edge. All these transformations are called *composite transformations*. A composite transformation application can be expressed as a sequence of primitive transformation applications.

Example 8. In Example 6, the sequence of the first two transformation applications deletes and inserts an attribute edge in order to modify its name (from E_1 to E_2). The sequence of the next two transformation applications inserts an attribute edge in order to modify its name (from E_0 to E_1), its associated expression (from $S_1.E_1 - V_5.E_1$ to $S_1.E_1$) and the status of its attribute node (from H to S). □

Completeness of the Primitive Transformations. A set of transformations is called *complete* if given any two views V_1 and V_2 defined over an integration environment, V_1 can be transformed to V_2 by applying the transformations in this set. Clearly, the set of the two primitive transformations presented in this section is complete.

Database Schema Transformations. Schema transformations are a special case of view transformations since, as mentioned above, a schema can be seen as a set of views. The syntax of the primitive schema transformations is similar to that of the primitive view transformations. By convention, $View\_name$ is the name of the schema. $Edge\_expression$ in the $InsertEdge$ transformation is a constant relation. The schema S of this relation depends on the type of the edge $Edge\_name$ to be inserted in the schema. If $Edge\_name$ is a relationship edge, S is the set $Renamings\_1 \cup \ldots \cup Renamings\_n$. If $Edge\_name$ is an attribute edge between the entity node $Construct$ and the attribute node $Attribute\_node$, S is the set $Renaming \cup \{Attribute\_node\}$. If $Edge\_name$ is an attribute edge between the relationship edge $Construct$ and the attribute node $Attribute\_node$, S is the set $Schema \cup \{Attribute\_node\}$, where $Schema$ is the schema of the relationship edge $Construct$. The primitive schema transformations are subject to a number of application constraints: a primitive schema transformation is not applicable if the graph resulting by the insertion or deletion of an edge violates the definition of a schema.

Example 9. The transformation $InsertEdge(A,\ S_1,\ E_8,\ WORKS,\ \emptyset,\ HOURS,\ s, \{EMP, DEPT, HOURS\})$ cannot be applied to the database schema S_1 of Fig. 2 because: (a) E_8 names an existing edge in S_1, and (b) there is already an edge between the relationship edge $WORKS$ and the attribute node $HOURS$ in S_1. □

Transformation Reversibility. A set of transformations is called *reversible* if whenever two views V_1 and V_2 defined over an integration environment are such that V_2 can be obtained from V_1 by applying the transformations of the set in sequence, V_1 can also be obtained from V_2 by a sequence of applications of transformations in this set. The set of the two primitive transformations presented here is reversible even in a stronger sense: each primitive transformation is the reverse of the other.

Example 10. Consider the sequence of transformation applications of Example 6 to view V_7 resulting in view V_{75} (which is identical to view V_5). A view identical to view V_7 can be obtained by applying to V_5 the inverted sequence of the reverse of each transformation:

$DeleteEdge(V_5,\ E_4)$
$DeleteEdge(V_{51},\ E_0)$
$DeleteEdge(V_{52},\ E_1)$
$InsertEdge(A,\ V_{53},\ E_0,\ EMP,\ \{EMP\},\ ENO,\ S,\ S_1.E_1 - V_5.E_1)$
$DeleteEdge(V_{54},\ E_2)$
$InsertEdge(A,\ V_{55},\ E_1,\ EMP,\ \{EMP\},\ ENAME,\ H,\ S_1.E_2).$ □

6 Conclusion

A typical approach for integrating heterogeneous data sources involves translating their modeling languages into a common data model and defining mediators. Mediators can be seen as views of the data stored in the data sources. We have presented HDM, a hypergraph based data model, and HQL, a hypergraph based query language. HDM is

used as a low-level common data model. HQL is used for integrating HDM schemas through the definition of views, and for querying HDM schemas and views. The semantics of HQL are provided by relational algebra expressions that allow us to deal with missing information and permit the use of well known query optimization techniques for evaluating queries. We have introduced a set of primitive transformations, which are complete and reversible. These transformations can be used for constructing a view from another view in an integration environment, and for dealing with view evolution issues. In our approach some or all of the mediator views can be materialized (hybrid data warehousing approach).

Our current research focuses on two issues: (a) Extending HQL with grouping/ aggregation queries. This type of query is extensively used in data warehousing applications. (b) Determining when a query defined using the source view of a primitive transformation can be rewritten using the target view. Answering this question allows us to avoid the expensive implementation of new mediators when the integration environment evolves.

References

1. T. Catarci, G. Santucci, and J. Cardiff. Graphical interaction with heterogeneous databases. *VLDB Journal*, 6:97–120, 1997.
2. M.P. Consens and A.O. Mendelzon. Graphlog: A Visual Formalism for Real Life Recursion. In *Proc. ACM Symp. Principles of Database Systems*, pages 404–416, 1990.
3. M.P. Consens and A.O. Mendelzon. Hy+: A Hygraph-based Query and Visualization System. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 511–516, 1993.
4. M.P. Cosens, F.C. Eigler, M.Z. Hasan, A.O. Mendelzon, E.G. Noik, A.G. Ryman, and D. Vista. Architecture and Applications of the Hy+ Visualization System. *IBM Systems J.*, 33(3):458–476, 1994.
5. R. Domenig and K. Dittrich. An Overview and Classification of Mediated Query Systems. *SIGMOD Record*, 28(3), 1999.
6. R. Elmasri and B. Navathe. *Fundamentals of Database Systems, Third Edition*. Addison-Wesley, 2000.
7. G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):183–236, 1994.
8. M. Gyssens, J. Paredaens, and D.V.V. Gucht. A Graph-Oriented Object Model for Database End-User Interfaces. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 24–33, 1990.
9. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based Wrappers in the TSIMMIS System. *SIGMOD Record*, 26(2):532–535, 1997.
10. R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *Proc. of the 16th ACM Symp. on Principles of Database Systems*, pages 51–61, 1997.
11. R. Hull and G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 481–492, 1996.
12. A. Levy, A.O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.
13. A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, pages 252–262, 1996.

14. P. McBrien and A. Poulovassilis. A Uniform Approach to Inter-Model Transformations. In *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering, Springer-Verlag, LNCS No 1626*, pages 333–348, 1999.
15. P. McBrien and A. Poulovassilis. A Semantic Approach to Integrating XML and Structured Data Sources. In *Proc. of the 13th Intl. Conf. on Advanced Information Systems Engineering*, 2001.
16. M.T. Oszu and P. Valduriez. *Priciples of Distributed Database Systems*. Prentice-Hall, 2nd edition, 1999.
17. Y. Papakostantinou, H. Garcia-Molina, and J. Widom. Object Exchange across Heterogeneous Information Sources. In *Proc. of the 11th Intl. Conf. on Data Engineering*, 1995.
18. A. Papantonakis and P.J.H. King. Syntax and Semantics of Gql, A Graphical Query Language. *J. Visual Languages and Computing*, 6(1):3–36, 1995.
19. J. Paredaens, P. Peelman, and L. Tanca. G-Log: A Declarative Graphical Query Language. In *Proc. 2nd Intl. Conf. Deductive and Object-Oriented Databases*, pages 108–128, 1991.
20. A. Poulovassilis and P. McBrien. A General Formal Framework for Schema Tranformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
21. M.T. Roth and P. Schwartz. Don't scrap it, wrap it! In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 266–275, 1997.
22. A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
23. D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.
24. A. Tomasic, L. Raschid, and P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.
25. J. Ullman. Information Integration Using Logical Views. In *Proc. of the 6th Intl. Conf. on Database Theory*, 1997.
26. J. Widom, editor. *Data Engineering, Special Issue on Materialized Views and Data Warehousing*, volume 18(2). IEEE, 1995.
27. J. Widom. Research Problems in Data Warehousing. In *Proc. of the 4th Intl. Conf. on Information and Knowledge Management*, pages 25–30, Nov. 1995.
28. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, 1992.
29. G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Supporting Data Integration and Warehousing Using H20. *Data Engineering*, 18(2):29–40, 1995.
30. Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 316–327, 1995.

A Document Database Query Language ⋆

Nieves R. Brisaboa, Miguel R. Penabad, Ángeles S. Places, and Francisco J. Rodríguez

Dep. de Computación Univ. de A Coruña
Campus Elviña s/n, 15071 - A Coruña, España
{brisaboa,penabad}@udc.es, {asplaces,franjrm}@mail2.udc.es

Abstract Abstract. This work presents a natural language based technique to build user interfaces to query document databases through the web. We call such technique Bounded Natural Language (BNL). Interfaces based on BNL are useful to query document databases containing only structured data, containing only text or containing both of them. That is, the underlying formalism of BNL can integrate restrictions over structured and non-structured data (as text).
Interfaces using BNL can be programmed ad hoc for any document database but in this paper we present a system with an ontology based architecture in which the user interface is automatically generated by a software module (User Interface Generator) capable of reading and following the ontology. This ontology is a conceptualization of the database model, which uses a label in natural language for any concept in the ontology. Each label represents the usual name for a concept in the real world.
The ontology includes general concepts useful when the user is interested in documents in any corpus in the database, and specific concepts useful when the user is interested in a specific corpus. That is, databases can store one or more corpus of documents and queries can be issued either over the whole database or over a specific corpus.
The ontology guides the execution of the User Interface Generator and other software modules in such a way that any change in the database does not imply making changes in the program code, because the whole system runs following the ontology. That is, if a modification in the database schema occurs, only the ontology must be changed and the User Interface Generator will produce a new and different user interface adapted to the new database.

1 Introduction

We present in this paper a technique, that we call Bounded Natural Language (BNL), to build user interfaces to query document databases. Interfaces generated using BNL are useful to query document databases containing only structured data, containing only text, or containing both of them. Moreover, these interfaces are suitable to query any document database independently of the DBMS model and/or the text retrieval techniques implemented in the underlying database.

The proposed interface is not codified *ad hoc* but automatically generated by a *User Interface Generator(UIG)* software module. The *User Interface Generator* uses

⋆ This work was partially supported by CICYT (TEL99-0335-C04-02)

B. Eaglestone, S. North, and A. Poulovassilis (Eds.): BNCOD 2002, LNCS 2405, pp. 183–198, 2002.

an ontology that represents the database conceptual model. Therefore, any change in a database does not mean changes in the system.

Since the sixties, when Salton [21][22] defined the Vector Model, a great deal of effort has been made in order to create new text retrieval techniques [23][19][28] and improve the existing ones [6][16] (see [5] for further information). Works about designing user interfaces to query document database and presenting the answers to the users [2][12][25] have also been developed. On the other hand, other techniques to improve the queries, for example to make Relevance Feedback easier [20][24], have been studied. Some query languages have focused on performing queries simultaneously over structured data and texts [3][4][27]. Some examples of a design of interfaces are [7][11][13][14][15][18].

Among the techniques used to perform queries, those that use natural language can be highlighted (see [26] for a good review) because this is the most natural and intuitive way for users to express queries.

The use of natural language is, undoubtedly, very easy for the user. However, it has some inconveniences, due to its intrinsic nature. Natural language does not follow a strict grammar (like programming languages). Natural language contains synonyms, phrases or idioms, and their sentences can be ambiguous. Therefore, it is very difficult to find a technique that accurately transforms a natural language sentence into a formal query in any database query language.

An example of the problems that can be found when natural language is employed is the one illustrated by the next example (in this case on the Web). Using the search engine *Askjeeves* (`http://askjeeves.com` [1]) and asking: *What is the distance between Madrid and Barcelona?* the server provides some answers or redirects us to some questions pre-stored in the database with their answers. Some of them are correct and lead us to the solution for the question. However, others are clearly not. One of those was *Where can I find a metric conversion table?*, probably obtained because the word "distance" was in the query.

It seems clear that the use of pure natural language to build queries presents many problems with respect to its management. In our opinion, building a user interface that fully exploits the natural language advantages but guides the user actions is an important challenge. Our proposal is to use a different technique, which we call Bounded Natural Language.

The Bounded Natural Language (BNL) technique [17] works by offering the user a set of sentences in Natural Language with some gaps that the user must fill in order to express the search conditions. Usually, several sentences about the same topic are presented to the user at the same time. The user must choose which sentences will be used, and fill their gaps. Finally, the set of selected sentences after their gaps have been filled by the user will express, in natural language, the features of the documents the user is looking for.

As we will show in the paper, the BNL technique can be used with any kind of document database. That is, the BNL technique can take advantage of any text retrieval technique and can be used to perform queries over structured data as well as over unstructured data. In fact, restrictions over both kinds of data can be expressed in the same BNL query.

The advantage of BNL is that the user expresses herself in natural language but the system avoids the main problems of dealing with understanding or translating natural language sentences, because using the BNL technique the sentences that express the query were not spontaneously generated by the user.

The rest of this paper is arranged as follows. Section 2 describes the system architecture. In Section 3 the Bounded Natural Language technique is fully described. Some aspects of *Query User Interface* are described in Section 4. The last section contains our conclusions and directions for future work.

2 System Architecture

In this paper we propose a technique to build user interfaces to query document databases. Such a technique could be implemented *ad hoc* adapting it to a specific database. However, that would be a rigid way to build a user interface. Our proposal avoids *ad hoc* programming because we propose the use of different software modules that automatically perform some tasks. They automatically generate the user interface (the *User Interface Generator*), understand user queries translating them to the DML of the DBMS or calling the text retrieval algorithm (the *Query Generator* module) and they manage the database answers for any query (the *Answer Manager*).

To make it possible, we propose the use of an architecture that gives physical and logical independence to the user interface with respect to the DBMS, text retrieval technique and database schema, so that any change in the database schema does not make necessary a recodification and a new compilation of the interface program. The architecture we propose has four layers, as it is shown in Fig. 4, and it is based on the use of an ontology to represent the database conceptual model. In the next subsections, we first describe our ontology and afterwards the whole system architecture.

2.1 Ontology

An ontology is a specification of a conceptualization [8][9][10], that is, a set of concepts and the relationships among them. An ontology describes a domain of interest.

In our system the ontology is the conceptualization of the document database schema. The *Ontology* describes concepts in the document database. Concepts in an ontology are those used in the database domain in the real world. Thus, the ontology provides a clear and understandable data model for users.

The concepts in an ontology are arranged in tree shapes. This structure is supported by means of redundancy but this redundancy helps the user and the *User Interface Generator* algorithm, as we will see in section 4.

There are two kinds of relationships among concepts in the ontology: *Generalization/Specialization* and *Description* relationships.

1. ***Generalization/specialization*** **relationships** (*"is a"* relationship): for instance, a work can be a book, a compilation book (a book edited for someone but with chapters written by different authors), a journal, a thesis, etc.
2. ***Description*** **relationships** (*"has"* relationship): for instance: a person *has* a name, one or more addresses, one or more telephone numbers, etc.

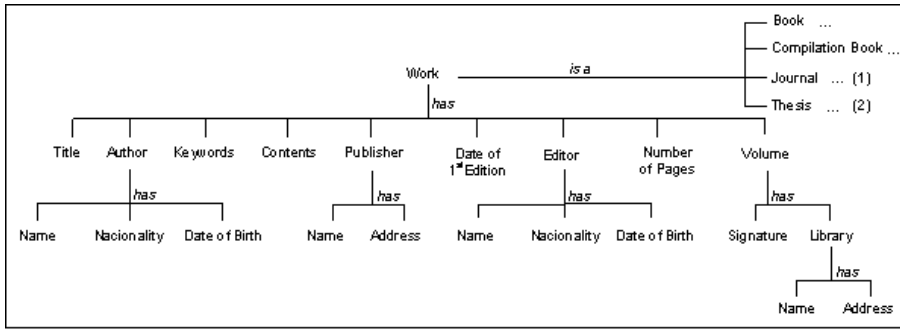


Figure 1. Ontology: *General Part*

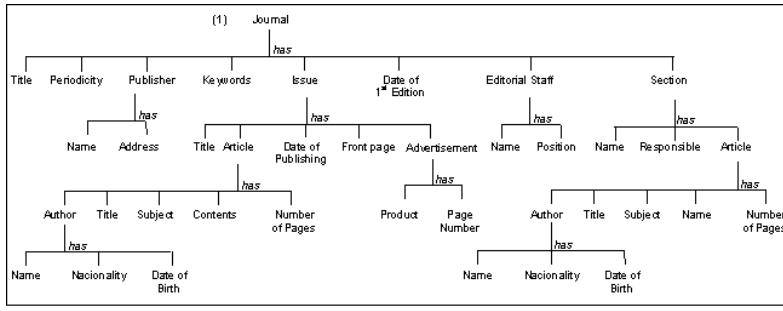


Figure 2. Ontology: Subtree for *Journal*

In the ontology, two parts can be distinguished: *General Part* and *Specific Part*.

1. The ***General Part*** includes the concepts that are common to every corpus in the database. Moreover, they are concepts that any user, even if she is not an expert in any corpus domain, can perfectly understand. The*General Part* of the ontology is shown in Fig. 1.
2. The ***Specific Part*** of the *Ontology* has a *"Corpus" Specific Set* for each corpus existing in the database. Therefore, the ontology includes at least one *"Corpus" Specific Set.* For each corpus included in the database, there is a set of concepts that perfectly describes every relevant concept in that corpus domain. This set that we arrange in a subtree shape, forms a *"Corpus" Specific Set*. The name of the corpus is the concept in the root of the subtree. Probably only experts in the domain can understand concepts in the corresponding "*Corpus" Specific Set* subtree.

In Fig. 1, the *General Part* of an ontology is shown. The root of the tree, *Work*, is specialized in four *"Corpus" Specific Sets*. Fig. 2 and 3 show *Journal Specific Set* and *Thesis Specific Set*. The remaining two subtrees are omitted. Every concept in the *Ontology* has associated two kinds of information:

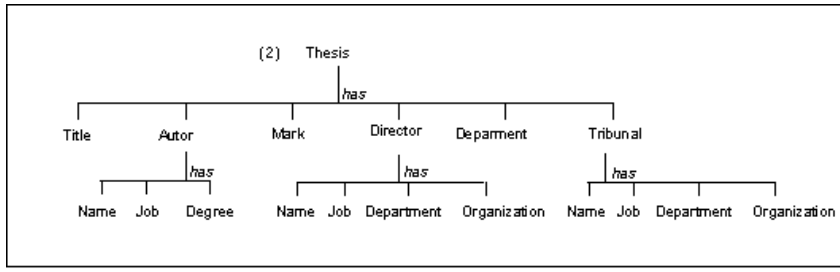


Figure 3. Ontology: Subtree for *Thesis*

1. **The *Ids* of the sentence skeletons,** needed to allow the user to express restrictions about a concept. The syntax of these sentence skeletons is described in section 3.
2. **The expression to access the database.** That is, the expression necessary to access the corresponding data in the associated database. This expression depends on the database DBMS. For example, in a relational database, a concept can have the relation and attribute names where the concept is stored. However it can be more complex. The concept can be represented by an attribute resultant of an algebraic operation between two columns in a table obtained through a *Join* over two different tables. That is, a concept **X** in the ontology can have associated the following expression:
   $\prod_{A+B}(r \bowtie s)$
   where ***A*** is an attribute in relation r and ***B*** is an attribute in relation s. These expressions, associated to the concepts in the Ontology, are used to build the query to retrieve documents from the associated database. Besides, if the database has some kind of text retrieval capabilities, the information associated to a concept like *Contents* or *Subject* will be the directions to call the Text Retrieval algorithm implemented in the database.

The ontology is defined using XML [29]. XML allows the definition of data and metadata formats and it is a universal language to give structure to documents and data. Among the possibilities to represent the ontology [15], we have chosen XML because this language allows us to define formats for data and metadata exchange that are easily human readable and parseable.

2.2 Architecture

The proposed architecture is composed of four isolated layers, and we define three exchange languages to communicate between them. This architecture is shown in Fig. 4.

The architecture layers are:

1. **Layer 1. User Interface:** The *User Interface* is generated every time a user accesses the system.

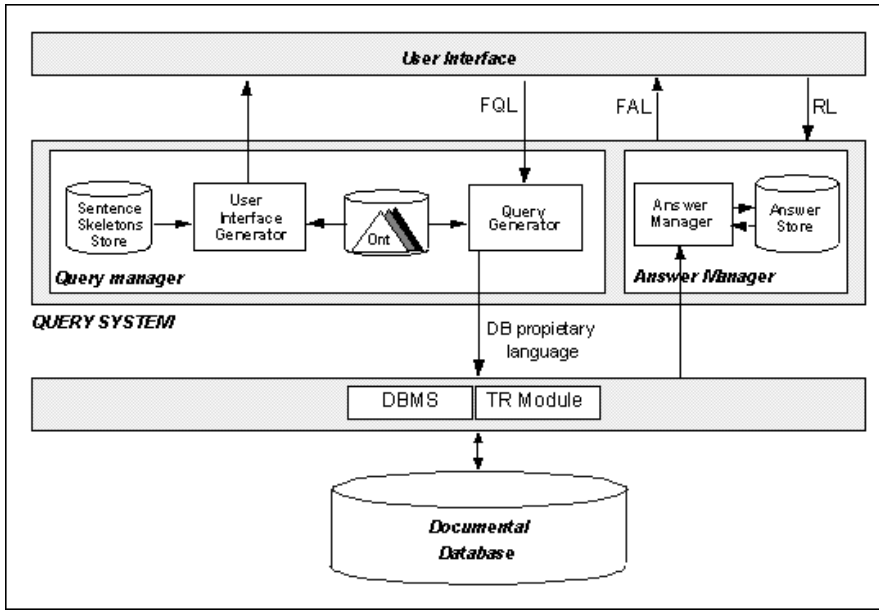


Figure 4. System Architecture

2. **Layer 2. Query System:** The *Query Manager* and the *Answer Manager* form the *Query System*. In this paper we focus on the *Query Manager*. Only a brief description of the *Answer Manager* will be given.
3. **Layer 3. DBMS and TR Modules:** This layer includes the DBMS as well as any other external software modules with text retrieval capabilities.
4. **Layer 4. Database:** The databases are pre-existing and autonomous. Notice that, if a database has text retrieval capabilities, any needed pre-process must have already been performed and those text retrieval techniques must have already been implemented. That is, managing the databases is not a task of our system.

The communication between *User Interface* and *Query System* layers is made by using three exchange languages defined by us to accomplish that communication goal: *Formal Query Language* (**FQL**), the *Request Language* (**RL**) and the *Formal Answer Language* (**FAL**). Notice that there are no software modules in the *User Interface* layer (Layer 1) and the database (Layer 4) and the DBMS and text retrieval techniques (Layer 3) are not the responsibility of the system. For this reason, only the *Query System* (Layer 2) is described below. A brief description of exchange data languages will also be given.

2.3 Query System

This layer includes the *Skeleton Sentence Store* and the *Ontology* that represents the database conceptual model. The following modules form the *Query System*:

- **User Interface Generator:** It automatically generates the *User Interface* using the *Ontology* and the skeleton sentences. The *Query User Interface* program code is independent of the database system, implemented text retrieval techniques and DBMS technology. That is possible because the *User Interface Generator* uses the *Ontology* and the*Sentence Skeletons* to build the *Query User Interface.* Since describing how the user interface is generated is one of the main aims of this paper, in the next section, the *User Interface Generator* module is fully described.
- The **Query Generator** builds the query that will be finally sent to the database, that is, the query to select the *Ids* of the documents that fulfil the user conditions. The*Query Generator* expresses that query in the underlying database language: SQL, OQL, etc. or calls the text retrieval algorithm available in that database. To accomplish this task, the *Query Generator* uses the information associated with the concepts in the *Ontology.* That is, the *Query Generator* reads, from the *Ontology*, for example, the relation, attribute names or a DML statement, which is associated with each concept specified in the query. So the *Query Generator* can translate the query to the specific query language of the underlying database.
- **Answer Manager:** Although describing the Answer Manager is not the aim of this paper, we are going to indicate the main tasks of this module. The first answer the database gives to a query is the set of identifiers of those documents that fulfill the restrictions expressed by the user. These *Ids*, stored in the *Answer Store* by the *Answer Manager,* can be sorted by any specific criteria stated in the query or by the relevance of the document with respect to the query. It is also necessary to store, for each document, an identifier of the session, to avoid conflicts when different users are querying concurrently.

  The *Answer Manager* generates the initial answer page including at least the number of retrieved documents and the title or any other characteristic of these documents that intuitively identifies them. The user can navigate through the set of documents that has been retrieved.

  Following the commands, codified in *Request language* (RL) format, issued by the user in the initial and further pages, the *Answer Manager* will query the database obtaining all the data associated to each document (i.e., images, texts and data).

  To query the database, the *Answer Manager* uses the document and session identifiers provided by the *Answer Store*. By using this technique, we avoid the transmission of all information about all documents at the same time, thus improving the response time. This is a fundamental aspect of the system, because the transmission speed of the Internet is still low, and sending all the information at once would make the system almost unusable.

2.4 Exchange Data Languages

As stated earlier, three formal languages have been defined to establish a communication between architecture layers. They are the *Formal Query Language* (FQL), the *Request Language* (RL) and the *Formal Answer Language* (FAL). FQL formats the queries, FAL formats answer documents and data, and RL formats the requests made by the user. Like the ontology, these languages are developed using XML.

- **Formal Query Language (FQL).** The user queries are expressed in the FQL format. The user builds a query by making restrictions about concepts in the *Ontology*, thus the FQL DTD depends on the *Ontology*. The FQL allows building queries about structured data and the document content (non-structured or semi-structured data) at the same time. Therefore, FQL is integrated in a unique format query restrictions on both structured data and document content.
- **Formal Answer Language (FAL).** The FAL is a format language to represent any document in the system.
- **Request Language (RL).** The RL format language is used to represent the request for documents (or document data) made by the user in the *Answer Interface.* That is, when a document is shown to the user, by means of *Answer Interface*, the RL represents the user requests for the next document, or for a digitized image of this document.

The FQL, FAL and RL exchange languages are defined by their own *Document Type Definition* (DTD).

3 Bounded Natural Language

3.1 User Interface

The main aim of the Bounded Natural Language in our system is to guide the user's queries on the document database. As we have stated, a Bounded Natural Language sentence is a sentence in natural language with some gaps that the user must fill. After filling the gaps, the sentence in natural language represents the restrictions that a document must match to be retrieved.

The inherent ambiguity of the natural language is overcome because the user only needs to select the sentences that represent her query. She does not need to think how to express this query and, obviously, does not have to study or understand the precise syntax and semantics of the system query language.

The user can specify restrictions over different concepts, and usually several sentences about the same topic are presented to the user at the same time. The user must choose which sentences will be used, and fill their gaps. Finally, the set of selected sentences with their filled gaps will express, in natural language, the whole query the user is asking.

3.2 Sentence Skeletons

In this section, the set of sentence skeletons that we propose for the ontology in Fig. 1 will be discussed. As we stated in section 2.1, each concept in the ontology has associated the *id* of the sentence skeleton designed to allow the user to express restrictions over such a concept. Different concepts can have associated the same sentence skeleton, because they can represent attributes with the same data type. In the following figures, the word <**CONCEPT**> represents the gap that the system will fill before it presents the sentence to the user. The system fills these gaps with the corresponding concept in the ontology that the user must restrict in a specific moment of the query process.

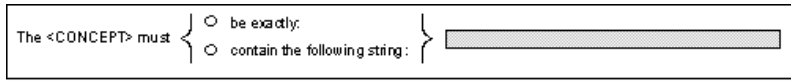


Figure 5. Sentence skeleton for *Short String* data type

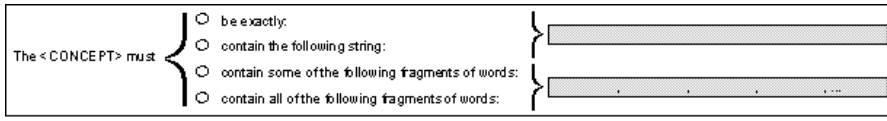


Figure 6. Sentence skeleton for *Long String* data type

That is, in each moment of the process, the system will choose a sentence skeleton (the one associated to the concept being processed) and will fill the <**CONCEPT**> gap in the sentence skeleton with the name of the concept. Therefore, the sentence skeleton becomes a sentence in Bounded Natural Language to be presented to the user. At this moment, the sentence still has gaps, those the user must fill, sometimes choosing one or more possibilities (with radio buttons), sometimes typing in edit fields (represented in figures as rectangles).

For concepts with a *string* type attribute the suitable sentence skeleton would be the one shown in Fig. 5 or the one shown in Fig. 6. It depends on how long the string is. Sentence skeleton in Fig. 5 is useful for concepts such as *Name* or *Nationality* in the ontology in Fig. 1, whereas if the attribute is a long string, like *Title,* the appropriate sentence skeleton is the shown in Fig. 6.

The sentence skeleton in Fig. 7 is useful for *multivalued string* type attributes. This would be the sentence skeleton appropriate for concepts such as *Keywords* in our ontology.

Notice that we ask for a *"fragment of word"* to avoid the use of wild cards (like *****, **%** or **?**) and to make it easier to the user the possibility of looking for any word associated with the same lexical root. The meaning of *"fragment of word"* as well as the meaning of "*string*" will be completely explained to the user.

To express restrictions over concepts associated to *Date* type attributes, the sentence skeleton in Fig. 8 can be used. Likewise, the skeleton in Fig. 9 permits the user to express restrictions over *numeric* type attributes. In the ontology in Fig. 1, concepts such as *Date of 1st Edition* and *Number of Pages* will be associated with sentence skeletons like the ones shown in Figs. 8 and 9, respectively. Concepts associated with *Text*

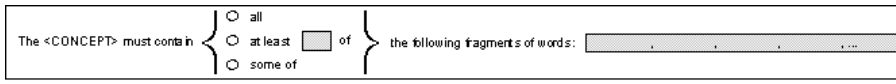


Figure 7. Sentence skeleton for *String* data type with a cardinality greater than 1

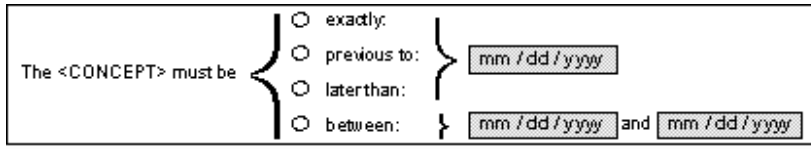


Figure 8. Sentence skeleton for *Date* data type

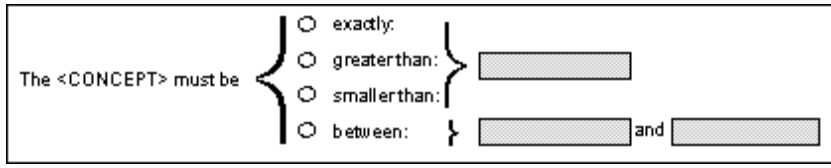


Figure 9. Sentence skeleton for *Number* data type

attributes can use the sentence skeleton in Fig. 10. Restrictions expressed using this sentence can be used even if no text retrieval capabilities are available (using the *LIKE* SQL statement). However any available text retrieval technique can also be used by the Bounded Natural Language technique. We will show through different examples how BNL sentences can be adapted to perform queries using different text retrieval capabilities. For example, if the implemented text retrieval technique can use a thesaurus of synonyms and related words, the system will present the user with the sentence in Fig. 11 following the sentence in Fig. 10. Filling the gaps in the sentence in Fig. 11, the user can enable or disable the use of such thesaurus. Another example that shows how a sentence in Bounded Natural Language can easily guide the user to take advantage of different text retrieval techniques is the sentence skeleton in Fig. 12. This sentence skeleton (presented by the system following the sentence in Fig. 10) allows the use of fuzzy search (words with 1 or 2 different characters to the word written by the user will be accepted).

If the implemented text retrieval technique allows different weights to the words in the query, a suitable BNL sentence skeleton to guide the user in the process of giving different weights to different words could be the one shown in Fig. 13 (such a system will use this skeleton instead the one in Fig. 10).

It easy to see how the BNL technique is flexible enough to allow the design of sentence skeletons for any kind of datatype attributes and for any text retrieval technique.

Figure 10. Sentence skeleton for *Text* data type

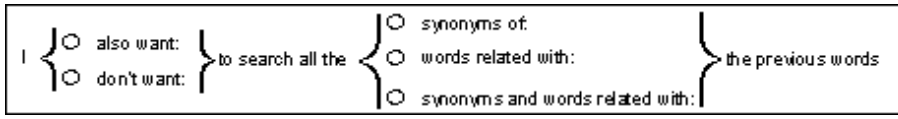


Figure 11. Sentence skeleton for use of the database thesaurus

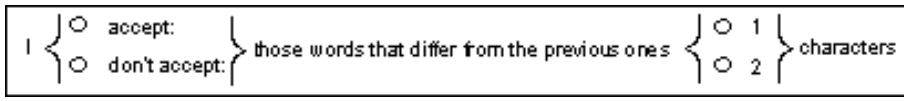


Figure 12. Sentence skeleton for approximate words

The BNL sentences always guide accurately the user in order for her to be able to express the restrictions that the documents she is looking for must satisfy in order to be retrieved.

4 Query User Interface

4.1 Navigational Sentence Skeletons

In this section, we describe how the *User Interface Generator* builds the user interface following the ontology. We have already said that any concept in the ontology has associated a sentence skeleton, but other sentences are also necessary to allow the user the navigation through the ontology so that she can express her whole query. There are two navigational sentence skeletons: *Disjunction* sentence skeleton and *Choice* sentence skeleton.

Disjunction Sentence Skeletons. This sentence skeleton must be presented to the user when the concept being processed has a specialization relationship (like the concept *work* in the ontology in Fig. 1). The *disjunction* sentence skeleton is the one in Fig. 14. In filling this sentence the user will say if she is interested in the general concept or in one of its specializations. In the example, the user will say if she is looking for any kind of work or if she is looking for one specific kind of work (as journals). At the beginning

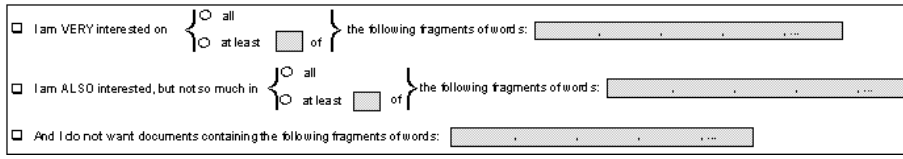


Figure 13. Sentence skeleton for another text retrieval techniques

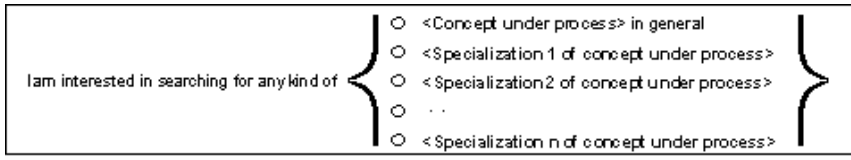


Figure 14. Sentence skeleton for a *Disjunction* sentence

Figure 15. Example of a *Disjunction* sentence

of the process of building a query with the ontology of our example, the *disjunction* sentence skeleton will be created as shown in Fig. 15.

Choice Sentence Skeletons. The second navigational sentence skeleton is used to ask the user which concepts describing the concept being processed she is interested in. That is, the system will ask the user, each time a new concept is being processed, which concepts, related with it by the *"has"*-relationship are relevant for her to express restrictions. This skeleton is represented in Fig 16. Notice that the concepts presented first in the skeleton are those which are leafs (those which have no children through a *"has"*-relationship path). Then the non-leaf concepts are listed. In this sentence, the user can choose as many concepts as she wants because *Choice* sentences have check buttons (not radio buttons). For example, for the concept *Article* in the ontology in Fig. 2, the *choice* sentence is shown in Fig. 17.

4.2 Query Process

In order to build the right chain of sentences the *User Interface Generator* internally uses the algorithm in Fig. 18, that we call *User Interface Algorithm (UIA)*. Initially, the

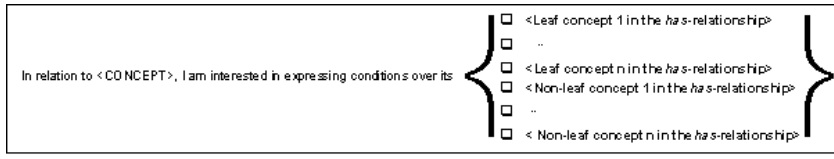


Figure 16. *Choice* Sentence skeleton

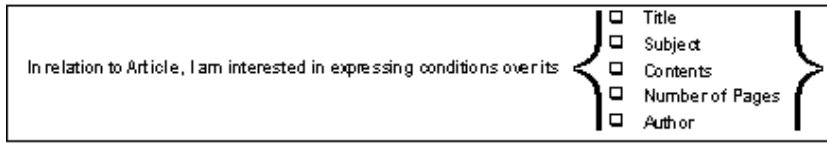


Figure 17. Example of a *Choice* sentence

input parameter for the UIA is the concept at the root of the ontology. The algorithm first checks if the concept being processed has a specialization (*"is a"*-relationship). In this case, the algorithm instantiates the *disjunction* sentence skeleton with the concept being processed and its specialization, takes the option the user chooses and recursively calls the UIA algorithm using as input parameter the concept chosen by the user.

If the concept being processed has no specialization the algorithm uses the *choice* sentence skeleton for that concept. With this sentence the user can choose as many options as she wants and two lists of concepts are built. The first list, called *LeafConceptsChosenSet* will contain the chosen concepts that are leaves in the ontology tree. The second list, called *NonLeafConceptsChosenSet*, will contain the list of children concepts of the concept being processed which also have children and have been chosen by the user. For example, if the concept being processed is *Article* in the ontology in Fig. 2, the instantiated *choice* sentence will be the one in Fig. 17. Assuming the user marks the check boxes corresponding to *Subject, Number of Pages* and*Author* then *LeafConceptsChosenSet* will contain *Subject* and *Number of Pages* and *NonLeafConceptsChosenSet* will contain *Author*.

These two lists are used to run the process. The algorithm will use concepts in both lists to continue its execution.

Notice that in this step the user says which features of the concept being processed she is interested in. That means that she restricts the concept being processed using those features. In the previous example, the user is saying that the *Author*, the *Number of Pages* and the *Subject* of the *Article* are the three features she wants in the *Articles* she is looking for.

For each concept in the *LeafConceptsChosenSet* the associated sentence skeleton is instantiated and presented to the user. Her restrictions are translated into a *Formal Query Language* (FQL) statement that will be part of the whole FQL document that will represent the query. To perform the translation, the algorithm uses the expression in DML associated to the concept in the ontology.

For each concept in the *NonLeafConceptsChosenSet,* the algorithm recursively calls itself using that concept as input parameter.

It is clear that using the algorithm the *User Interface Generator* guides the user through the ontology, expressing the necessary restrictions to build her query. The *User Interface* will present the ontological subtree corresponding to the general part of the ontology or to the specific corpus the user is interested in. In that tree concepts selected by the user will be highlighted. In different windows all the sentences already

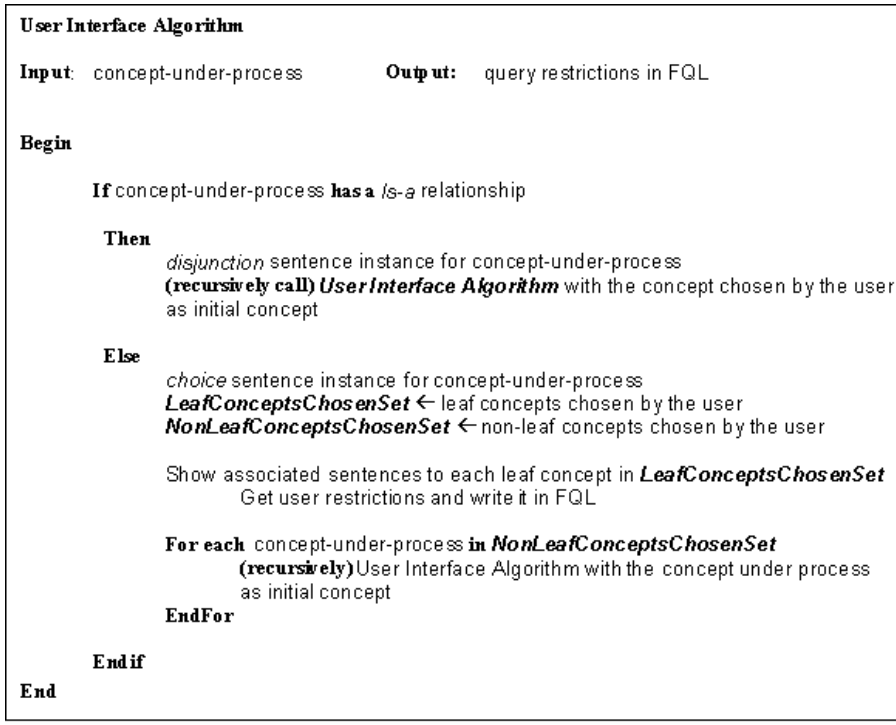


Figure 18. User Interface Algorithm

customized by the user are presented in order for the user to easily see the query she is building.

5 Conclusions and Future Work

We have described, in this paper, the basic algorithm for a *User Interface Generator* module that allows the user to express queries in Bounded Natural Language. The Bounded Natural Language technique allows queries over structured and non-structured data, taking advantage of any text retrieval technique implemented in the database. Although the entire user interface can be programmed *ad hoc*, we think that the proposed architecture is more useful because it provides the *User Interface* with logical and physical independence with respect to the underlying database, so any change in the database schema only implies little modifications in the ontology and maybe in the *Sentence Skeletons Store*. We are developing a first prototype of this system that we think can be adapted to integrate different document databases in a federated architecture.

References

1. Askjeeves `http://www.askjeeves.com`. 2001
2. Alonso, O. and Baeza-Yates, R. "A model and software architecture for search results visualization on the WWW", *Proceedings of the International Symposium on String Processing and Information Retrieval SPIRE 2000*, IEEE Computer Society Press, A Coruña Spain, September 27-29, 2000, pp:8-15.
3. Baeza-Yates, R.; Navarro, G. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67-79, Marzo 1996.
4. Baeza-Yates, R.; Navarro, G.; Vegas, J.; Fuente, P. A model and a visual query language for structured text. En Berthier Ribeiro-Neto (Eds.) *Proc. of the 5th Symposium on String Processing and Information Retrieval*, pp:7-13, Santa Cruz, Bolivia, Sept 1998. IEEE CS Press.
5. Baeza-Yates, R.; Ribeiro-Neto, B. Modern Information Retrieval, Addison-Wesley, 1999.
6. Berry, M. W.; Dumais, S. T.; O'Brien, W. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review* 37:573-595, 1995.
7. Cousins, Steve B., Paepcke, Andreas, Winograd, Terry, Eric A. Bier and Ken Pier; The digital library integrated task environment (DLITE); *Proceedings of the 2nd ACM international conference on Digital libraries*, 1997, Pages 142 - 151
8. Gruber, T. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. IJHCS, 43 (5/6): 907-928. 1994.
9. Gruber, T. `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`
10. Guarino, N. (ed.), Formal Ontology in Information Systems. *Proceedings of FOIS'98*. Amsterdam, IOS Press, pp. 3-15. , Trento, Italy, 6-8 June 1998.
11. Hearst, Marti A. and Chandu Karadi; Cat-a-Cone: An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy; Proceedings of the 20th annual international ACM SIGIR. Conference on Research and development in information retrieval, 1997, Pages 246 - 255.
12. Hearst, M. "User interfaces and visualization" in *Modern Information Retrieval*, Addison-Wesley, London, 1999
13. Koenemann, Juergen and Belkin, Nicholas (1996). A case for interaction: A study of interactive information retrieval behavior and effectiveness. *Proc. CHI'96 Human Factors in Computing Systems*, ACM Press, New York, NY, pp. 205-212.
14. Landauer, T., Egan, D., Remde, J., Lesk, M., Lochbaum, C., and Ketchum, D. Enhancing the usability of text through computer delivery and formative evaluation: The SuperBook project. Hypertext – A Psychological Perspective. Ellis Horwood, 1993.
15. Mena, E., Illarramendi, A., Kashyap, V., Sheth, A. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. *Journal Distributed And Parallel Databases (DAPD)*. 1998.
16. Ogawa, Y.; Morita, T.; Kobayashi, K. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*, 39:163-179, 1991.
17. Penabad, M., Durán, M.J., Lalín, C., López, J.R., Paramá, J, Places, A. S. y Brisaboa, N.R. Using Bounded Natural Language to Query Databases on the Web. Proceeding of the *Information Systems, Analysis and Synthesis* ISAS'99. Orlando (Florida), Julio - Agosto 1999.
18. Rao, Ramana, Card, Stuart K., Jellinek, Herbert D., Mackinlay, Jock D. and Robertson, George G. The information grid: A framework for information retrieval and retrieval-centered applications. Proceedings of the fifth annual *ACM symposium on User interface software and technology*, 1992, Pages 23 - 32
19. Rijsbergen, C.J. van. Information Retrieval. Butterworths, 1979.

20. Robertson, G. C.; Sparck Jones, K. Relevance weighting of search terms. *Journal of the American Society for Information Sciences*, 27(3):129-146, 1976.
21. Salton, G. Automatic information Organization and Retrieval. McGraw-hill, 1968.
22. Salton, G. The SMART Retrieval System - Experiments in Automatic Document Processing. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.
23. Salton, G.; Fox, E. A.; Wu, H. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022-1036, November 1983.
24. Salton, G., and Buckley, C. 1990. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science* 41:288–297.
25. Shneiderman. B. "Designing the User Interface: Strategies for Effective Human-Computer Interaction", 3ed, Massachusetts Addison-Wesley, 1998.
26. Strzalkowski, Tomek, editor. Natural Language Information Retrieval. Kluwer Academic Publishers, Dordrecht, April 1999
27. Vegas, J. "Un sistema de recuperación de información sobre estructura y contenido", Tesis doctoral. Universidad de Valladolid, Valladolid, Spain, 1999.
28. Wong, S. K. M.; Ziarko, W.; Wong, P. C. N. Generalized vector space model in information retrieval. *Proc. 8th ACM SIGIR Conference on Research and Development in information Retrieval*, pp:18-25, New Yok, USA, 1985.
29. World Wide Web Consortium. Standard X

Author Index