Lecture Notes in Computer Science 3664

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Can Türker Maristella Agosti
Hans-Jörg Schek (Eds.)

Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures

6th Thematic Workshop
of the EU Network of Excellence DELOS
Cagliari, Italy, June 24-25, 2004
Revised Selected Papers

Springer

Volume Editors

Can Türker
Functional Genomics Center Zurich, UNI / ETH Zurich
Winterthurerstrasse 190, Irchel, Y32 H06, 8057 Zurich, Switzerland
E-mail: tuerker@fgcz.ethz.ch

Maristella Agosti
University of Padua, Department of Information Engineering
Via Gradenigo 6/a, 35131 Padua, Italy
E-mail: maristella.agosti@unipd.it

Hans-Jörg Schek
University for Health Sciences, Medical Informatics and Technology
Institute for Information Systems
Eduard Wallnöfer-Zentrum 1, 6060 Hall i. Tirol, Austria
E-mail: schek@umit.at

Library of Congress Control Number: 2005931330

CR Subject Classification (1998): H.2, H.4, H.3, H.2.4, H.5

ISSN 0302-9743
ISBN-10 3-540-28711-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28711-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11549819 06/3142 5 4 3 2 1 0

Preface

Newer computing areas like peer-to-peer, grid, and service-oriented computing provide a number of opportunities and challenges for architectures of future digital libraries. Peer-to-peer data management allows for loosely coupled integration of information services and sharing of information such as recommendations and annotations. Grid computing middleware is needed because certain services within digital libraries are complex and computationally intensive, e.g., extraction of features in multimedia documents to support content-based similarity search or for information mining in biomedical data. The service-orientation provides mechanisms to describe the semantics and usage of information services and to combine services into workflow processes for sophisticated search and maintenance of dependencies. Elements of all three directions should be combined in a synthesis for future digital libraries architectures.

This volume contains selected and revised papers from the Sixth Thematic Workshop of the EU Network of Excellence DELOS on Digital Library Architectures, which was held in S. Margherita di Pula (Cagliari), Italy, 24–25 June 2004. This workshop was co-located with the 12th Italian Symposium on Advanced Database Systems (SEBD 2004) and organized jointly by the DELOS Network of Excellence and the Department of Information Engineering of the University of Padua, Italy. DELOS (http://www.delos.info) is an interdisciplinary EU FP6 Network of Excellence with a broad vision: future digital libraries should enable any citizen to access human knowledge any time and anywhere, in a friendly, multi-modal, efficient and effective way. The main objective of DELOS is to define and conduct a joint program of activities in order to integrate and coordinate the ongoing research activities of the research teams in the field of digital libraries for the purpose of developing next generation digital library technologies, and the papers in this proceedings volume address a broad range of issues in this field.

This volume presents and discusses relevant architectural aspects that must be taken into account when designing Digital Library Management Systems (DLMS) able to face the challenges the final users impose on future DLMS. The authors of the papers have made an effort to concentrate on and deal with the architectural areas where present Digital Library systems present specific problems to be solved.

The impact on open problems in this area of digital libraries made by the papers in this volume is relevant, as the reader will discover. Here are some of the topics covered:

- The relation of the EGEE (Enabling Grids for E-sciencE) project to digital libraries is explained and examples of how the project infrastructure can be used in the field are highlighted.
- The fundamental tenets of Service-Oriented Architecture (SOA) and their relevance to Internet-scale computing (or Grid computing) are addressed and clarified together with the presentation of the application of SOA principles to building Internet-scale applications using Web Services technologies and how to avoid software pitfalls.

- A series of experiments confirm that SOA and a service-oriented component architecture is indeed applicable to building flexible, effective and efficient digital library systems, by evaluating issues of simplicity and understandability, reusability, extensibility and performance.
- Digital libraries in healthcare represent an important type of future implementation. This type of system hosts an inherently large and continually growing collection of digital information. Especially in medical digital libraries, this information needs to be analyzed and processed in a timely manner. Sensor data streams, for instance, providing continuous information on patients, have to be processed online in order to detect critical situations. A novel information management infrastructure based on a hyperdatabase system that combines the process-based composition of services and operators needed for sensor data stream processing with advanced grid features is presented to solve this type of challenge.
- The problem of collaborative search across a large number of digital libraries and query routing strategies in a peer-to-peer (P2P) environment is faced and experiments with the MINERVA prototype testbed study the benefits and costs of P2P search for keyword queries.
- Similarity search in metric spaces represents an important paradigm for content-based retrieval of many applications. Scalable and distributed new types of indexes are proposed and experimented by exploiting parallelism in a dynamic network of computers.
- Similarity search can benefit from the support of an infrastructure that combines various information technologies like databases, service-oriented architectures, peer-to-peer and grid computing. Query distribution and load balancing based on domain-specific knowledge can be exploited by such an infrastructure and it is shown that it is possible to reduce query response times.
- Since digital libraries are dispersed over several peers of a steadily increasing network, dedicated peers may provide specialized services. Examples of this sort of system would be a system that performs specialized image similarity searches or a system that manages annotations in an automatic way in order to support users and their annotative practices.

Finally, we would like to thank all those who contributed directly or indirectly to this volume. Our thanks goes to all the authors for submitting their papers to these post-proceedings as well as to all the members of the program committee for reviewing and helping in selecting the papers.

June 2005

Can Türker
Maristella Agosti
Hans-Jörg Schek

Organization

Program Committee

Maristella Agosti	University of Padua, Italy
Elisa Bertino	University of Milan, Italy
Donatelli Castelli	CNR-ISTI, Italy
Wilhelm Hasselbring	OFFIS Oldenburg, Germany
Yannis Ioannidis	University of Athens, Greece
Martin Kersten	CWI Amsterdam, The Netherlands
Erich Neuhold	FhG Darmstadt, Germany
Hans-Jörg Schek	UMIT Innsbruck, Austria (chair)
Heiko Schuldt	UMIT Innsbruck, Austria
Can Türker	FGCZ Zurich, Switzerland (co-chair)
Gerhard Weikum	MPI Saarbrücken, Germany
Pavel Zezula	Masaryk University Brno, Czech Republic

External Referees

S. Balko
G. Brettlecker

Table of Contents

An Overview of the EGEE Project

Bob Jones

EGEE Technical Director,
CERN, 1211 Geneva 23
Bob.Jones@cern.ch

Abstract. The aim of the EU funded EGEE (Enabling Grids for E-sciencE) project is to build on recent advances in Grid technology and develop a service Grid for scientific applications. This paper provides an overview the EGEE project goals and structure and explains the work being performed in the areas of grid operations, middleware re-engineering and application deployment and support. The relation of EGEE to digital libraries is explained and examples of how the project infrastructure can be used in the field are highlighted.

1 Background

Computer and networking technology make the seamless sharing of computing resources on an international or even global scale feasible. The EGEE (Enabling Grids for E-SciencE) project is funded by the European Union to build a scientific computing Grid, using clusters or farms of PCs and associated data storage facilities in major research establishments around the world which are connected via high-speed networks. EGEE focuses on applications requiring high-throughput computing and a diverse scientific community already takes advantage of the opportunities that the production-quality Grid developed by EGEE provides.

This article describes the current status of the project, illustrating the scale and complexity of the challenge involved in establishing a scientific infrastructure of this kind, but also gives examples of some applications already ported on the Grid.

2 The EGEE Project

The EGEE project aims to provide a seamless and high quality service to multiple scientific communities, through the development of a production-quality Grid service. More than 70 institutions in 27 countries, organised in twelve partner regions or "federations", work together to build the Grid infrastructure that not only provides simple, reliable round-the-clock access to the underlying computing resources but also performance monitoring tools, user training programmes and other support.

The three main goals of EGEE are:

- the efficient delivery of a production level Grid service, which needs to be manageable, robust, resilient to failure, and include a consistent security model;
- professional Grid middleware re-engineering in support of the production service, including the support and continuous upgrade of a suite of software tools capable

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation …, LNCS 3664, pp. 1 – 8, 2005.

of providing production level Grid services to a base of users which is anticipated to rapidly grow and diversify;
- strong outreach and training efforts on Grid technology, from induction to advanced topics.

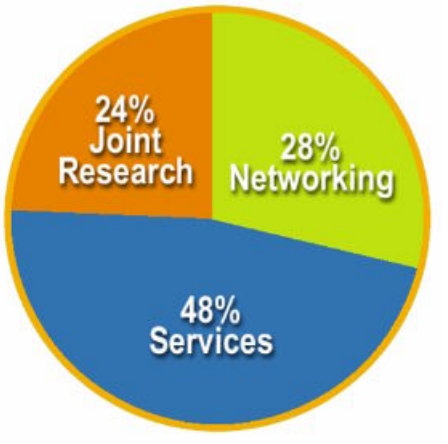


Fig. 1. EGEE structure and distribution resources

EGEE's structure, as seen in Figure 1, reflects these goals in its three main areas: services, joint research and networking. The service activities deploy, support, and manage an international production quality Grid infrastructure including resources from centres around the globe, which is made available to a broad range of user communities. Providing a continuous, stable Grid resource is the main objective of EGEE, reflected in the fact that this activity receives nearly 50% of the EGEE budget.

Most Grid projects to date have concentrated on demonstrating the feasibility and possibilities in establishing Grid testbeds, as did the European DataGrid (EDG) project [1], precursor to EGEE. The joint research activities in EGEE focus on re-engineering existing middleware to develop and improve Grid middleware (see next section).

With a view to expanding and diversifying EGEE's user community, the networking facilities disseminate appropriate information to new scientific fields and take into account their emerging Grid infrastructure needs. EGEE also runs an extensive training programme to ensure that the different stakeholders can make the best use of the resources provided.

3 EGEE Middleware

The Joint Research Activities in EGEE focus primarily on delivering reliable production quality middleware. This is made possible by re-engineering existing middleware and incorporating the experience of past and present R&D projects. Since EGEE is focused on providing a production quality Grid, this re-engineering process includes feedback from Grid user communities and lessons learnt in Grid operations.

However, since standards are still emerging in the field of grids, the project is also actively engaged in preparing for the next generation of Grid standards (e.g. GGF [2] and OASIS [3]), while taking a prudent approach in adopting current emerging standards. The EGEE middleware is branded under a new name: gLite (pronounced "gee-lite" – see www.glite.org).

Fig. 2. The EGEE middleware gLite

gLite builds on the best-practice experience and middleware produced by the most well-known Grid middleware projects of this generation: Condor, Globus, VDT, AliEn, EDG, DataTAG, etc. In order to guide the re-engineering, EGEE has delivered Grid middleware Architecture and Design documents, targeting Web Services as the baseline for the implementation of its Service Oriented Architecture.

Security is another key domain where re-engineering of current Grid services is required. Retrofitting security to an already existing implementation that was not designed with this requirement in mind can be difficult. This is an area were the right balance has to be struck between re-engineering of current services through wrapping techniques, more invasive work and re-factoring. The gLite roadmap includes short term security solutions based on Transport-Level Security, while in the longer term it is likely to be based on Message-Level Security, using standards such as WS-Security.

As for all other IT infrastructures (e.g. telecommunications, telephony, networking) the Grid requires a rich set of stable and well adopted standards to guide service developers and providers, and ensure interoperability between Grid Services. Since we do not believe the right level of maturity has been reached yet, gLite tries, wherever possible, to be inline with the "spirit" of Grid standards and recommendations, while sometimes implementing a temporary custom solution. These solutions are then shared with the international Grid communities as examples of what can be achieved with the current technologies as well as feedback from the practical experience of their deployment and use.

4 Services

The need to support several applications and user communities meant that a new, scalable operation and user support structure had to be developed. For a large-scale,

multi-disciplinary Grid such as EGEE, a federated approach was deemed the most appropriate, where regional support organisations offer faster response times due to their local knowledge about the specifics of the resources. The service activities deploy, operate, support and manage an international production quality Grid infrastructure, including resources from many resource centres primarily across Europe, but also extending around the globe.

Prior to the deployment of a new release of the Grid middleware to the production service, the service activities run tests on a pre-production service. Only once these rigorous tests are passed successfully can new releases be deployed. This intermediate step is necessary since EGEE is working on re-engineering existing middleware and developing novel Grid applications in parallel, thus increasing the risk of disturbing operations on the production quality Grid service. Furthermore, the pre-production service serves the double purpose of testing new Grid middleware features in a multi-site context and providing a realistic environment for testing new applications and policies, minimizing deployment corrections due to the diversity of EGEE's user community.

The structure of the Grid services is shown in Figure 3. The operations activity is coordinated by the Operations Management Centre (OMC) team and supported by the Core Infrastructure Centres (CIC) that provide operations support using a weekly rotating responsibility scheme, operational and performance monitoring, troubleshooting as well as general grid services. The responsibility for middleware certification, deployment, day to day operations and user support within the regional federations rests with the Regional Operations Centres (ROCs), which are in close contact with the Resource Centres (RC) that actually execute the applications.

5 Applications

To guide the implementation and to certify the performance and functionality of the evolving European Grid infrastructure, two pilot application areas have been selected:

- High-Energy Physics (HEP) with several partners, including a close collaboration with the Large Hadron Collider Computing Grid (LCG) [4], which will provide the Grid infrastructure to store and analyse petabytes of real and simulated data from the LHC accelerator experiments at CERN;
- Biomedicine, where several communities are facing equally daunting challenges to cope with the flood of bioinformatics and healthcare data.

The pilot applications are completed by a more generic component trying to identify new applications from a broad range of scientific disciplines and to provide them with the support and tools needed to accelerate their transition to the Grid. An important tool in that respect is a dedicated testbed called GILDA [5], the Grid INFN Laboratory for Dissemination Activities, which was developed as part of the Italian INFN Grid project [6] and the EGEE project. GILDA acts as a Grid applications incubator, and is also used to host hands-on tutorials in many of the EGEE training events.

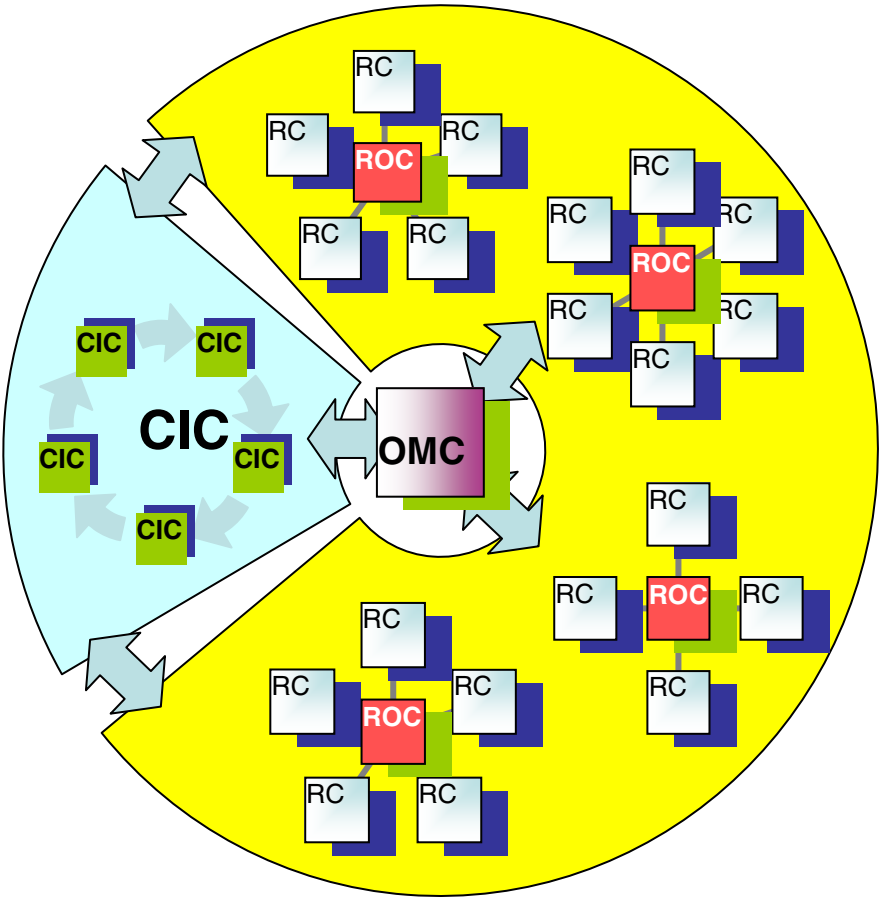


Fig. 3. Organisation of the EGEE Grid services

The applications currently running on the EGEE Grid infrastructure are predominantly from the HEP and biomedical domains, while several other domains such as Earth Observation, Geophysics, and Computational Chemistry have also been deployed and clearly show the breadth of fields using the Grid. The EGEE project has also attracted an application deployed by an industrial partner, Compagnie Générale de Geophysique (CGG) in France.

To expand its user community, the project has set up the EGEE Generic Applications Advisory Panel (EGAAP), which acts as the formal entry point for new applications wishing to take advantage of the EGEE infrastructure. Three applications have been approved by the EGAAP in June 2004, which are already up and running on the Grid infrastructure, and a further four were reviewed and recommended by EGAAP in November 2004.

Keeping up the rapid growth of the computing resources available to the Grid infrastructure, as well as the number of scientific communities that use it, requires a constant "virtuous cycle":

- from making contact with new scientific communities through the many outreach events organised;
- follow-up meetings by application specialists that may lead to the identification of new requirements for the infrastructure and its middleware;
- providing appropriate training to the new community in question, such that its users become established and autonomous;
- through peer communication and dissemination events these new users then spread the work and attract new communities, resources and ideas.

This cycle binds the different activities together and ensures the cohesive expansion of the Grid and its user communities.

6 Collaborations

EGEE integrates current national, regional and thematic Grid efforts, computer centres supporting one specific application area, or general computer centres supporting all fields of science in a region.

This infrastructure builds on the EU research network GÉANT [7] and exploits Grid expertise that has its roots in projects such as EDG. Interoperability with other Grids around the globe, including the US National Science Foundation Cyberinfrastructure [8], and contributing to efforts to establish a worldwide Grid infrastructure is therefore of prime importance for EGEE. Possible collaborations with Russia, Baltic, Asian, North and Latin American states, as well as further links around the Mediterranean (EUMedConnect [9]) are being explored.

The EGEE project collaborates with other European research infrastructure projects to actively participate in pan-project "concertation" meetings, in order to support the effort of consolidating a common approach by the largest possible number of European Grid Projects to vital issues like authentication, authorization and accounting, business and work models and applications. The First Concertation Meeting on e-Infrastructures was organized in conjunction with the second EGEE project conference, November 2004 in Den Haag (The Netherlands). The projects participating represent "service providers", "technology providers" and the potential consumers of infrastructure services:

- DEISA http://www.deisa.org
- SEEGRID http://www.see-grid.org
- DILIGENT http://www.diligentproject.org
- GÉANT2 http://www.geant2.net
- COREGRID http://www.coregrid.net
- GRIDLAB http://www.gridlab.org
- SIMDAT http://www.scai.fraunhofer.de/simdat.html
- GRIDCC http://www.gridcc.org

- LOBSTER http://www.ist-lobster.org/
- GRIDSTART http://www.gridstart.org
- NEXTGRID http://www.nextgrid.org
- AKOGRIMO http://www.mobilegrids.org

As well as providing an opportunity for technical experts from different projects and policy makers to exchange information about the research questions in key areas, the meetings aid in creating a tight networked community of researchers.

7 EGEE and Digital Libraries

DataGrid [1], a predecessor to the EGEE project, has been working with the GRACE (Grid Search and Categorization Engine) project [10]. The goal of the project is to develop a search and categorization engine capable of making vast amounts of geographically distributed information highly accessible. The types of documents that GRACE will process include unstructured text and heterogeneously structured text in the form of text files, web pages, and text stored in databases. While these types of documents are currently searchable using existing meta-search engines, GRACE hopes to break new ground by providing the necessary storage and processing power to search documents on a scale not feasible with current search tools.

EGEE can contribute to the GRACE project and related digital library projects by offering the services listed above and implemented within the new middleware (gLite):

- Workflow management
  Including published interfaces to grid services and additional support for Directed Acyclic Graphs (DAGs)
- Publish/Subscribe techniques
  Including an information service to discover sites and data sources
- Data and document services
  There are a significant set of services foreseen in gLite including:
  Storage Element can be interfaced to data sources (e.g. MSS)
  File catalogs cross multiple sites and replica location services
  OGSA-DAI support for interfacing to data in multiple formats
  Direct support for meta-data catalogs
  Digital rights management with security & certification
  can restrict access to data in SEs using ACLs and PKI certificates (VOs).

Groups building digital libraries (DL) may make use of these services directly or build their own higher-level services that combine the functionality of several gLite services to offer a more appropriate interface. As an example, the DILIGENT project [11] has recently defined an architecture that will be composed of a set of interacting services providing:

- a set of typical DL functions, like search, annotation, personalisation, document visualisation;
- access to information sources and applications provided by third-parties;

- features necessary for handling the shared content and application resources; and
- support for the creation and operation of on-demand, transient digital libraries.

These services will exploit the high computational and storage capabilities of the Grid infrastructure released by the EGEE project, in order to support complex and time consuming functionalities, while focusing on optimizing resource usage and satisfying Quality-of-Service contracts.

8 Conclusions

It is a great challenge to work with demanding communities such as HEP and to develop a production level Grid infrastructure in close collaboration with them. At the same time, this infrastructure has to be versatile enough to be used by the largest possible domain of applications.

Open-source software is the best approach for publicly funded projects and necessary for fast and wide adoption of the developed infrastructure. Nevertheless, intellectual property rights need to be developed together with the industrial partners to make a commercial exploitation possible as well.

One of the reasons Europe is leading in the field of Grid technology is due to the initial success of the EGEE project and its precursor EDG. However, by its very global nature, the Grid is not confined to a geographic region. Therefore, it is of prime importance that EGEE establishes firm collaborations across national and international programmes and funding agencies, and to secure long support of the Grid infrastructure EGEE is producing.

EGEE is a project funded by the European Union under contract INFSO-RI-508833.

References

1. The European DataGrid project: http://www.eu-datagrid.org
2. GGF: http://www.gridforum.org/documents/GFD/GFD-C.3.pdf
3. OASIS: “Web Services Security (WS-Security)” http://www.oasis-open.org/committees/wss
4. LCG: I. Bird et al. “Operating The LCG and EGEE Production Grids for HEP”. In Proceedings of the CHEP’04 Conference, Interlaken, Switzerland, September 27th – October, 41st, 2004. Published on InDiCo.
5. GILDA: https://gilda.ct.infn.it/
6. INFN Grid (Grid-it): http://grid-it.cnaf.infn.it/
7. GEANT project : http://www.geant.net/
8. NSF Cyberinfrastructure: http://www.cise.nsf.gov/sci/reports/toc.cfm
9. EuMedConnect: http://www.eumedis.net/en/eumedconnect/
10. GRACE project: http://www.grace-ist.org/
11. DILIGENT: http://www.diligentproject.org

Grid Computing Using Web Services Technologies

Savas Parastatidis, Paul Watson, and Jim Webber

School of Computing Science,
University of Newcastle upon Tyne, UK
{Savas.Parastatidis, Paul.Watson, Jim.Webber}@newcastle.ac.uk

Abstract. Service-Oriented Architecture (SOA) is the contemporary paradigm of choice for developing scalable, loosely-coupled applications that span organisations. However the architectural paradigm that is SOA is often confused with the implementation technology that is Web Services. In this paper we aim to clarify the fundamental tenets of SOA and their relevance to Internet-scale computing (or Grid computing). We then show how to apply the principles of SOA to building Internet-scale applications using Web Services technologies and how to avoid software pitfalls by adhering to a number of deliberately simple architectural constraints.

1 Introduction

With the advent and subsequent rise to prominence of Web Services, there has been renewed enthusiasm for service-orientation and Service Oriented Architectures in the development community. While service-orientation is independent of, and pre-dates Web Services technology, the rise and rise of Web Services has meant that the application of SOA has become de rigueur for architects and developers.

Concurrently, 'Grid computing' [11] has emerged as a popular paradigm for enabling the formation of virtual organisations and for integrating distributed resources. A significant investment in terms of capital and human resources has been made in architecting the vision of Grid computing around the concepts of service-orientation [10] using Web Services technologies as an implementation technology.

However there is common misconception concerning Web Services technologies in which they are seen as a form of software magic which automatically yields a loosely coupled solution which is scalable, robust, and dependable. There is sometimes the assumption that the use of Web Services technologies is sufficient to implement high quality Grid applications. It is certainly possible, and generally desirable, to build Grid applications using Web Services protocols and toolkits. However it is equally possible to build such applications in ways that violate every architectural principle and tenet of SOA and lack the characteristics of SOA-based systems.

The central tenet of this paper is that Grid applications built using Web Services technologies maximise their potential only when implemented in a manner that follows the principles of SOA, as opposed to alternative approaches such as platform-independent RPC or distributed Object-Orientation [29].

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation …, LNCS 3664, pp. 9 – 24, 2005.

The views we present here are based on a distillation of implementation effort and experience [22, 24, 25, 35], and form the basis of a simple and scalable abstract view of service-orientation upon which real concrete Grid applications can be based. The rest of this paper is structured as follows: Section 2 briefly introduces the term "Grid computing" and puts it in the context of this paper. Section 3 discusses Service Oriented Architectures independently of any implementation technology, while Section 4 describes how the suite of Web Services technologies could be used to implement service-oriented applications. Section 5 presents a set of principles for building Web Services-based applications while Section 6 discusses the relationship of the suite of Web Services protocols with the principles for Service-Oriented Architectures. Finally, Section 7 draws conclusions.

2 Grid Computing

The vision of Grid computing has evolved from interconnected supercomputers in the 1990s, to a paradigm for Internet-scale, inter-organisation computing. While there is no widely-accepted definition of the term 'Grid computing,' some common uses are:

- 'Utility computing' which is about providing computing resources (e.g. CPU, data storage, access to specialised devices, etc.) in a seamless fashion to end users similarly to the way electricity is delivered to our homes (e.g. [14]).
- 'On-demand computing' which is a term usually used by vendors to promote the concept of outsourced computing and enabling services (e.g. [15]).
- 'Seamless computing", or the interconnection of computing facilities and transparent access to computational, data, and other resources and services (e.g. [20]).
- 'Global data integration' where information is allowed to flow between organisations after the necessary security, trust, policy, privacy, etc. restrictions have been put in place (e.g. [23]).
- 'SETI@home' [3] type applications where communities of altruistic individuals are formed to solve large computational problems (e.g. [7]).
- 'Virtual organisations', or the infrastructure necessary for the dynamic formation, management, and exploitation of alliances between organisations in order to achieve a common goal (e.g. [12]).
- 'Universal computer' where the Internet becomes the operating platform for all users' applications (e.g. [8]).

Irrespective of which of the definitions is adopted, it is clear that those working on building the Grid computing vision have a large set of interesting problems to address, like the pooling of computational capacity, data integration, security, digital contracts, service-level agreements, negotiation, policies, quality-of-service, dependability, electronic payment, etc. Such problems, often encountered in distributed systems research, now have to be addressed and applied at magnitudes up to and including Internet scale. It is due to its large scale and the common belief that service-orientation is the most appropriate paradigm for addressing these issues that

we define 'Grid computing' as *Internet-scale, service-oriented computing* and choose to make use of Web Services technologies to provide the underlying infrastructure.

We argue that Grid system architects face a similar set of problems whether they apply the vision of Internet-scale, service-oriented computing within or across organisation boundaries. We argue that the same set of solutions can be applied in both cases.

Inside Organisations

Within organisations the notion of sharing computational resources, data, network and so forth is already an established practice. However to-date that practice has occurred on a per-enterprise basis where application and data integration is managed at the enterprise architecture level. While enterprise architecture is invaluable in managing today's IT infrastructure because of the proprietary nature of a typical rollout it is difficult to transfer anything other than best practices between projects.

The promise of Grid computing at this level is primarily the opportunity for virtualising access to computational and data resources in a standardised fashion. That is, to make access to typical enterprise resources seamless and repeatable between the different entities of an organisation.

Across Organisations

When working across organisations there are new challenges for Grid computing, different from the kinds of problems faced when working within a single administrative domain. At this level the Grid addresses Internet-scale computing issues including federation of identities, contracts, service-level agreements, quality of service, etc.

The promise of Grid computing at this level is that it will provide a suitably constrained architecture and framework for Internet computing. That is, it will allow applications to be built and integrated with other arbitrary applications exposed across the Internet whilst maintaining high levels of quality of service and be resilient to increases in workload and robust in the presence of failures. As a consequence, new types of science and commercial applications and services will emerge.

3 Service-Oriented Architecture

Service Oriented Architectures [2, 13, 18, 28, 29] exist independently of any specific implementation technology like Web Services, but it was the advent of Web Services, and its accompanying hype, which reinvigorated interest in service-orientation and SOA.

However as researchers and developers have shifted their work to be in vogue with the latest buzzwords, the term SOA has become overloaded. Therefore before discussing how to build Web Services applications, the fundamental constituents of SOA must be pared from the hyperbole surrounding it. In this section we present an

abstract view of Service-Oriented Architecture, which we will later concretise in terms of Web Services.

During the course of our work on Web Services and Grid computing [24-26], we have identified what we believe are the two fundamental components of SOA upon which all higher-level functionality is built:[1]

1. **Services.** A service is the logical manifestation of some physical or logical resources (like databases, programs, devices, humans, etc.) and/or some application logic that is exposed to the network that may be executed in response to the arrival of messages.
2. **Messages.** A message is a unit of communication for exchanging information. All communication between services is facilitated by the sending and receiving of messages.

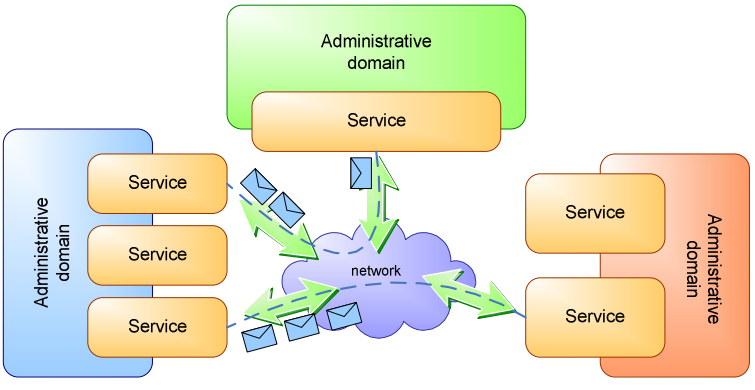


Fig. 1. The relationship between services and messages

Fundamentally Service-oriented systems are based on message-passing not on higher level abstractions like method calls[2]. It is this characteristic which enables loose-coupling since it allows services to be created and versioned in isolation based on message-level contracts. Services are not permitted to share knowledge of the internals of other services, but only exchange messages with them within the context of specific applications, as shown in Figure 1.

Given the importance of the two fundamental building blocks of SOA, in the following sections we explore the makeup of services and messages.

[1] Note that the W3C's Web Services Architecture document [33] presents a total of 16 components in its service-oriented model, plus a large number of interrelationships. This is not at odds with our view since it is a higher level view of the architecture.

[2] Method calls and events are often useful abstractions at the application level and most Web Services toolkits build such abstractions on top of the network-level messaging libraries. This can improve developer productivity but can be dangerous if developers fail to understand that once outside of a service's boundary, the abstractions which are presented to them as method calls and events are actually message exchanges.

The Anatomy of a Service

The architecture of a generic service is shown in Figure 2. Outwardly, a service is simply an addressable endpoint which processes messages. The internal architecture of a service is a classic N-Tier architecture utilising a message-router pattern. The beauty of Service-Oriented Architecture is that it is not revolutionary, but ordinary and therefore comprehensible to any proficient software engineer.

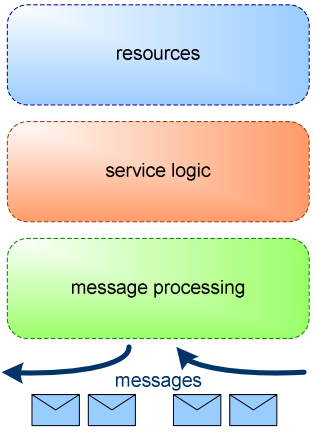


Fig. 2. The anatomy of a Web Service

The arrival of a message at the service endpoint normally causes the message to be validated by the messaging layer (although there may be situations where validation may not occur until further up the stack). Once validated, the message can be internally dispatched up the service stack and ultimately cause some processing at the logic layer of the service.

For ease of recovery and scalability, the service logic layer for an individual service implementation should manipulate only soft state; that is state which can be recomputed or recovered in the event of failure. Using soft state (effectively making the service implementation stateless) means that if a service fails, a backup service can be seamlessly brought online or the service can recover gracefully once the cause of the failure has been rectified. Maintaining only soft state in the service logic is important, since it alleviates the need for services to contain intricate recovery and consistency routines.

The uppermost layer in the stack is the resources, often representing persistent state, which may be shared by many copies of a service, and indeed by many services. This is where the enterprise data resides in a variety of hardened data storage mechanisms like (transactional) databases and queues and sometimes in less hardened media such as card files and human memories[3].

[3] The choice of enterprise storage is important for the architect of an individual service, yet fundamentally out-of-scope for an application which consumes that service.

Service Intercommunication

While understanding the tiered architecture of a service is of paramount importance for service architects, application architects have a different set of concerns. A service-oriented application is an aggregation of services, where the application orchestrates the message exchanges between services in order to facilitate some domain-specific work.

The underlying transport protocol for transferring messages may vary from application to application and may differ between different message exchanges within the same application. Depending on the level of quality of service required from a particular message exchange, an architect might elect to use a reliable message transport for a specific service (such as a queue) or use something more lightweight like TCP/IP. It is however important to distinguish that at this abstract level of architecture the fact that messages are transferred is the key notion, and the details of moving bits over the wire is architecturally transparent.

No matter how messages are ultimately moved across the network, messages themselves are *rich*. Rich messages are self-descriptive and meaningful in the context in which they are sent and received. To be truly meaningful, a message must contain all of the information that a service requires to execute its application logic. This not only reduces network overheads (which may be significant in an application which spans enterprises) but also supports stateless interactions (c.f. HTTP) which improves the prospects for scalability and reliability (as exemplified by the WWW) [9].

However, being meaningful does not necessarily imply any shared understanding beyond the structure of a message; it implies only that both sender and receiver understand the message within their own scopes, orchestrated by some overarching application or business process which understands the overall application or process semantics. That is, services themselves are unaware of the processes which they will support and are therefore able to be integrated with arbitrary partners and business processes.

The use of meaningful messages has ramifications for both service and application architects. For the service architect, fewer, richer, messages simplify the design of the service, while improving prospects for scalability, and simplifying fail-over fault tolerance. For the application architect, fewer, richer message exchanges enhance network performance and reduce the likelihood of transient failures disrupting normal application execution.

4 Applying Service Oriented Architectures to Web Services

Having discussed the Service-Oriented Architecture as a conceptual model, we can now proceed to concretise SOA in terms of Web Services. While a Web Service inherits the generic characteristics of a service, we place an additional constraint on the architecture of a Web Service that all messages exchanged must be in SOAP format. SOAP is the de facto standard message transfer protocol for cross-platform message-level interoperability and is universally supported. Furthermore since SOAP is extensible via its header construct, it has become the protocol of choice for the higher-level Web Services protocols (security, reliability, transactions and so forth).

While some may find it contentious, we believe that it is for the greater good that SOA + SOAP = Web Services, and that anything else (for example C++ objects with WSDL descriptions) is not.

While we constrain Web Services to using SOAP for interoperability reasons, for practical reasons we strongly advocate the addition of a service description to a Web Service to ease composition of services into applications since it describes the contract through which a service is willing to be bound. One obvious candidate for describing Web Services is WSDL [32] which can be used to describe the messages that a Web Service understands, and to a limited extend also describe the message exchange patterns for orchestration purposes.

A more powerful alternative is the SOAP Service Description Language (SSDL) [27] which can describe not only the format and chorography of message exchanges that a Web Service supports but has formal underpinnings which enables automated checking of the protocols that a service supports for deadlocks, consistency and so forth.

In addition to the syntactic aspects of a contract, policies can be used to describe the quality of service characteristics that a service supports. In the Web Services arena, WS-Policy [6] is an extensible framework for describing quality of service aspects of a Web Service, and has already been extended to include specific policy frameworks for security, secure conversations, and reliable messaging.

Adding the SOAP constraint and WSDL or SSDL and WS-Policy descriptions to services concretises the SOA abstract architecture presented in Figure 1 into an application and integration platform as shown in Figure 3.

In the following sections we will discuss the basic Web Services model, highlighting salient technologies where appropriate and showing how Web Services can be constructed and deployed in a manner which is adherent to the principles of service-orientation.

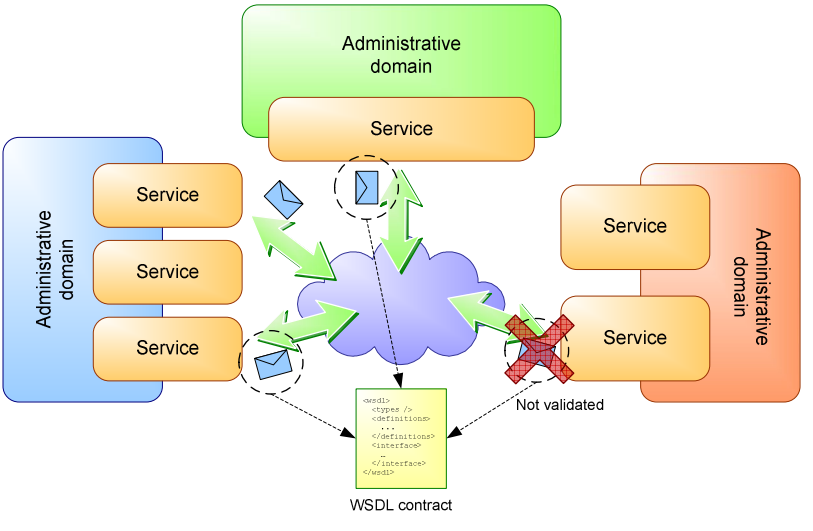


Fig. 3. Message structure and exchange patterns adhere to a WSDL contract

The Anatomy of a Web Service

A Web Service is the logical manifestation of some physical resources and application logic to the network and can be realised as network-capable units of software that implement logic, manage state, communicate via messages, and are governed by policy [21]. Like the abstract service architecture presented above, the canonical Web Service architecture is a multi-tiered artefact built from network, messaging, application, and state layers as shown in Figure 4.

The service logic layer deals only with solving the problem from the application domain. This layer should contain only soft state which, as mentioned earlier, confers benefits in terms of scalability and fail-over fault tolerance. A service containing only soft state typically delegates its requirements for replication and state consistency to the back-end data storage tier (which is designed precisely for such purposes).

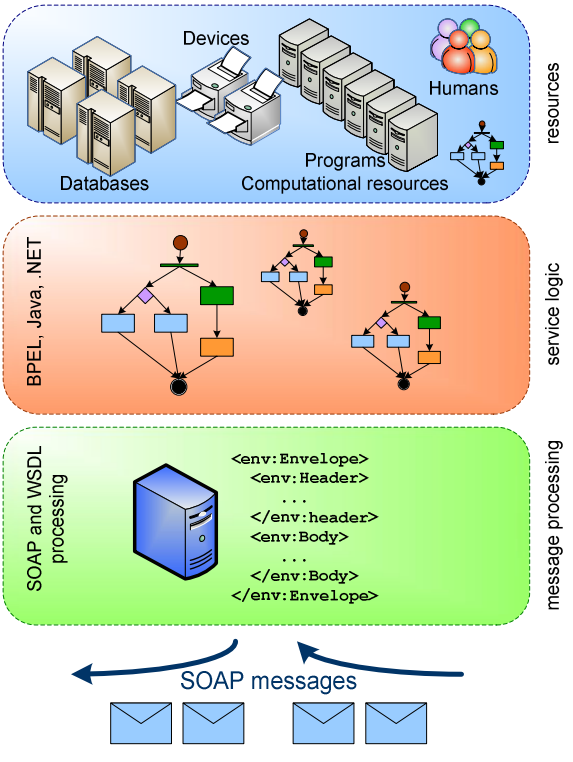


Fig. 4. The canonical architecture of a Web Service

If long-lived state is present within the application layer of a Web Service implementation, these benefits are significantly reduced and the service developer must implement appropriate failure recovery code as part of the application logic, which is both complex to develop and tends to impact scalability. Without failure recovery code, a newly recovered service would effectively be reset in terms of the

conversation it was having with its consumers. The consumers may not expect such behaviour and would most likely fail unless the consumer-service protocol has been designed to allow replays of conversations to occur. Delegating such responsibilities to the consumer (and by implication complicating the service-consumer protocol) is poor practice.

The messaging layer provides the programming abstractions for the application code to exchange messages with other services. The application code has to explicitly reason about those exchanges in terms of messages and message exchanges patterns. The application logic binds to, and directly manipulates message contents, as well as to notifications of the receipt of messages from other services (which may sensibly be delivered via events). In this way, the importance of crossing service boundaries is emphasised and service developers are encouraged to explicitly program services in terms of message interactions and the contents of the messages that make those interactions.

While some Web Services toolkits are beginning to support message-orientation for building Web Services (e.g. WSE [16], Indigo [19], Axis [1]), most toolkits still focus on presenting Web Services as objects. The method call paradigm is flawed in the general case [34] since it does not highlight the difference between invoking a method on a local object and exchanging messages with a remote Web Service. It is clear that the latency and failure modes of a distributed computing environment make distributed computing more complicated than centralised computing, and it is flawed to try to mask the differences [34] – even with Web Services.

Conversely a message-oriented API helps to corral developers into considering the application domain in SOA terms. This in turn loosens coupling since the focus shifts to (validated) messages which, because of extensibility features peppered throughout Web Services technologies (e.g., the introduction of metadata information in a message which can be safely ignored by its ultimate recipients but used by intermediaries to provide a particular quality of service, like security or transactions), can be evolved and versioned over time without breaking existing applications.

The network layer deals with routing of messages to and from the messaging layer. This layer is typically a piece of middleware rather than a component written by the service developer and goes by a variety of designations including the erstwhile “SOAP server”, “SOAP processor”, to the contemporary “Web Services platform” or “Web Services container.” While the network layer is predominantly implemented by off-the-shelf software, it is normal for the layer to be augmented during service deployments in order to expand its capabilities to support non-functional requirements such as security and transactions. Such augmentation is usually accomplished by registering “plugins” or “message processing handlers” from third-party toolkits (or written by the service developer) with the Web Services platform.

This separates the concerns of the functional requirements of the service which are addressed by the service implementation, and the non-functional requirements which are addressed by augmenting the message-processing layer. This decoupling permits the quality of service aspects of a service to evolve independently from the service implementation and permits different quality of service characteristics to be applied to different service endpoints which share the same implementation.

Given the layering of application, message, and network layers the options for mapping a message-exchange to a back-end action are wide open. While policy

descriptions [5] and semantics [30] of an action may augment a service's WSDL or SSDL contract, these should expose intent and not physical service characteristics. We implore service architects to use these layers to their best effect and decouple networking details from application implementation using messages as the interface.

SOAP Messages

In a Web Services environment, we impose the additional architectural constraint that messages are conveyed in SOAP format[4]. That is, for a Web Services-based application SOAP is the transfer mechanism, and in turn it is SOAP messages that are propagated by the underlying transport protocol(s). Whether those protocols are application protocols like HTTP or traditional transport protocols like TCP/IP is unimportant, what is important is that there is a standard model – the SOAP processing model – which provides the fundamental constraints for the entire distributed system architecture.

While in theory any SOAP style is valid, we would advise against using SOAP-RPC (that is rpc/encoded SOAP) because it encourages transmission of application-level objects as parameters to a (remote) procedure call. Instead it is better for the messages that are exchanged to resemble the kinds of business documents that the service's owner deals with. Thus, rather than encoding graphs of objects into SOAP messages, we suggest that un-encoded documents are exchanged (i.e., use of document/literal style SOAP). This advice is underscored by the fact that SOAP-RPC is optional (effectively deprecated) in SOAP 1.2 [31] and the rpc/encoded style is not supported by the WS-I Basic Profile 1.0a [36].

While traditionally SOAP messages do not contain addressing information and instead rely on the addressing of the underlying transport protocol, we add the further constraint that addressing should be part of the SOAP envelope to ensure transport protocol independence. Although there are as yet no open standards for embedding addressing data inside the SOAP envelope, the WS-Addressing [4] specification is an example of one suitable approach (which is fortunately nearing standardisation). In WS-Addressing, the addressing information is placed into a SOAP header block and

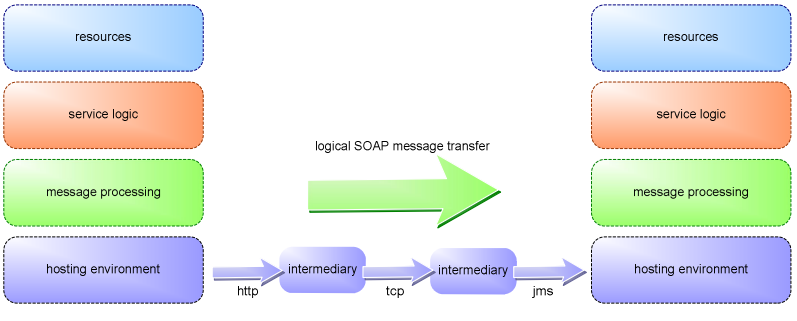


Fig. 5. Embedded addresses enable SOAP transport independence

[4] While WSDL can supports variety of protocol bindings, we assert that it is SOAP which characterises true Web Services and that it is SOAP which supports interoperability and extensibility that are key to the success of Web Services-based applications.

bound to the addressing mechanism of the underlying transport protocol by the sender of the message. Thusly equipped, SOAP messages can navigate arbitrary networks utilising a variety of protocols for various levels of quality of service and reliability as shown in Figure 5.

Upon receipt of a SOAP message the network layer is able to extract information from the header blocks and perform certain processing before the message is delivered up the stack. The information contained in the header blocks may be used, for example, to enlist transaction participants, to authenticate and authorise a message, to decrypt the contents of a message – that is, it provides *context* for the eventual processing of the message. A similar process happens in reverse when a service sends a message, where at the network layer protocol payload can be inserted into the headers, and sections of the body may be re-written according to the rules of the associated protocol – and provide context on the wire for recipients of the message. This is depicted in Figure 6.

The contents of SOAP headers are not fixed, which allows a service to determine its own protocol stack which maps onto the headers in the messages it exchanges. Such extensibility is supported in implementation terms by the plugins registered with the server platform as described above.

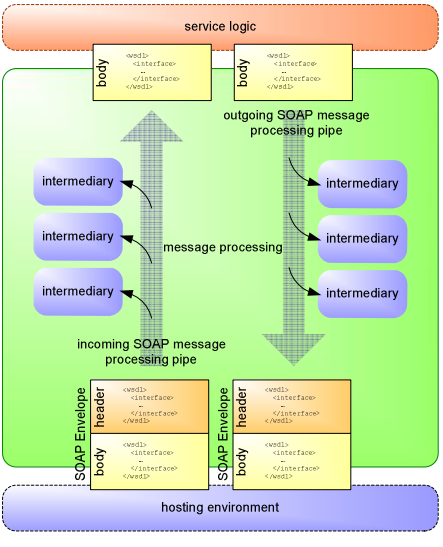


Fig. 6. SOAP extensibility and processing of SOAP messages

5 Architectural Principles for Building Grid Applications with Web Services

Web Services are computational entities which are deployed onto networks of arbitrary scale and as such present additional architectural challenges compared to intranet scale systems. To facilitate the deployment of robust and scalable Web

Services-based Grid applications, we suggest the following set of architectural and engineering best practices:

1. Services do not have interfaces in the object-oriented sense, but instead have an associated contract which defines the structure of messages that a service understands, the exchanges into which those messages can be composed, and other policy and quality of service characteristics which the service supports. It is this contract only which defines the externally observable characteristics of the service;
2. Services should be designed not to expose their implementation details or resource representations to consumers. Messages which contain data that directly maps onto a specific implementing object (or that alludes to the existence of such an object) encourage tight coupling and should be avoided. Similarly, mapping of operations at the contract level directly to method names in the service implementation couples implementation and contract and is considered poor practice;
3. Service-based application development should proceed as if the application's developers have no knowledge about the internals of any consumed services, even if they have intricate knowledge in reality. The only understanding a consumer has of the service is its contract through which it advertises supported message exchanges (and possibly policies). Taking such a strict view of service composition supports loose coupling and enables service implementations to evolve without breaking existing applications;
4. The API for service implementations and applications (where the implementation logic meets the network layer) should be cast in terms of the message exchanges that occur. An API which reflects the fact that (potentially) inter-domain message exchanges occur helps to reinforce the notion that services are autonomous and remote and promotes loose coupling.

These rules help to ensure loose coupling since both service and consumer are developed in mutual isolation in accordance to the service's advertised contract. Thus a consumer can choose to use any service which adheres to the same contract, and the service can provision functionality for any consumer which agrees to be bound by that contract. Furthermore, since all the network-level APIs are based on the message-passing paradigm the crossing of boundaries between local implementation and remote service actions is explicit[5].

6 Composite Applications and the WS-* Protocols

When aggregating services, especially from multiple administrative domains, into composite applications it is likely that we will require additional quality of service (QoS) features such as security, transactions, and reliable message delivery. Such QoS features are especially needed in the emergent field of Grid computing. Since the set of general principles for service-oriented applications is only concerned with

[5] While some older toolkits may wrap this to appear like objects with method calls (e.g. ASP.Net) the next generation of toolkits expose message-orientation directly to developers (e.g. WSE and Indigo, Axis).

message exchanges which are opaque from an architectural perspective, such quality of service features were not explicitly introduced in the architecture discussion.

However the fact that SOAP supports extensibility through its header mechanism means that quality of service protocol information can be packaged with application messages. Protocol information can be acted on by services to provide such features as non-repudiation, security, encryption, transaction (or activity) scope, and reliable message delivery (Figure 7).

While the WS-* protocols are fundamentally important and a key part of any Web Services toolkit, it is important to understand that they are not part of the model for Service-Oriented Architectures. The underlying architectural principles of services, contracts, and message-passing are pervasive, whereas other protocols are rolled into the stack (and the corresponding SOAP messages) only as needed. A simple parallel is found in the common object-oriented programming languages: object-orientation is a pervasive architectural principle whereas a platform's libraries are included in a program on an as-needed basis.

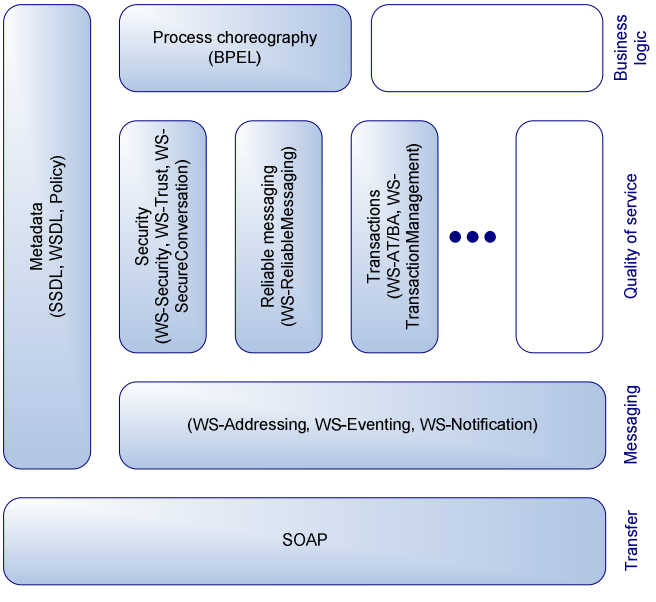


Fig. 7. The Web Services stack (adapted from [17])

However the WS-* protocols are themselves Web Services technology and are therefore not immune to the suggestions made in this paper. Indeed if a piece of infrastructure violates the principles discussed here, then there is far more scope for havoc than if a single service or application disregards them. The hope is that the developers of the WS-* specifications will maintain their largely good record of creating SOA-friendly, independent, composable protocols, and actively reject any work which does not align with those principles.

7 Conclusions

The terms 'Grid computing', 'service-orientation', and 'Service-Oriented Architecture' have been much overused recently and overloaded with different meanings. Furthermore the suite of technologies that is Web Services has been implicitly linked with SOA leading to false assumption of scalability and robustness of Grid applications simply by dint of the fact that some Web Services technologies (like SOAP and WSDL) have been used.

To counter such misguidance, this paper has presented what we believe to be a concise definition of the abstract SOA and the implied conceptual model based on the notion of services which exchange messages. We have shown that these fundamentals can then be concretised using Web Services technologies to provide the fabric for real world Internet-scale (Grid), service-oriented computing. In particular we advocate a constrained definition of a Web Service to be inline with our interpretation of SOA as a service which exchanges messages in SOAP format. Such services may have an associated contract (in WSDL or SSDL) which describes the format of messages, the message exchanges that service will participate in, and any additional Quality-of-Service features that the service supports.

Deriving from this architecture we proposed a set of simple rules for architecting services, which are designed to keep service implementations loosely coupled and scale to arbitrary size. These rules can be characterised as: Web Services use a message-passing paradigm where contracts govern the message exchanges that a service can participate in, and where no knowledge about the service or consuming application must be assumed or inferred.

We also discussed SOA-friendly mechanisms for aggregating services into applications, with mechanisms for orchestrating message exchanges at the application scope. Finally, we showed how quality of service protocols for "enterprise strength" computing can be layered on top of the architecture.

From these points, we propose that it is possible to support the levels of quality of service that enterprise-grade computing demands (security, reliability, transactions, etc) in a manner that is conformant with the principles of SOA, and thus derives the inherent benefits of that architecture. Furthermore we maintain that Grid applications can be built with today's Web Services technology.

References

[1] "Axis." http://ws.apache.org/axis/.
[2] "Service-Oriented Architecture (SOA) Definition." http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.
[3] "SETI@home." http://setiathome.ssl.berkeley.edu/.
[4] "Web Services Addressing (WS-Addressing)." http://msdn.microsoft.com/ws/2004/03/ws-addressing.
[5] "Web Services Policy (WS-Policy)." http://msdn.microsoft.com/ws/2002/12/policy.

[6] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratnam, M. Nottingham, H. Prafullchandra, C. v. Riegen, J. Schlimmer, C. Sharp, and J. Shewchuk, "Web Services Policy Framework (WS-Policy)." http://msdn.microsoft.com/webservices/
[7] default.aspx?pull=/library/en-us/dnglobspec/html/ws-policy.asp, 2004.
[8] Berkeley, "SETI@Home." http://setiathome.ssl.berkeley.edu/.
[9] P. Emeagwali, "Metaphysics of the Grid," vol. 2. http://www.gridtoday.com/03/0721/101710.html, 2003.
[10] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, pp. 115-150, 2002.
[11] I. Foster and D. Gannon, "Open Grid Services Architecture Platform (OGSA)," https://forge.gridforum.org/projects/ogsa-wg, 2003.
[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum 22 June 2002.
[13] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organisations," *International Journal of Supercomputer Applications*, vol. 15, 2001.
[14] H. He, "What is Service-Oriented Architecture." http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html, 2003.
[15] HP, "Grid & Utility Computing." http://devresource.hp.com/drc/topics/utility_comp.jsp.
[16] IBM, "On demand computing." http://www-1.ibm.com/grid/.
[17] Microsoft, "Web Services Enhancements (WSE)." http://msdn.microsoft.com/webservices/building/wse.
[18] Microsoft, "Web Services Specifications Index." http://msdn.microsoft.com/webservices/understanding/specs/.
[19] Microsoft, "Application Conceptual View." http://msdn.microsoft.com/library/en-us/dnea/html/eaappconland.asp, 2002.
[20] Microsoft, "Indigo." http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo, 2004.
[21] Microsoft, "Remarks by Bill Gates, Chairman and Chief Software Architect, Microsoft Corporation" Seamless Computing: Hardware Advances for a New Generation of Software" Windows Hardware Engineering Conference (WinHEC) 2004." http://www.microsoft.com/billgates/speeches/2004/05-04winhec.asp, 2004.
[22] M. Mullender and M. Burner, "Application Conceptual View." http://msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnea/html/eaappconland.asp: Microsoft, 2002.
[23] OASIS, "Web Services Composite Application Framework (WS-CAF)." http://www.oasis-open.org/committees/ws-caf.
[24] ORACLE, "Oracle Grid Computing." http://www.oracle.com/technologies/grid/.
[25] S. Parastatidis, J. Webber, and P. Watson, "Using Web Services to Build Grid Applications - The “No Risk” WSGAF Profile." http://www.neresc.ac.uk/ws-gaf/WSGAF-NoRiskProfile.pdf, 2004.
[26] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "A Grid Application Framework based on Web Services Specifications and Practices." http://www.neresc.ac.uk/ws-gaf, 2003.
[27] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "WS-GAF: A Grid Application Framework based on Web Services Specifications and Practices." Submitted for publication, 2004.

[28] S. Parastatidis, J. Webber, S. Woodman, D. Kuo, and P. Greenfield, "SOAP Service Description Language (SSDL)," School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-899, 2005.
[29] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture." http://msdn.microsoft.com/library/en-us/dnmaj/html/aj1soa.asp, 2004.
[30] W. Vogels, "Web Services Are Not Distributed Objects," *IEEE Internet Computing*, vol. 7, pp. 59-66, 2003.
[31] W3C, "Semantic Web." http://www.w3.org/2001/sw/.
[32] W3C, "SOAP Version 1.2 Part 1: Messaging Framework." http://www.w3.org/TR/soap12-part1.
[33] W3C, "Web Services Description Language (WSDL)." http://www.w3.org/2002/ws/desc.
[34] W3C, "Web Services Architecture." http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2004.
[35] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A Note on Distributed Computing," Sun Microsystems, Mountain View, CA SMLI TR-94-29, 1994.
[36] J. Webber and S. Parastatidis, "The WS-GAF Registry Service," presented at Building Service-based Grids Workshop (GGF 11), Honolulu, Hawaii, 2004.
[37] WS-I, "Web Services Interoperability (WS-I) Interoperability Profile 1.0a." http://www.ws-i.org.

Similarity Grid for Searching in Metric Spaces

Michal Batko[1], Claudio Gennaro[2], and Pavel Zezula[1]

[1] Masaryk University, Brno, Czech Republic
{xbatko, zezula}@fi.muni.cz
[2] ISTI-CNR, Pisa, Italy
claudio.gennaro@isti.cnr.it

Abstract. Similarity search in metric spaces represents an important paradigm for content-based retrieval of many applications. Existing centralized search structures can speed-up retrieval, but they do not scale up to large volume of data because the response time is linearly increasing with the size of the searched file. The proposed GHT* index is a scalable and distributed structure. By exploiting parallelism in a dynamic network of computers, the GHT* achieves practically constant search time for similarity range queries in data-sets of arbitrary size. The structure also scales well with respect to the growing volume of retrieved data. Moreover, a small amount of replicated routing information on each server increases logarithmically. At the same time, the potential for interquery parallelism is increasing with the growing data-sets because the relative number of servers utilized by individual queries is decreasing. All these properties are verified by experiments on a prototype system using real-life data-sets.

1 Introduction

Search operations have traditionally been applied to structured (attribute-type) data. Therefore, when a query is given, records exactly matching the query are returned. Complex data types – such as images, videos, time series, text documents, DNA sequences, etc. – are becoming increasingly important in modern data processing applications. A common type of searching in such applications is based on gradual rather than exact relevance, so it is called the *similarity* or *content-based* retrieval. Given a query object q, this process involves finding objects in the database D that are similar to q. It has become customary to assume similarity measure as a distance metric d so that $d(o_1, o_2)$ becomes smaller as o_1 and o_2 are more similar. Formally, (D, d) can be seen as the mathematical *metric space*. From an implementation point of view, the distance function d is typically expensive to compute. The primary challenge in performing similarity search for such data is to structure the database D in such a way that the search can be performed fast – that is a small number of distance computations is needed to execute a query.

Though many metric index structures have been proposed, see the recent surveys [3] and [8], most of them are only main memory structures and thus not suitable for a large volume of data. The scalability of two disk oriented metric indexes (the M-tree [4] and the D-index [7]) have recently been studied in [6]. The results demonstrate significant speed-up (both in terms of distance computations and disk-page reads) in comparison

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 25–44, 2005.

with the sequential search. Unfortunately, the search costs are also linearly increasing with the size of the data-set. This means that when the data file grows, sooner or later the response time becomes intolerable. Though the approximate versions of similarity search structures, e.g. the approximate M-tree in [1], improve the query execution a lot, they are also not scalable for large volumes of data. Classical parallel implementations of such indexes, see for example [15], have not been very successful either.

On the other hand, it is estimated that 93% of data now produced is in a digital form. The amount of data added each year exceeds exabyte (i.e. 10^{18} bytes) and it is estimated to grow exponentially. In order to manage similarity search in multimedia data types such as plain text, music, images, and video, this trend calls for putting equally scalable infrastructures in motion. In this respect, the Grid infrastructures and the Peer-to-Peer (P2P) communication paradigm are quickly gaining in popularity due to their scalability and self-organizing nature, forming bases for building large-scale similarity search indexes at low costs.

Most of the numerous P2P search techniques proposed in the recent years have focused on the single-key retrieval. Since the retrieved records either exist (and then they are retrieved) or they do not, there are no problems with the query relevance or degree of matching. The *Content Addressable Network* (CAN) [12] provides a distributed hash table abstraction over the Cartesian space. CAN allows efficient storage and retrieval of (key, object) pairs, but the key is seen as a point in n-dimensional Cartesian space. Such architecture has recently been applied in [13] to develop a P2P structure to support text retrieval in the Information Retrieval style. Since data partitioning coincides with the multidimensional space partitioning, such strategy cannot be applied to the generic problem of metric data such as strings compared by the *edit distance*. We are not aware of the existence of any distributed storage structure for similarity searching in metric spaces.

Our objective is to develop a distributed storage structure for similarity search in metric spaces that would scale up with (nearly) constant search time. In this respect, our proposal can be seen as a *Scalable and Distributed Data Structure* (SDDS), which uses the P2P paradigm for communication between computer nodes and assumes a Grid-like infrastructure for dynamically adjusting computational resources. We achieve the desired effects in a given (arbitrary) metric by linearly increasing the number of network nodes (computers), where each of them can act as a client and some of them can also be servers. Clients insert metric objects and issue queries, but there is no specific (centralized) node to be accessed for all (insertion or search) operations. At the same time, insertion of an object, even the one causing a node split, does not require immediate update propagation to all network nodes. A certain data replication is tolerated. Each server provides some storage space for objects and servers also have the capacity to compute distances between pairs of objects. A server can send objects to other peer servers and can also allocate a new server.

The rest of the paper is organized as follows. In Sec. 2, we summarize the necessary background information. Section 3 presents the GHT* distributed structure, its functionality, and basic analytic properties. Section 4 reports the results of performance evaluation experiments. Section 5 concludes the paper and outlines directions for future work. Appendix A contains the key algorithms of the GHT*.

2 Preliminaries

Probably the most famous scalable and distributed data structure is the LH* [11], which is an extension of the *linear hashing* (LH) for a dynamic network of computers enabling the *exact-match* queries. The paper also clearly defines the necessary properties of SDDSs in terms of *scalability*, *no hot-spots*, and *update independence*. The advantage of the *Distributed Dynamic Hashing* DDH [5] over the LH* is that it can immediately split overflowing buckets – LH* always splits buckets in a predefined order. The first tree-oriented structure is the *Distributed Random Tree* ([10]). Contrary to the hash-based techniques, this tree structure keeps the search keys ordered. Therefore, it is able to perform not only the exact-match queries, but also the *range* and the *nearest neighbor* queries on keys from a sortable domain, but not on objects from the generic metric space.

2.1 Metric Space Searching Methods

The effectiveness of metric search structures [3,8] consists in their considerable *extensibility*, i.e. the degree of ability to support execution of diverse types of queries. Metric structures can support not only the exact-match and range queries on sortable domains, but they are also able to perform similarity queries in the *generic metric space*. In this respect, the important *Euclidean vector spaces* can be considered a special case.

The mathematical metric space is a pair (D, d), where D is the *domain* of objects and d is the *distance function* able to compute distances between any pair of objects from D. It is typically assumed that the smaller the distance, the closer or more *similar* the objects are. For any distinct objects $x, y, z \in D$, the distance must satisfy the following properties:

$$\begin{array}{ll} d(x,x) = 0 & \textit{reflexivity} \\ d(x,y) > 0 & \textit{strict positiveness} \\ d(x,y) = d(y,x) & \textit{symmetry} \\ d(x,y) \leq d(x,z) + d(z,y) & \textit{triangle inequality} \end{array}$$

Though several sophisticated metric search structures have been proposed in literature, we have concentrated on one of the fundamental concepts called the *Generalized Hyperplane Tree* (GHT) [14], because it can easily be combined with the locally autonomous splitting policy of the DDH. For the sake of clarity of the further discussion, we describe the main features of the GHT in the following.

Generalized Hyperplane Tree – GHT. The GHT is a binary tree with metric objects kept in leaf nodes (buckets) of fixed capacity. The internal nodes contain two pointers to descendant nodes (sub-trees) represented by a pair of objects called the *pivots*. Pivots represent routing information, which is used in bucket location algorithms. As a rule, all objects closer to the first pivot are in the leaf nodes of the left sub-tree and the objects closer to the second pivot are in the leaf nodes of the right sub-tree.

The creation of the GHT structure starts with the bucket B_0. When the bucket B_0 is full, we create a new empty bucket, say B_1, and move some objects from B_0 (one

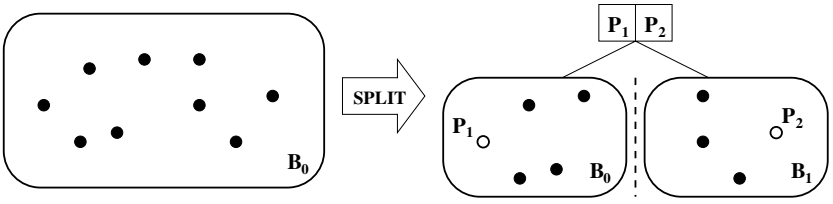


Fig. 1. Split of the bucket

half of them if possible) to B_1. In this way, we gain some space in B_0 (see Fig. 1 for illustration). This idea of split is implemented by choosing a pair of pivots P_1 and P_2 ($P_1 \neq P_2$) from B_0 and by moving all the objects O, which are closer to P_2 than to P_1, into the bucket B_1. The pivots P_1 and P_2 are then placed into a new root node and the tree grows by one more level. This split algorithm is an autonomous operation, which can be applied to any leaf node – no other tree nodes need to be modified. In general, given an internal node i of the GHT structure with the pivots $P_1(i)$ and $P_2(i)$, the objects that meet Condition (1) are stored in the right sub-tree. Otherwise, they are found in the left sub-tree.

$$d(P_1(i), O) > d(P_2(i), O). \tag{1}$$

To **Insert** a new object O, we first traverse the GHT to find the correct storage bucket. In each inner node i, we test Condition (1): if it is true, we follow the right branch. Otherwise, we follow the left one. This is repeated until a leaf node is found. Finally, we insert O into the leaf bucket and, if necessary, the split is applied.

In order to perform a similarity **Range Search** for the query object Q and the search radius r (i.e. a distance in a given metric), we recursively traverse the GHT following the left child of each inner node if Condition (2) is satisfied and the right child if Condition (3) is true.

$$d(P_1(i), Q) - r \leqslant d(P_2(i), Q) + r \tag{2}$$

$$d(P_1(i), Q) + r > d(P_2(i), Q) - r \tag{3}$$

Depending on the size of the radius r, Conditions (2) and (3) can be met simultaneously. This means that both of the sub-trees can contain qualifying objects and therefore both of them must be searched.

3 GHT*

In general, the scalable and distributed data structure GHT* consists of network nodes that can insert, store, and retrieve objects using similarity queries. The nodes with all these functions are called *Servers* and the nodes with only the insertion and query formulation functions are called *Clients*. The GHT* architecture assumes that:

– Network nodes communicate through the *message passing* paradigm. For consistency reasons, each *request message* expects a confirmation by a proper *reply message*.

– Each node of the network has a unique *Network Node IDentifier* (NNID).
– Each server maintains data objects in a set of *buckets*. Within a server, the *Bucket IDentifier* (BID) is used to address a bucket.
– Each object is stored exactly in one bucket.

An essential part of the GHT* structure is the *Address Search Tree* (AST). In principle, it is a structure similar to the GHT. In the GHT*, the AST is used to actually determine the necessary (distributed) buckets when data objects are stored and retrieved.

3.1 The Address Search Tree

Contrary to the GHT, which contains data objects in leaves, every leaf of the AST includes exactly one pointer to either a bucket (using BID) or a server (using NNID) holding the data. Specifically, NNIDs are used if the data are on a remote server. BIDs are used if the data are in a bucket on the local server. Since the clients do not maintain data buckets, their ASTs contain only the NNID pointers in leaf nodes.

In order to avoid hot-spots caused by the existence of a centralized node accessed by every request, a form of the AST structure is present in every network node. This naturally implies some replication. Due to the autonomous update policy, the AST structures in individual network nodes may not be identical – with respect to the complete tree view, some sub-trees may be missing. As we shall see in the next section, the GHT* provides a mechanism for updating the AST automatically during the insertion or search operations.

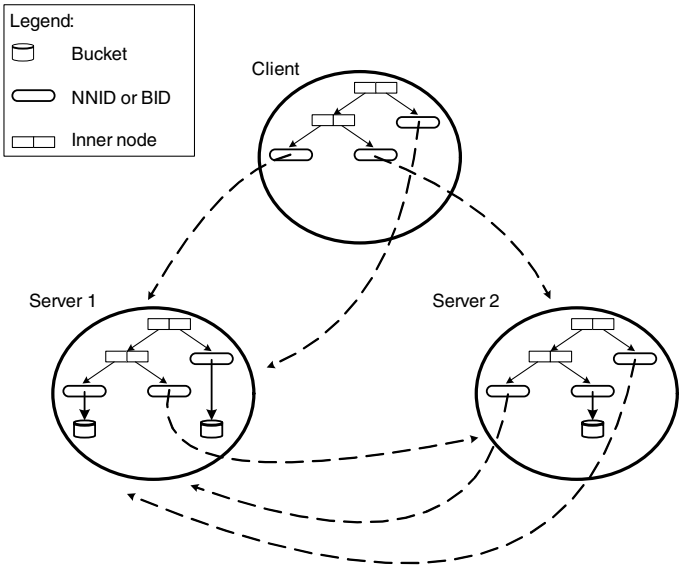


Fig. 2. Address Search Tree and the GHT* network

Figure 2 illustrates the AST structure in a network of one client and two servers. The dashed arrows indicate the NNID pointers while the solid arrows represent the BID pointers. In the following part, we describe the basic operations of the GHT*. Specification of the most important algorithms can be found in Appendix A.

Insert. Insertion of an object starts in the node asking for insertion by traversing its AST from the root to a leaf using Condition (1). If a BID pointer is found, the inserted object is stored in this bucket. Otherwise, the found NNID pointer is applied to forward the request to the proper server where the insertion continues recursively until an AST leaf with the BID pointer is reached. If a client starts the insertion, the AST traversal always terminates in a leaf node with an NNID pointer since clients do not maintain buckets.

In order to avoid repeated distance computations when searching the AST on the new server, a once-determined path specification in the original AST is also forwarded. The path sent to the server is encoded as a bit-string called BPATH, where each node is represented by one bit – "0" represents the left branch, "1" represents the right branch. Due to the construction of the GHT*, it is guaranteed that the forwarded path always exists on the target server.

Range Search. By analogy to insertion, the range search also starts by traversing its local AST, but as Sec. 2.1 explains, multiple paths can qualify. For all qualifying paths having a NNID pointer in their leaves, the request is recursively forwarded (including known BPATH) to identified servers until a BID pointer occurs in every leaf. If multiple paths point to the same server, the request is sent only once but with multiple BPATH attachments. The range search condition is evaluated by the servers in every bucket determined by the BID pointers.

3.2 Image Adjustment

An important advantage of the GHT* structure is the update independence. During object insertion, a server can split an overflowing bucked without informing the other nodes of the network. Consequently, the network nodes need not have their ASTs up to date with respect to the data, but the advantage is that the network is not flooded with multiple messages at every update. The updates of the ASTs are thus postponed and actually done when respective insertion or range search operations are executed.

The inconsistency in the ASTs is recognized on a server that receives an operation request with corresponding BPATH from another client or server. In fact, if the BPATH derived from the AST of the current server is longer than the received BPATH, this indicates that the sending server (client) has an out-of-date version of the AST and must be updated. The current server easily determines a sub-tree that is missing on the sending server (client) because the root of this sub-tree is the last element of the received BPATH. Such a sub-tree is sent back to the server (client) through the *Image Adjustment Message*, IAM.

If multiple BPATHs are received by the current server (which can occur in case of range queries) more sub-trees are sent back through one IAM (provided inconsistencies are found). Naturally, the IAM process can also involve more pairs of servers. Whenever

a server finds a NNID in its AST leaf during the path expansion, the request must be forwarded to the found server. This server can also detect an inconsistency and respond with an IAM. This image adjustment message updates the ASTs of all the previous servers, including the first server (client) starting the operation. This is a recursive procedure which guarantees that, for an insert or a search operation, every involved server (client) is correctly updated.

3.3 Logarithmic Replication Strategy

Using the described IAM mechanism, the GHT* structure maintains the ASTs practically equal on all servers. However, every inner node of the AST contains two pivots. The number of replicated pivots increases linearly with the number of servers used. In order to reduce the replication, we have also implemented a much more economical strategy which achieves logarithmic replication on servers at the cost of moderately increased number of forwarded requests. The image adjustment for clients is performed as described in Sec. 3.2.

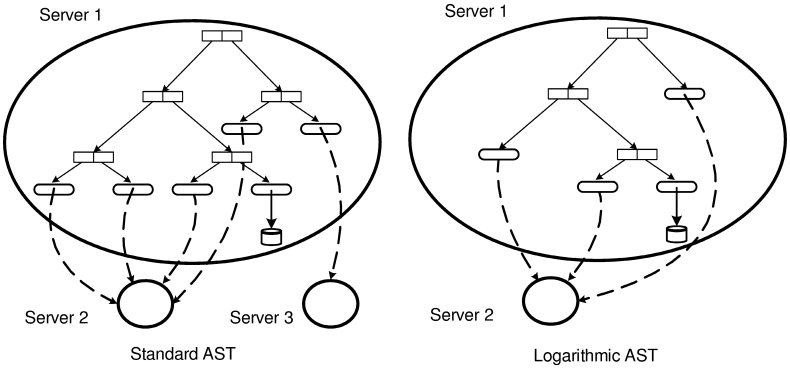


Fig. 3. Example of the logarithmic AST

Inspired by the *lazy updates* strategy from [9], our logarithmic replication scheme uses a slightly modified AST containing only the necessary number of inner nodes. More precisely, the AST on a specific server stores only the nodes containing pointers to local buckets (i.e. leaf nodes with BID pointers) and all their ancestors. However, the resulting AST is still a binary tree where all the sub-trees leading exclusively to leaf nodes with the NNID pointers are substituted by the leftmost leaf node of this sub-tree. The reason for choosing the leftmost leaf node is connected with our split strategy which always keeps the left node and adds the right one. Figure 3 illustrates this principle. In a way, the logarithmic AST can be seen as the minimum sub-tree of the fully updated AST. The search operation with the logarithmic replication scheme may require more forwarding (compared to the full replication scheme), but the replication is significantly reduced.

3.4 Storage Management

As we have already explained, the atomic storage unit of the GHT* is a bucket. The number of buckets and their capacity on a server are bounded by specified constant numbers, which can be different for different servers. Since the bucket identifiers are only unique within a server, a bucket is addressed by a pair (NNID, BID). To achieve scalability, the GHT* must be able to split buckets and allocate new storage and network resources.

Bucket Splitting. The bucket splitting operation is triggered by an insertion of an object into an already-full bucket. The procedure is performed in the following three steps:

1. A new bucket is allocated. If there is a capacity on the current server, the bucket is activated there. Otherwise, the bucket is allocated either on another existing server with free capacity or a new server is used (see Sec. 3.4).
2. A pair of pivots is chosen from the objects of the overflowing bucket (see Sec. 3.4).
3. Objects from the overflowing bucket that are closer to the second pivot than to the first one are moved to the new bucket.

Choosing Pivots. A specific choice of pivots directly affects the performance of the GHT* structure. However, the selection can be a time-consuming operation typically requiring many distance computations. To make this process smooth, we use an incremental pivot selection algorithm as proposed in [2]. It is based on the hypotheses that the GHT structure performs better if the distance between the pivots is high.

At the beginning, the first two objects inserted into an empty bucket become the candidates for pivots. Then, we compute distances to the current candidates for every additionally inserted object. If at least one of these distances is greater than the distance between the current candidates, the new object replaces one of the candidates so that the distance between the new pair of candidates grows. After a sufficient number of insertions is reached, it is guaranteed that the distance between the candidates, with respect to the bucket data-set, is large. When the bucket overflows, the candidates become pivots and the split is executed.

New Server Allocation. The GHT* scales up to processing a large volume of data by utilizing more and more servers. In principle, such an extension can be solved in several ways. In the Grid infrastructure, for example, new servers are added by standard commands. In our prototype implementation, we use a pool of available servers which is known to every active server. We do not use a centralized registering service. Instead, we exploit the *broadcast messaging* to notify the active servers. When a new network node becomes available, the following actions occur:

1. The new node with its NNID sends a broadcast message saying “I am here”. This message is received by each active server in the network.
2. The receiving servers add the announced NNID to theirs local pool of available servers.

Additional data and computational resources required by an active server are extended as follows:

1. The active server picks up one item from the pool of available servers. An activation message is sent to the chosen server.
2. With another broadcast message, the chosen server announces: "I am being used now" so that other active servers can remove its NNID from their pools of available servers.
3. The chosen server initializes its own pool of available servers, creates a copy of the AST, and sends to the caller the "Ready to serve" reply message.

3.5 Replication Analysis

In this section, we provide analytic formulas concerning the bucket load and the level of replication – the validity of proposed formulas is verified by experiments in Sec. 4. The necessary symbols are given in the following table:

symbol	description
S	total number of active servers
N	size of the data-set
n_b	maximal number of buckets per server
b_s	maximal bucket capacity in number of objects
T_L	length of the average path in the AST in number of nodes involved

The *load factor* L represents a relative utilization of active buckets. We define the load factor as the total number of objects stored in the GHT* structure divided by the capacity of storage available on all servers, thus:

$$L = \frac{N}{b_s \cdot n_b \cdot S} \tag{4}$$

A peculiar feature of the GHT* structure is the *replication factor* R – the objects used as routing keys (pivots) are replicated on the copies of the ASTs. We define the factor R as the ratio of the number of pivots that are replicated among the servers and the total amount of stored objects. Using Expression (4), the total number of stored objects can be determined as $N = L \cdot b_s \cdot n_b \cdot S$.

The worst replication factor occurs when all the servers have the fully updated ASTs. If maximally n_b buckets are maintained on each server, the AST must have $S \cdot n_b$ leaf nodes and $S \cdot n_b - 1$ inner nodes with $2(S \cdot n_b - 1)$ pivots. Consequently, $(S-1) \cdot 2(S \cdot n_b - 1)$ objects are replicated ($S-1$ because a version of each pivot is considered to be original on one server). Thus, the replication factor is given by

$$R = \frac{2(S-1)(S \cdot n_b - 1)}{S \cdot n_b \cdot b_s \cdot L} \cong \frac{2 \cdot S}{b_s \cdot L} \tag{5}$$

Obviously, such a replication factor grows linearly with the number of servers S.

With the logarithmic replication scheme (see Sec. 3.3), we do not replicate all the inner nodes of the full AST, but only those nodes that are on the paths ending in leaf nodes with the BID pointers. Corresponding to a balanced AST, the average length of

such a path is $T_L = \log_2(S \cdot n_b)$, thus the number of pivots on that path is $2(T_L - 1)$– the leaf nodes do not contain pivots. Since a server can store up to n_b buckets, a server AST can have maximally n_b paths from the root to a leaf node with the BID pointer. In such a situation, the number of pivots for all buckets of a server is bounded by $n_b \cdot 2(T_L - 1)$ because we neglect the common beginnings of the paths (such as the root that is present in all the paths). Then we can express the average replication factor as

$$R = \frac{(S-1) \cdot n_b \cdot 2(T_L - 1)}{S \cdot n_b \cdot b_s \cdot L} \cong \frac{2 \cdot \log_2(S \cdot n_b)}{b_s \cdot L} \tag{6}$$

Naturally, the factor is logarithmic with the number of servers S. Though our GHT* does not guarantee a balanced AST, our experiments in Sec. 4 reveal that the replication factor also grows in a logarithmic way when organizing real-life data-sets.

4 Performance Evaluation

In this section, we present results of performance experiments that assess different aspects of our GHT* prototype implemented in Java. To isolate the GHT* algorithms from the network and storage bucket implementation details, we use a layered architecture – two lowest layers implement the buckets and the network, the GHT* algorithms represent the middle layer, and the top layer forms the servers (clients) available to end users as an application.

Specifically, the bucket layer implements a particular storage strategy for buckets. It is responsible for allocation of buckets and it also maintains chosen limits (the maximal number of buckets per server and the maximal capacity of each bucket on a server). Moreover, the bucket layer implements the necessary strategy for choosing pivots.

The network layer is responsible for message delivery. Our prototype uses the standard TCP/IP network with multicast capability to report new servers. In general, we use the UDP protocol for sending messages. For transferring large volumes of data (for instance after a bucket split), the TCP connection is used.

The GHT* algorithms exploit the two lower layers and provide methods for the insertion of objects and the similarity search retrieval used in server (client) applications.

We conducted our experiments on two real-life data-sets. First, we have used a data-set of 45-dimensional vectors of color image features with the L_2 (Euclidian) metric distance function (*VEC*). Practically, the data in this set have a normal distribution of distances and every object has the same size. The second data-set is formed by sentences of Czech national corpus with the *edit distance* function as the metric (*TXT*). The distribution of distances in this data-set is rather skewed – most of the distances are within a very small range of values. Moreover, the size of sentences varies significantly. There are sentences with only a few words, but also quite long sentences with tenths of words.

We have focused on three different properties of the GHT*: insertion of objects and its consequences on the load and replication factors (Sec. 4.1), global range search performance (Sec. 4.2), and parallel aspects of query execution (Sec. 4.3).

4.1 Load and Replication Factors

In order to assess the properties of the GHT* storage, we have inserted 100,000 objects using the following four different allocation configurations:

n_b	1	5	5	10
b_s	4,000	1,000	2,000	1,000

We used a network of 100 PCs connected by a high-speed local network (100Mbps). These were gradually activated on demand by the split functions. In Fig. 4, we report load factors for the increasing size of the TXT data-set, because similar experiments for the VEC data-set exhibit similar behavior. Observe that the load factor is increasing until a bucket split occurs, then it goes sharply down. Naturally, the effects of a new bucket activation become less significant as the size of the data-set and the number of buckets grow.

In sufficiently large data-sets, the load factor was around 35% for the TXT and 53% for the VEC data-set. The lower load for the TXT data is mainly attributed to the variable size of the TXT sentences – each vector in the VEC data-set was of the same length. In general, the overall load factor depends on the pivot selection method and on the distribution of objects in the metric space. It can also be observed from Fig. 4 that the server capacity settings, i.e. the number of buckets and the bucket size, do not significantly affect the overall bucket load.

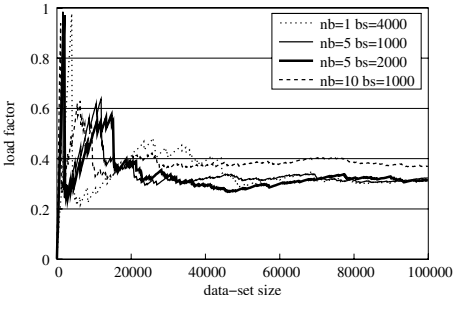


Fig. 4. Load factor for the TXT data-set

The second set of experiments evaluates the replication factor of the GHT* separately for the logarithmic and the full replication strategies. Figures 5a and 5b show the replication factors for increasing data-set sizes using different allocation configurations. We again report only results for the TXT data-set because the experiments with the VEC data-set did not reveal any different dependencies.

Figure 5a concerns the full replication strategy. Though the observed replication is quite low, it grows in principle linearly with the increasing data-set size, because complete AST is replicated on each computer node. In particular, the replication factor rises whenever a new server is applied (after splitting) and then it goes down slowly as new objects are inserted until another server is activated. The increasing number of used (active) servers can be seen in Fig. 6.

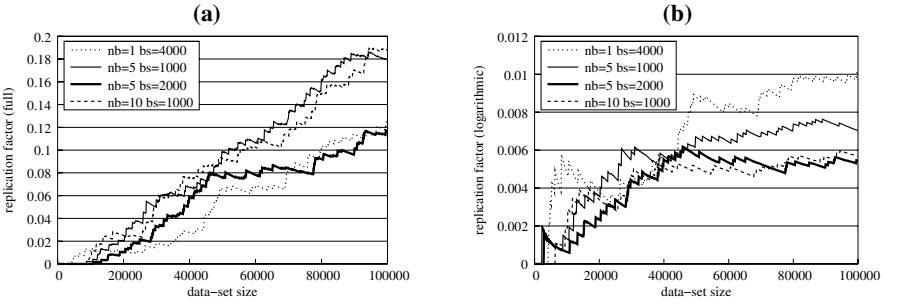


Fig. 5. Replication factor for the TXT data-set: (a) full replication scheme, (b) logarithmic replication scheme

Figure 5b is reflecting the same experiments, but using the logarithmic replication scheme. The replication factor is more than 10 times lower than it is for the full replication strategy. Moreover, we can see the logarithmic tendency of the graphs for all the allocation configurations. Using the same allocation configuration (for example when $n_b = 5, b_s = 2000$), the replication factor after 100,000 inserted objects is 0.118 for the full replication scheme and only 0.005 for the logarithmic replication scheme. In this case, the replication using the logarithmic scheme is more than twenty times lower than for the full replication scheme.

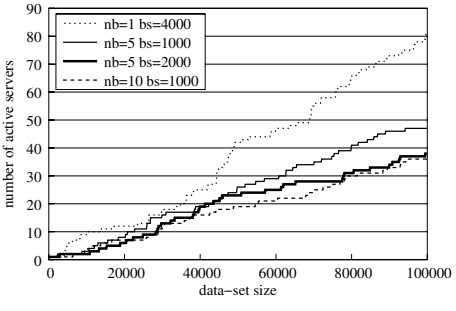


Fig. 6. Number of servers used to store the TXT data-set

Though the trends of the graphs are very similar, the specific instances depend on the applied allocation configuration. In particular, the more objects stored on one server, the lower the replication factor. For example, the configuration $n_b = 5, b_s = 1,000$ (i.e. maximally 5,000 objects per server) results in a higher replication than the configuration $n_b = 5, b_s = 2000$ with maximally 10,000 objects per server. Moreover, we can see that the configuration with one bucket per server is significantly worse than the similar setting with 5 buckets per server and 1,000 objects in one bucket. On the other hand, the other two settings with 10,000 objects per server (either as 10 buckets with 1 000 objects or 5 buckets with 2,000 objects) achieve almost the same replication.

Using the graph in Fig. 5a, it is possible to verify that the replication factor for the full scheme is actually bounded by Equation (5). For example, if $n_b = 10$, $b_s = 1,000$, and the TXT data-set of $N = 100,000$ objects are used, what we can read from Fig. 6 is that $S = 36$ and from Fig. 4 that $L = 0.37$. Therefore, the replication factor calculated using Equation (5) is

$$R \cong \frac{2 \cdot S}{b_s \cdot L} = \frac{2 \cdot 36}{1,000 \cdot 0.37} = 0.19$$

and the graph in Fig. 5a shows the value of 0.184. All the other cases can be easily verified by analogy.

The graph for the logarithmic scheme in Fig. 5b shows the value of 0.0055 for the same configuration. If we use Equation (6), we get the following upper-bound of the replication factor

$$R \cong \frac{2 \cdot \log_2(S \cdot n_b)}{b_s \cdot L} = \frac{2 \cdot \log_2(36 \cdot 10)}{1,000 \cdot 0.37} = 0.046$$

Due to the very pessimistic assumptions of the formula, the real replication is well below the analytic boundary.

4.2 Range Search Performance

In these experiments, we have analyzed the performance of the range searches with respect to the different sizes of query radii. We have measured the costs of the range search in terms of: (1) the number of servers involved in execution of a query, (2) the number of buckets accessed, (3) the number of distance computations in the AST and in all the buckets accessed. We have not used the query execution time as the performance criterion, because we could not ensure the same environment for all participating workstations on which other processes might be running at the same time. In this way, we were unable to maintain the deterministic behavior of the computer network.

Experiments in this section were computed using the GHT* structure with configuration $n_b = 10, b_s = 1,000$ which was filled with 100,000 objects either from the VEC or the TXT data-sets. We have used the logarithmic replication strategy. Each point of every graph in this section was obtained by averaging results of 50 range queries with the same radius and a different (randomly chosen) query object.

In the first experiment, we have focused on relationships between query radius sizes and the number of buckets (servers) accessed. Figure 7 reports the results of these experiments together with the number of objects retrieved. If the radius increases, the number of accessed servers grows practically linearly, but the number of accessed buckets grows a bit faster. However, the number of retrieved objects satisfying the query grows even exponentially. This is in accordance with the behavior of centralized metric indexes such as the M-tree or the D-index on the global (not distributed) scale. The main advantages of the GHT* structure are demonstrated in Sec. 4.3 when the parallel execution costs are considered.

Another important aspect of a distributed structure is the number of messages exchanged during search operations. Figure 8 presents the average number of messages

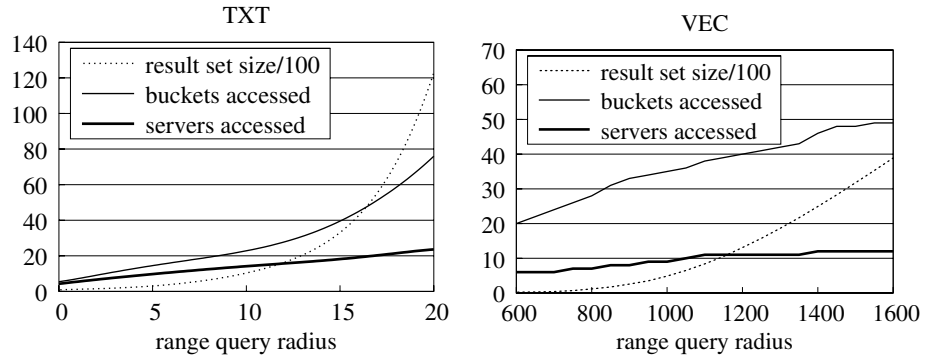


Fig. 7. Average number of buckets, servers, and retrieved objects (divided by 100) as a function of the radius

sent by a client and servers involved in a range search as a function of the query radii. We have measured the number of messages sent by servers (i.e. forwarded messages) and the number of messages sent by a client (this values are divided by 10) separately. In fact, the total number of messages is strictly related to the number of servers accessed. We have observed that even with the logarithmic replication strategy the number of forwardings is below 10% of the total number of messages sent during the query execution.

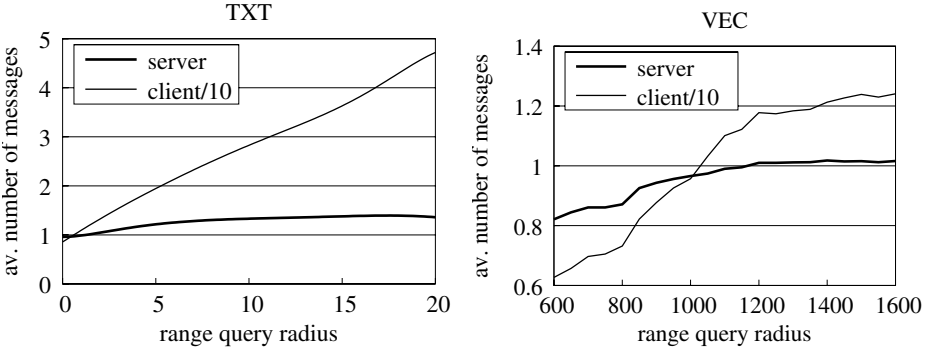


Fig. 8. Average number of messages sent by a client (divided by 10) and server as a function of the radius

In Fig. 9, we show the average number of distance computations performed by a client and the necessary servers during a query execution. We only focus on distance computations that are needed during the traversal of the AST (two distance computations must be evaluated per inner node traversed). We do not report the computations inside the accessed buckets, because they are specific to the bucket implementation strategy. In our current implementation, the buckets are organized in a dynamic list

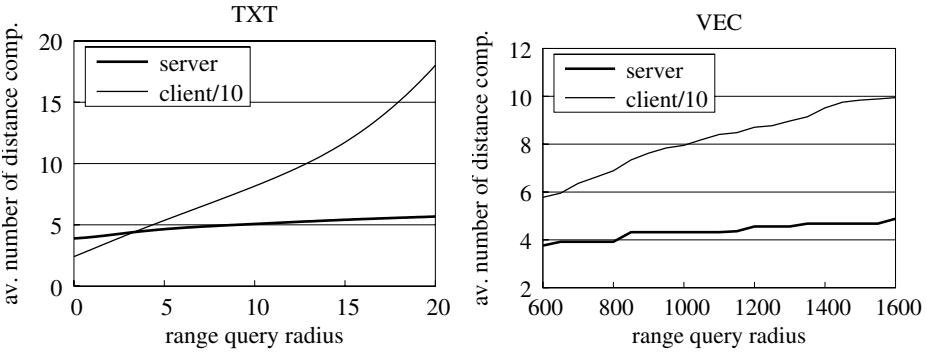


Fig. 9. Average number of distance computations in the AST for clients (divided by 10) and servers as a function of the radius

so the number of distance computations per bucket is simply given by the number of objects stored in the bucket.

4.3 Parallel Performance Scalability

The most important advantage of the GHT* with respect to the single-site access structures concerns its scalability through parallelism. As the size of a data-set grows, new server nodes are plugged in and their storage as well as the computing capacities are exploited. In the following, we experimentally study different forms of the *intraquery* and *interquery* parallelism to show how they actually contribute to the GHT* scalability. First, we consider the scalability from the perspective of a growing data-set and a fixed set of queries, i.e. the *data volume scalability*. Then, we consider a constant data-set and study how the structure scales up with the growing search radii, i.e. the *query volume scalability*.

We quantify the intraquery parallelism as the parallel response time of a range query. It is determined as the maximum of the costs incurred on servers involved in the query plus the search costs of the AST. For evaluation purposes, we use the number of distance computations (both in the AST and in all the accessed buckets) as the computational costs of a query execution. In our experiments, we have neglected the communication cost and we have assumed that all active servers start evaluating the query at the same time. This is reasonable, because the distance computations are much more time consuming than sending messages throughout the network.

The interquery parallelism is more difficult to quantify. For simplicity, we characterize the interquery parallelism as the ratio of the number of servers involved in a range query to the total number of servers. In this way, we assume that the lower the ratio, the higher the chances for other queries to be executed in parallel. Naturally, such assumption is only valid if each server is used with equal probability.

In summary, the intraquery parallelism is proportional to the response time of a query and the interquery parallelism represents the relative utilization of computing resources.

To evaluate the data volume scalability, we used the GHT* configuration of $n_b = 10$ $b_s = 1,000$. The graphs in Fig. 10 represent the parallel search time for two different query radii depending on the data-set size. The results are available separately for the vector (VEC) and the sentence (TXT) data-sets. By analogy, Fig. 11 shows the relative utilization of servers for two types of queries and growing data-sets. Each point in the graphs was obtained by averaging the outputs of 50 range queries with the same radius and different query objects.

Our experiments show that the intraquery parallelism remains very stable and the parallel search time is practically constant, i.e. independent of the data-set size. Though the number of distance computations needed for the AST traversal grows with the size of the data-set, this contribution is not visible. The reason is that the AST grows is logarithmic, while the server expansion is linear.

On the other side, the ratio characterizing the interquery parallelism is even decreasing as the data-set grows in size. This means that the number of servers involved during the query grows much more slowly than the number of active servers and thus the percentage of servers used to evaluate the query is dropping.

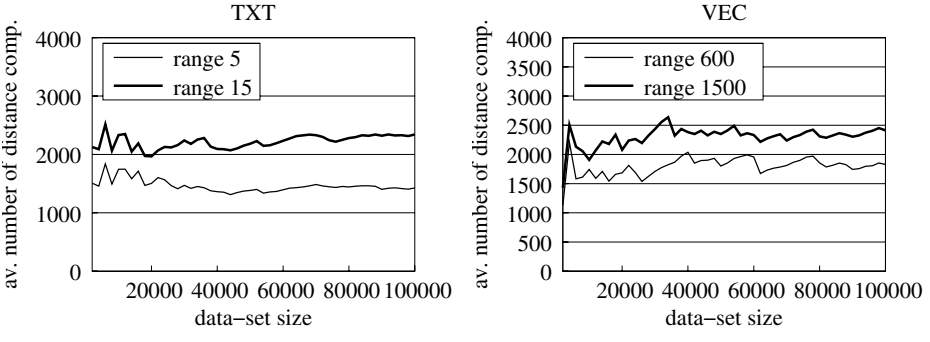


Fig. 10. Parallel costs as a function of the data-set size for two different radii

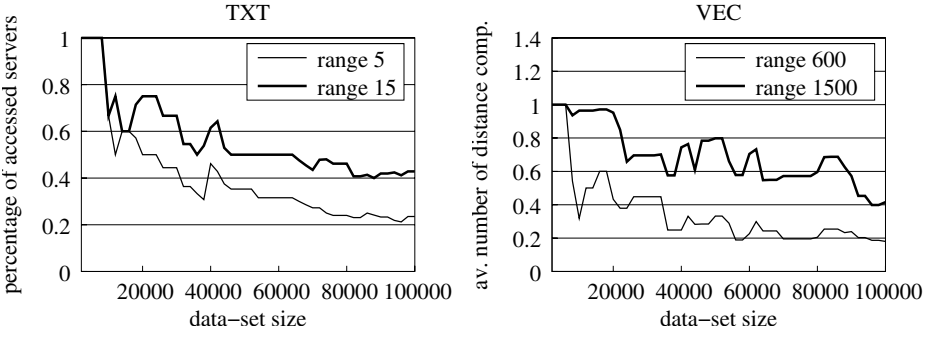


Fig. 11. Fraction of servers used to execute range queries as a function of the data-set size

5 Conclusions and Future Work

To the best of our knowledge, the problem of distributed index structures for metric data has not been studied yet. Our structure is thus the first attempt at combining properties of scalable and distributed data structures and the principles of metric space indexing.

The GHT* structure can store any arbitrary metric space data-set and meets all the necessary conditions of SDDSs. It is scalable in that it distributes the structure over more and more servers. In addition, the parallel search time becomes practically constant for arbitrary data volume and the larger the data-set the higher the potential for the interquery parallelism. It has no hot spot – all clients and servers use as precise addressing as possible and they all incrementally learn from misaddressing. Finally, updates are performed locally and a node splitting never requires sending multiple messages to many clients or servers. The main contributions of our paper can be summarized as follows: (1) we have defined a metric scalable and distributed similarity search structure; (2) we have specified key analytic properties of the structure; (3) we have experimentally validated the functionality on real-life data-sets; (4) we have presented key algorithms in the appendix. Our future work will concentrate on strategies for updates, presplitting policies, and more sophisticated policies for organizing buckets. We will also consider other metric space partitioning schemes (not only the generalized hyperplane) and we will study their suitability for implementation in distributed environments.

References

1. G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM TOIS*, 21(2):192–227, 2003.
2. B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pages 33–40, 2001.
3. E. Chavez, G. Navarro, R. Baeza-Yates, and J. Marroquin. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
4. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 426–435, 1997.
5. R. Devine. Design and implementation of DDH: A distributed dynamic hashing algorithm. In *Proc. of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO)*, volume 730, pages 101–114, Chicago, 1993.
6. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–13, 2003.
7. C. Gennaro, P. Savino, and P. Zezula. Similarity search in metric databases through hashing. In *Proc. of the 3rd Work. on Multimedia Inf. Retrieval*, pages 1–5, 2001.
8. G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.
9. T. Johnson and P. Krishna. Lazy updates for distributed search structure. In *Proc. of the ACM SIGMOD*, volume 22(2), pages 337–346, 1993.
10. B. Kröll and P. Widmayer. Distributing a search tree among a growing number of processors. In *Proc. of the ACM SIGMOD*, volume 23(2), pages 265–276, 1994.

11. W. Litwin, M. Neimat, and D. A. Schneider. LH* - a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.
12. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of ACM SIGCOMM 2001*, pages 161–172, 2001.
13. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. of Conference on Applications, tech., archit., and protocols for computer communications*, pages 175–186, 2003.
14. J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *IPL: Information Processing Letters*, 40:175–179, 1991.
15. P. Zezula, P. Savino, F. Rabitti, G. Amato, and P. Ciaccia. Processing m-trees with parallel resources. In *Proc. of the 8th International Workshop on Research Issues in Data Engineering (RIDE'98)*, pages 147—154, Orlando, FL, February 1998.

A The Algorithms

In this appendix, we provide a concise illustration of the most important algorithms of the GHT* structure.

When a client has to insert a new object, it first traverses its local AST. This task is accomplished by the *GetLeafNodeForObjectCl* function (see Algorithm A.1). It takes the inserted object O as input parameter and returns the leaf node containing the NNID of the server that (according to the current view) should accommodate the object. The function also returns the BPATH leading to the leaf node returned.

Algorithm A.1.
function *GetLeafNodeForObjectCl(O)*
    Create empty BPATH
    Let N *be the root node of AST*
    While N *is not leaf node*
        Let P_1 *and* P_2 *be the pivots stored in* N
        If $d(P_1, O) > d(P_2, O)$ ***then***
            Append ***1*** *to BPATH*
            Set N *to right descendant of* N
        Else
            Append ***0*** *to BPATH*
            Set N *to left descendant of* N
        End If
    End While
    Return *leaf node* N *and BPATH*

After the *GetLeafNodeForObjectCl* execution, the client sends an insert request message to the server NNID. The request contains the object to insert and the BPATH that was built during the tree traversal. The receiving server traverses its own AST exploiting the supplied BPATH as much as possible. When the BPATH is not long enough to cover the complete search (the client does not have a proper view of the AST), the traversing continues by using the standard distance computation to pivots extending the BPATH accordingly. The *GetLeafNodeForObjectSr* (see Algorithm A.2) implements the traverse operation of a server AST.

Algorithm A.2.
function *GetLeafNodeForObjectSr(O, BPATH)*
    Let N *be the root node of AST*
    While N *is not leaf node*
        If *next bit exists in BPATH* ***then***
            Let b *be the next bit from BPATH*
        Else
            Let P_1 *and* P_2 *be the pivots stored in* N
            If $d(P_1, O) > d(P_2, O)$ ***then***
                Let b *= 1*
            Else
                Let b *= 0*
            End If
            Append b *to BPATH*
        End If
        If b *is 1* ***then***
            Set N *to right descendant of* N
        Else
            Set N *to left descendant of* N
        End If
    End While
    Return *leaf node* N *and modified BPATH*

The result of *GetLeafNodeForObjectSr* is the leaf node pointing to the local bucket BID where an object should be stored, or to another server NNID (which implies forwarding). In the case of forwarding, the BPATH stored in the message is changed to the BPATH returned from the function *GetLeafNodeForObjectSr*. This situation also indicates that the client has an improper view and image adjustment is computed.

Algorithm A.3.
function *RangeTraverseNode(Q, r, node N, BPATH)*
    If N *is leaf node* ***then***
        Let $Return\_set$ *= [NNID of* N*, BPATH]*
        Return $Return\_set$
    End If
    Create empty set $Return\_set$
    Let P_1 *and* P_2 *be the pivots of* N
    If $d(P_1, Q) - r \leqslant d(P_2, Q) + r$ ***then***
        Let $Return\_set$ *=* $Return\_set$ *+ RangeTraverseNode*
            (Q, r, left subnode of N, BPATH + ***0****)*
    End If
    If $d(P_1, Q) + r > d(P_2, Q) - r$ ***then***
        Let $Return\_set$ *=* $Return\_set$ *+ RangeTraverseNode*
            (Q, r, right subnode of N, BPATH + ***1****)*
    End If
    Return $Return\_set$

In order to perform a range search, we use an algorithm similar to the algorithm for insertion. The main difference is that, in general, we can obtain a set of NNIDs representing servers on which the search is performed. This is because a range query

may follow both the sub-trees of the AST. The *RangeTraverseNode* function describes a recursive algorithm for traversing branches of the AST performed by clients during the range search (see Algorithm A.3). This function returns a set of pairs [NNID, BPATH] for every leaf node found.

The server again uses a set of BPATHs to traverse their ASTs first and possibly expands them to reach the leaf nodes. Also in this case, forwarding and IAM operations are applied, if necessary.

Organisation-Oriented Super-Peer Networks for Digital Libraries

Ludger Bischofs[1] and Ulrike Steffens[2]

[1] University of Oldenburg, FK-2, Software Engineering Group, 26111 Oldenburg, Germany
ludger.bischofs@informatik.uni-oldenburg.de
[2] OFFIS, Department of Business Information and Knowledge Management,
26121 Oldenburg, Germany
ulrike.steffens@offis.de

Abstract. The transition from traditional paper libraries to digital libraries enables new strategies for the use and maintenance of artifact collections. We present an organization-oriented super-peer network which represents the organizational structures of distributed digital libraries in a natural way and is able to integrate distributed resources as well as local devices. Organizational structures can be considered in searching and accessing distributed services and documents. The approach leads to a generalised support for the self-organization of widely distributed, loosely coupled, and autonomous digital library systems which can be structured by arbitrary inter- and intra-organizational relationships.

1 Introduction

The transition from traditional paper libraries to digital libraries enables new strategies for the use and maintenance of artifact collections. Collections are globally distributed and maintained by different organisations and even private persons. Digital binding techniques allow for construction of project specific reference libraries by reorganising existing library material and for the reintegration of project results [13]. As production, storage and classification of documents are now accomplished digitally, library support for intermediate and final results of collaborative writing can be achieved. Furthermore, the collection and organisation of assets other than documents, like for example works of art [14] or services [9], is possible by referencing them from within the digital library.

Taking advantage of digital libraries in the described manner calls for a flexible support by a system architecture which enables combination of collections against the background of different organisational, topical, and technical contexts, offering simple access to potential library users on the one hand and guaranteeing autonomy to library patrons on the other hand.

Distributed software development can be regarded as a special case of digital library utilisation, where developers or groups of developers are working on the same software geographically dispersed in time zones which might differ. As the use of a central repository for shared artifacts might have substantial drawbacks like slower and less reliable network connections, software developers rely on distributed artifact collections or, in other words, on distributed digital libraries of software engineering artifacts.

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 45–62, 2005.

In this paper we introduce the approach of organization-oriented super-peer networks in order to support a flexible distributed digital library architecture. The use of organization-oriented super-peer networks can be regarded as a general method in order to achieve flexibility and autonomy for both a loose and a tight coupling of digital library collections supporting inter- and intra-organizational library cooperation. This is achieved by adding a third layer on top of already existing concepts for representation of physical and virtual peer-to-peer networks. This additional layer is used to model organizational units and their relationships and can furthermore serve as a basis for integration of data and service resources into peer-to-peer networks as well as for realisation of look-up services which are able to consider organizational structures for query routing.

This paper is organised as follows. Section 2 offers some basic definitions from the area of peer-to-peer networks and classifies existing peer-to-peer architectures, before section 3 extends the existing approaches by adding another network layer for representation of organizational structures. This extension leeds to the concept of organization-oriented super-peer networks. Integration of additional resources like annotation services or version control systems into an organization-oriented super-peer network is demonstrated in section 4. Section 5 shows how an appropriate look up service reflecting a given organizational structure can be designed for an organization-oriented super-peer network. Two instances of organization-oriented super-peer networks for special digital libraries in the area of software development and for distributed digital libraries in general are explained in sections 6 and 7. After presenting related work in section 8 we conclude and outline some future work in section 9.

2 Virtual Peer-to-Peer Networks

Understanding of peer-to-peer networks as virtual networks forming an additional layer above the underlying physical networks is one foundation of the concepts presented in this paper. For this reason we first introduce some basic definitions closely related to peer-to-peer networks and describe the super-peer architectures we rely on within the context of a classification of peer-to-peer architectures.

2.1 Basic Definitions

At present, there exist different definitions for the term ”peer-to-peer” and related terms. An often cited definition from Shirky [16] understands peer-to-peer as a class of applications with specific characteristics. Steinmetz and Wehrle [17] describe ”peer-to-peer systems” as self-organised systems with equal, autonomous networked units (so called ”peers”) which operate without a central coordination aiming to share common resources. Schollmeier [15] defines the term ”peer-to-peer network” without giving a definition for the term ”peer-to-peer” itself. Client/server architectures are often understood as the opposite of peer-to-peer architectures but an actual definition of the peer-to-peer concept is missing in most publications on client/server architectures.

Beyond the definitions considered above, we are now going to define different terms related to the peer-to-peer concept in order to achieve a clear and detailed view on

the topic and to avoid misunderstandings. The terms ”peer-to-peer”, ”peer-to-peer network”, ”peer-to-peer protocol”, ”peer-to-peer system”, and ”peer-to-peer architecture” are explicitly defined in the following:

peer-to-peer: peer-to-peer (p2p in short) is a communication model of equal units (peers) which communicate directly with each other.

Equality between peers is realised by equipping them with both client- and server functionality. The peers' ability to act as client and server at the same time is also the most distinctive difference between p2p and client/server networks. Furthermore, peers are accessible by other peers directly without passing intermediary entities [15].

peer-to-peer network: A peer-to-peer network is a virtual network which is based on a physical network and enables connection between peers.

A virtual network is used to provide peers with necessary basic communication possibilities which the physical network as such does not offer. One example for these possibilities is bidirectional communication between peers which cannot be realised directly by the physical network, e.g. because of routers or firewalls blocking communication. Push or pull mechanisms or special routing methods, however, can simulate such a bidirectional connection within a virtual p2p network.

peer-to-peer protocol: A peer-to-peer protocol describes communication rules for the peer-to-peer network.

peer-to-peer system: A peer-to-peer system is a distributed system which is based on the peer-to-peer communication model.

A p2p system is usually based on a p2p network. However, this is no necessity, as a p2p system can directly rely on a physical network, if this physical network offers adequate communication possibilities. In both cases the underlying network is part of the p2p system.

peer-to-peer architecture: A peer-to-peer architecture describes the architecture of a peer-to-peer system.

2.2 Classification of Peer-to-Peer Architectures

The p2p communication model can be achieved by a family of different architectures. These architectures can be classified by the way they realise the idea of the p2p communication model.

Pure p2p architectures represent the direct counterpart of the model, because they are completely decentral, because peers have both client and server functionality, and because there are no central servers at all (cf. figure 1).

Hybrid p2p architectures are a combination of pure p2p architectures and client/server architectures. The difference to client/server architectures is that in hybrid p2p architectures only some services are offered by servers whereas other services are based

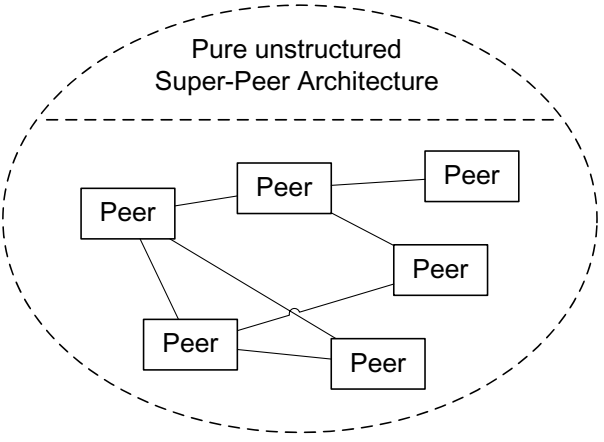


Fig. 1. Pure unstructured p2p architecture

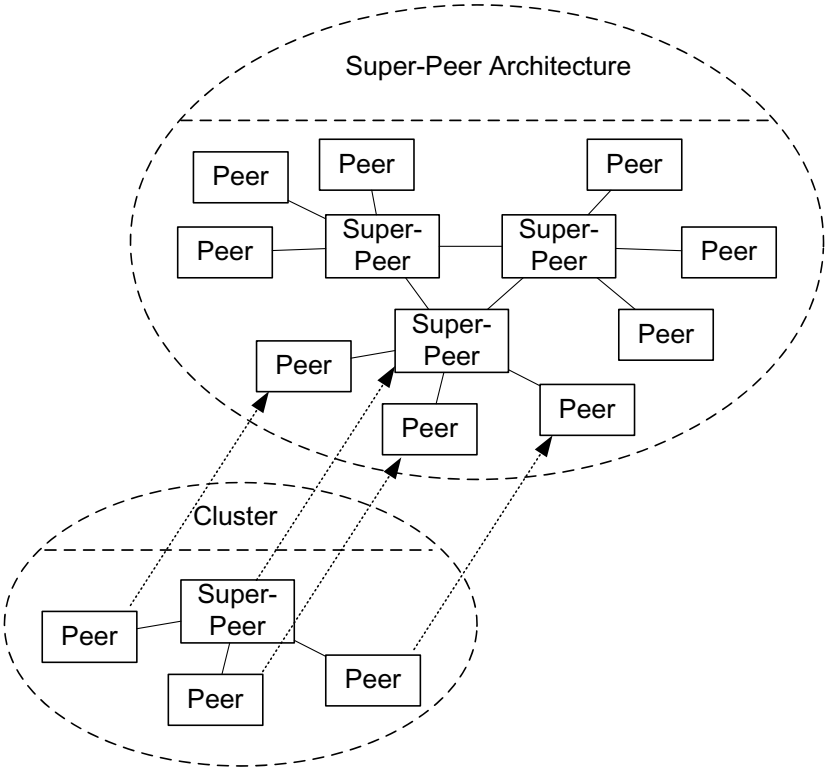


Fig. 2. Hybrid Super-peer architecture

on the p2p communication model. *Centralised architectures* and *super-peer architectures* are subclasses of hybrid p2p architectures, which are in frequent use.

In centralised architectures nodes with pure server functionality offer exclusive services. As the network depends on these servers, occurrence of "single points of failure" is one problem which has to be faced in centralised architectures. Napster is an example for a centralised p2p system.

Super-peer architectures are also called *hierarchical architectures*. They conceptually lie in between decentralised and centralised architectures. A *super-peer* is a peer which acts as a server for a set of normal peers and usually interacts with other super-peers (cf. figure 2). Normal peers are typically connected to one super-peer. A super-peer together with its normal peers is called a *cluster*. Because a super-peer is a "single point of failure" for its peers, super-peers are often implemented redundantly.

Super-peer networks have some advantages in comparison to pure p2p networks. They combine the efficiency of the centralised client-server model with autonomy, load balancing, and robustness of distributed search. They also take advantage of the heterogeneity of capabilities across peers. Parameters like cluster size and super-peer redundancy have to be considered by designing a super-peer network. Redundancy decreases individual super-peer load significantly whereas the cluster size affects behavior of the network in an even more complex manner [20,7].

3 Organisation-Oriented Super-Peer Networks

In order to support inter- and intra-organisational cooperation by use of p2p networks, peers and the functionality they offer have to be regarded within their organisational context. For this reason we introduce an organisational layer as a third layer above the physical network and the virtual p2p network. This section motivates the use of the organisational layer for representation of capabilities and requirements of organisational units involved in distributed cooperation. Furthermore, the three layers and the mappings between them are formally described.

3.1 A Layered Approach to Distributed Cooperation

Figure 3 illustrates the approach to model organisation-oriented super-peer networks within three different layers. The lowest layer is given by the physical structure of the network, which is characterised among other things by its partition into subnetworks, by protecting parts of the network with firewalls, and by applying different protocols.

The second layer represents a virtual network where in principle all peers are equal and each peer has its own ID independent of the underlying physical network. This layer's super-peers are high-capacity peers and as opposed to the simple peers fulfill additional tasks such as query routing. They are the main nodes within the network's subareas [20]. A uniform p2p protocol is used for communication, independent of the physical layer's protocols. The peers of the virtual layer are related in a virtual network uninfluenced by the underlying physical network structure.

In contrast to known p2p architectures, in our approach there is a third layer based upon the virtual layer which makes up the relationship between peers and organisational units using them. In this way, organisational structure is mapped to the virtual

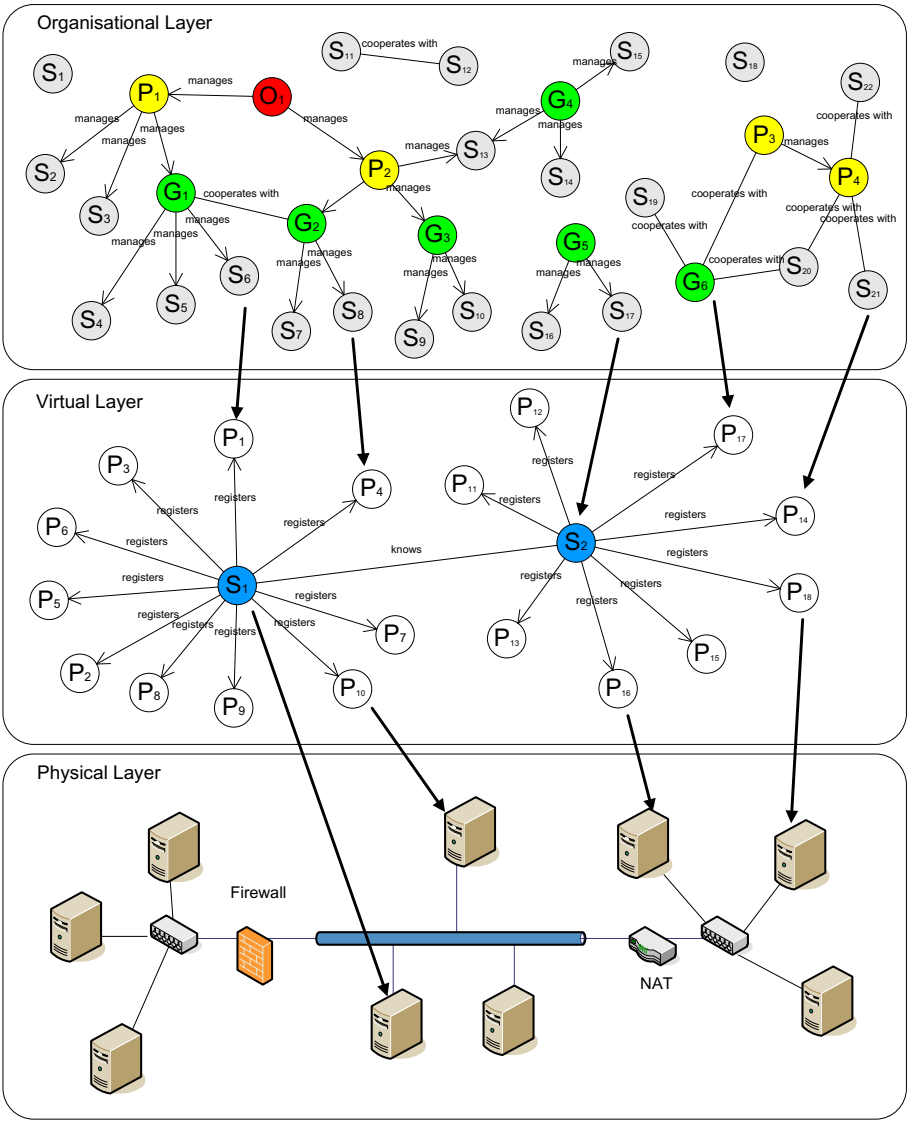


Fig. 3. Layers of an organisation-oriented super-peer network

network. With respect to this, peers in the virtual layer can take the role of a number of different organisational units and, depending on these roles, provide different data and services within the virtual network. The organisational structure is application- or domain-specific. Thus, organisational units as well as their relationships have to comply with the application domain.

Mapping organisational structure to the virtual layer enables self-organisation of organisational units. However, the benefit of an additional organisational layer is not

only given by this mapping. Moreover, the organisational layer exerts strong influence on the virtual layer with respect to searching organisational units and their relationships as well as data and services provided by them. The virtual layer can route queries taking into account organisational structure. Thereby network load can decrease and response time is reduced compared to search algorithms without organisational routing.

Example: The exemplary organisational layer in figure 3 contains the organisational unit types *Organisation* (O), *Project* (P), *Group* (G) and *Single* (S) and also represents the semantic relationships between them, which, in this example, signify either a hierarchical order or a cooperation between these units. Each organisational unit can offer data and services via the virtual network. The query "Search for document X in project P_1" formulated by an arbitrary peer would result in a search within the virtual layer, which could not be executed efficiently by the layer itself and which possibly would have to consider the whole network, i.e. all of the peers, in order to find the desired document. Furthermore, the search could not be successful, if the information of which peers belong to project P_1 remained unknown. By means of the organisational layer and its semantic relationships, however, the search can be executed in a targeted way. Assuming that a super-peer is able to send the query to project P_1 by disposing of metadata on organisational units registered with this super-peer, the query is handed on from P_1 to S_2, S_3 and G_1. G_1 routes the query to S_4, S_5, S_6 and G_2. G_2 passes the query to S_7 and S_8, so that finally document X is searched for within the whole project P_1, while other insignificant parts of the network remain unsearched. As soon as the document is found, search results might be sent directly to the requesting peer. Describing the above search sequence, we have assumed that peers in the virtual layer know their related peers within the organisational layer, in order to pass on the query directly, without having to consult the respective super-peer.

Figure 3 shows three levels of abstraction of an organisation-oriented super-peer system. The separate layers' structure and its evolution over time as well as the relationships between the layers are mathematically described in the following subsections.

3.2 The Organisational Layer

O is a set of elements of the organisational layer. Set O is the union of its disjoint subsets $O_1, ..., O_x$, which represent x different classes of organisational units.

$$O = O_1 \cup ... \cup O_x$$

The elements in O_1, ..., O_x are linked by relationships of different types. Sets O_1, ... O_x in conjunction with relationships R_1, ..., R_y build a structure S_O which describes the state of the organisational layer at a specific point in time t.

$$S_{O_t} = (O_1, ..., O_x, R_1, ..., R_y)$$

This formalisation doesn't aim at modeling concrete organisational structures, which are rather built by self organisation. Instead the formal considerations are used to provide data structures, which are stored by the peers and offer a basis for specialised search techniques.

Structural changes in the organisational layer over time can be described with sequences which consist of states in chronological order:

$$(S_{O_t}, ..., S_{O_{t+z}})$$

Since the system is based on a distributed super-peer architecture, data needed for representing the state of the organisational layer cannot be saved and managed centrally. Organisational units and peers work autonomously and only have information about parts of the system. Only the totality of the distributed information can give an overview over the state of the actual system.

Example: Figure 3 contains the following elements partitioned into four different classes of organisational units:

$$O_O = \{o_1\}$$
$$O_P = \{p_1, p_2, p_3, p_4\}$$
$$O_G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$$
$$O_S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, ..., s_{22}\}$$

The set O is composed as follows:

$$O = O_O \cup O_P \cup O_G \cup O_S$$

In the following, for reasons of simplicity, we only define relationships on the elements of set O to build a structure. Relationships could also be defined on O's subsets O_O, O_P, O_G and O_S instead. For $x, y \in O$ we define the following semantic relationships:

$x \overset{v}{\sim} y \Leftrightarrow x$ manages y.
$x \overset{b}{\sim} y \Leftrightarrow x$ owns y.
$x \overset{k}{\sim} y \Leftrightarrow x$ cooperates with y. This relation is bijective.

Relationships $\overset{v}{\sim}$, $\overset{b}{\sim}$, and $\overset{k}{\sim}$, however are only examples for possible semantic relationships. Arbitrary relationships coined by the respective organisational context are conceivable here.

We are now able to build a structure S_O which describes the organisational layer at the time t:

$$S_{O_t} = (O, \overset{v}{\sim}, \overset{b}{\sim}, \overset{k}{\sim})$$

3.3 The Virtual Layer

V is the set of elements of the virtual layer, i.e. the peers. The set V is the union of the subset V_P which contains all the "normal" peers with the subset V_S which contains all the super-peers. The subsets V_P and V_S are disjoint.

$$V = V_P \cup V_S$$

The following relationships can exist between the elements of V:

$s \overset{r}{\sim} p :\Leftrightarrow s$ registers p with $s \in V_S, p \in V_P$
$s_x \overset{k}{\sim} s_y :\Leftrightarrow s_x$ knows s_y with $s_x, s_y \in V_S, x \neq y$

Unlike the semantic relationships of the organisational layer, these two relationships are not defined individually but are given by the nature of virtual p2p networks.
The virtual layer at time t can be described by the following structure:

$$S_{V_t} = (V_P, V_S, \overset{r}{\sim}, \overset{k}{\sim})$$

3.4 The Physical Layer

P is the set of elements of the physical layer. This set represents physical units like PCs, PDAs and mobile phones, which can operate as peers. The physical layer is not focus of our investigation and can be described by the following simplified structure, again consisting of subsets of layer elements P_i and connection types C_j:

$P = P_1 \cup \ldots \cup P_x$
$S_{P_t} = (P_1, ..., P_x, C_1, ..., C_y)$

3.5 The Layer Mappings

The state of the separate layers at time t can be described by their structures:

$S_{O_t} = (O_1, ..., O_x, R_1, ..., R_y)$ (organisational layer)
$S_{V_t} = (V_P, V_S, \overset{r}{\sim}, \overset{k}{\sim})$ (virtual layer)
$S_{P_t} = (P_1, ..., P_x, C_1, ..., C_y)$ (physical layer)

Now the relationships between the layers can also be described. Set O can be mapped to set V, and set V can be mapped to set P. The mappings are neither injective nor surjective.

$f : O \rightarrow V$ (mapping of organisational units on peers)
$g : V \rightarrow P$ (mapping of peers on physical units)

The composition of f and g results in a mapping of organisational units to physical units:

$g \circ f : O \rightarrow P$ with $O \overset{f}{\rightarrow} V \overset{g}{\rightarrow} P$

4 Resource Integration

In addition to distributed artifacts cooperative work also calls for adequate access structures to distributed services: Cooperative work groups typically use a number of tools like version control systems (e.g. CVS or subversion) in software development or annotation services in digital libraries [1]. In the following, such tools, storages, or services

are understood as *resources*. In the context of a tight cooperation, organisational units share resources among each other over the network. The introduction of organisational structure described in section 3 allows for the modelling of such resource sharing. In software development for example, a project peer usually shares its resources with the involved groups, i.e. group peers are granted access to these resources. Group peers typically share their resources with single peers, i.e. all developers in a project can use resources which are connected to the appropriate project peer.

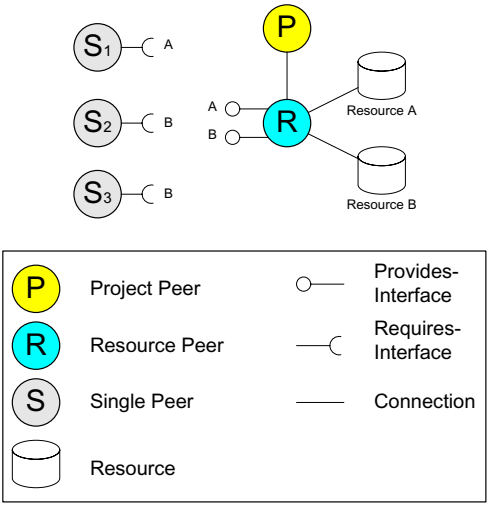


Fig. 4. Resource peer

Resources can directly be integrated into the organisational structure of a p2p network as can be seen in figure 4. Resource peer R connects to resources A and B in an application specific way. Externally another interface (e.g. in form of a web service) is offered to the rest of the peers. The peers connect to a resource through a special adapter which uses the corresponding external interface of the resource peer. The resource peer either manages access control lists of peers which have access to the resources or queries another peer, like project peer P in figure 4, whether to grant access to the requesting peer. All in all, the following steps are necessary to connect to a resource which is integrated into a peer-to-peer network:

1. The requesting peer (e.g. S_2) asks the resource peer for access to a resource.
2. The resource peer asks the responsible peer for access rights or consults its access control list.
3. Afterwards, access is granted or denied to the requesting peer.
4. If access is granted, the resource can be accessed.

By using the interfaces of a resource peer other peers can for example access documents which are stored on a CVS server or a local device. Another example for the

shared use of resources is the registration of group members for a forum or a groupware system carried out by their group peer via a resource peer.

5 Multi-tier Look Up Service

The lookup service is a central requirement for peer-to-peer systems. It assigns and locates resources and artifacts among peers [2]. Distributed "flat" peer-to-peer lookup services are e.g. Chord [18], CAN [10], Pastry [11] and Tapestry [21]. The approach presented in this paper is the introduction of a multi-tier lookup service where peers are organised in disjoint clusters as it is depicted in figure 5. Super-peers route messages along clusters to the destination cluster. Within the clusters, messages move through the peers' organisational structure. Organisational peer structures in combination with their super-peers form an organisation-oriented super-peer network which can be regarded as a generalisation of hierarchical super-peer networks as presented by [5]. Super-peers hold a common metadata index of available artifacts which are distributed over different organisational units. They are able to answer simple queries. Detailed queries additionally pass through the peers' organisational structure. Exchange of located artifacts typically takes place directly from peer to peer.

The most important benefits of the approach discussed in this paper are scalability and administrative autonomy. A super-peer can independently route messages within its cluster. Similarly, organisational unit peers can route the messages to subordinated peers using their own strategy. Queries to selected organisational units do not flood the entire network, but can be routed directly.

Super-peers are connected to each other and share a common metadata index of available artifacts. Physically, any peer which has enough computing power, storage capacity, and an adequate network connection can be chosen to be a super-peer. In figure 5, a ring of super-peers is shown for reasons of simplicity. There are other techniques like HyperCuP [12] that are able to increase scalability and reduce lookup time drastically. Peers which are registered at a super-peer make up a cluster as depicted in figure 5, where peers of the virtual layer and the physical layer's structure are left out for simplicity reasons.

There are different ways in which super-peers can collect metadata. Typically, peers which only loosely cooperate with other peers on the organisational level only register at the associated peers and send their metadata directly to the super-peer whereas peers which are involved in a close cooperation register at a peer which is in some respect superior to them and send their metadata there. Superior peers are a concept of the organisational layer used to represent arbitrary steps of an organisational hierarchy. Thus, the superior peer has extensive knowledge of its subordinated peers. The superior peers send the available metadata to other peers again higher in the hierarchical order until the super-peer is reached. As the use of them term "superior" already shows, this second approach is especially suitable for hierarchical organisational structures where cycles between peers are not likely to appear. One advantage of this approach is that not every peer in a network needs an internet connection to make its metadata available. Furthermore, superior peers have control over metadata which is sent to a super-peer.

Hence, superior peers have the ability to decide whether metadata of subordinated peers is made available or unavailable to superior peers or the global index, respectively.

Within the metadata index additional information on organisational units is stored. Moreover, metadata can also be extracted from resources which are connected to a resource peer. In figure 5 a simplified view of resource connections is illustrated in which the resource peer is not explicitly visible. In this case the associated peers are assumed to have the capabilities of a resource peer.

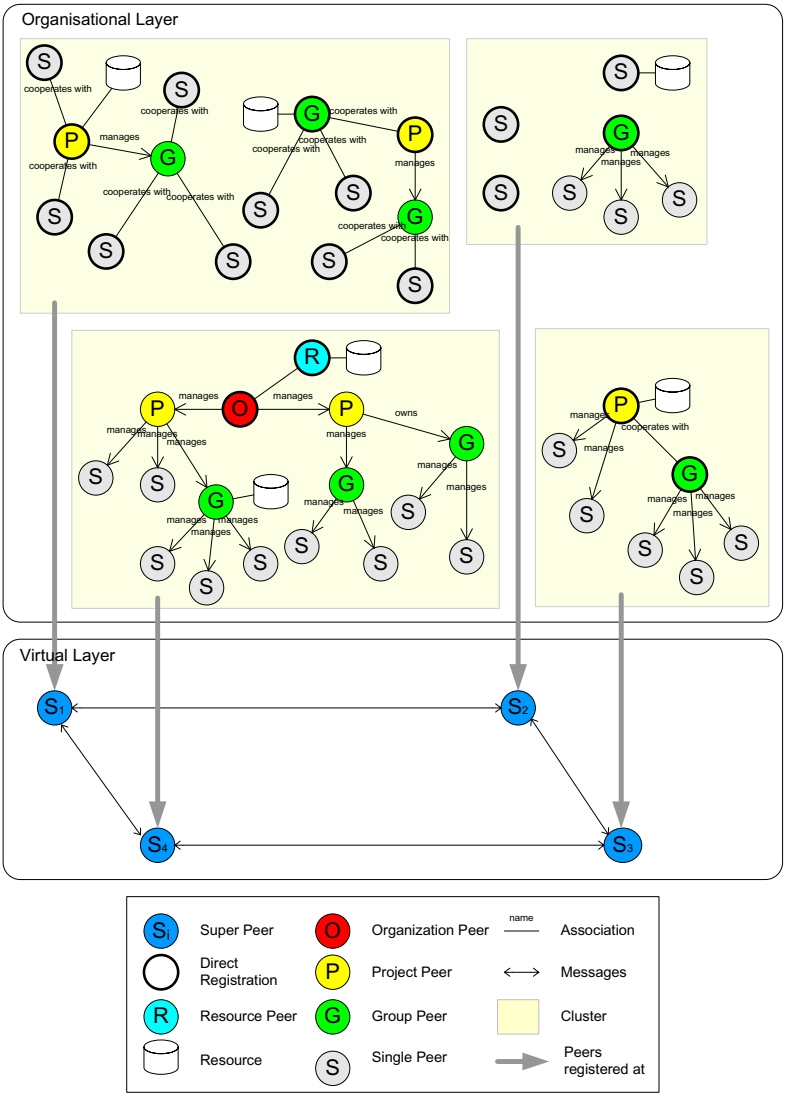


Fig. 5. Multi-tier lookup service in a hierarchical super-peer network

6 An Organisation-Oriented Super-Peer Network for Software Development

Distributed software development is a special case of distributed use of a digital library where the library is equipped with special artifacts embodied by the developers' shared working resources. Cooperation in this context can be adequately supported by organisation-oriented super-peer networks. For this purpose we now map the organisational structures of distributed software development to the organisational layer as described in section 3.

In case of distributed software development each developer can be considered a single peer called *developer peer* here. A developer peer can offer and access artifacts within the peer-to-peer network. Developers are often organised in groups which are managed by special *group peers*. Beyond, developers and groups of developers are organised in projects to reach a common goal. A *project peer* offers needed project management services. Organisations (e.g. an enterprise or institution) consist of projects, groups, and developers and are managed by *organisation peers*. Figure 6 refines the organisational layer of the example in figure 3 and depicts a logical view of a possible structure of peers which does not necessarily reflect the structure of the virtual and the physical network. P_2 and G_3 for example could physically reside on the same computer.

There are two different kinds of semantic relationship between organisational units. On the one hand, organisational units might closely cooperate, with one of them being superior to the other with regard to the underlying organisational hierarchy. The superior

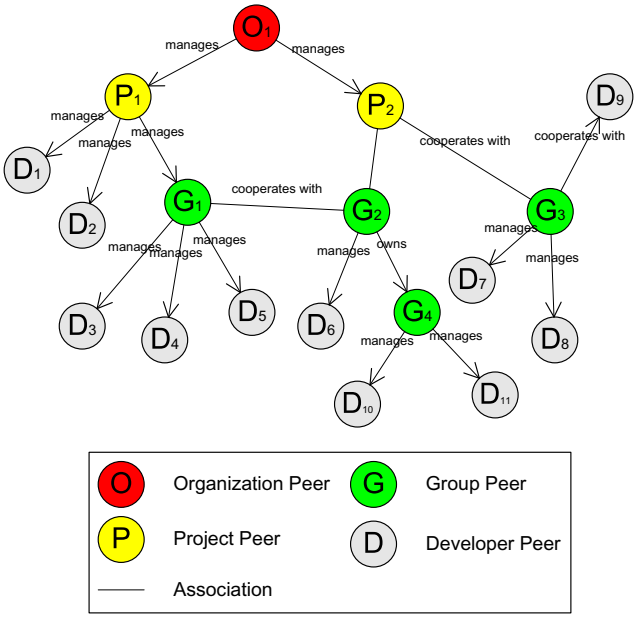


Fig. 6. Hierarchical structure of organisational units

unit then ”manages” or even ”owns” the subordinate unit (cf. figure 6). If a subordinate unit is owned by a superior unit, it cannot exist without the superior unit, whereas a unit managed by another unit might also exist on its own. Typically, to establish a ”manages”- or ”owns”-relationship a peer registers at a superior peer. The superior peer has the special ability to control its registered peers. In figure 6 for example group peer G_1 manages the developer peers D_3 to $_5$. Group peer G_4 is owned by peer G_2.

On the other hand, organisational units might only loosely cooperate as autonomous partners. No clear hierarchical structure can be extracted from such a ”cooperates with”-relationship. The groups G_1 and G_2 in figure 6 cooperate with each other. The same can be deduced for the developers within these groups so that access to resources of the respective other group can be granted to them. The developer at peer D_9 cooperates with group peer G_3. Since the ”cooperates with”-relationship signifies a loose connection only, a developer can cooperate with more than one group at a time.

7 An Organisation-Oriented Super-Peer Network for Digital Libraries

As already stated above, distributed software development can be regarded as special case of distributed digital library use. Based on the considerations in section 6 the organisation-oriented super-peer network for distributed software development is now generalised to support flexibility and self-organisation of widely distributed, loosely coupled, and autonomous general digital library systems. The architecture allows for search over collections of arbitrary artifacts as for example traditional documents, online books, digital images, and videos, which is a basic service requirement for digital libraries [3]. Furthermore, the network enables library users to also store, administer, and classify their own artifacts. Thus, it supports scenarios like the construction of personal or group reference libraries and collaborative authoring.

Figure 7 depicts an organisation-oriented super-peer network for digital libraries. Organisational units and their respective peer types are adapted to the situation within a general digital library. Persons are able to search for artifacts and offer artifacts on their own. They are therefore supplied with *person peers*. On the next level of the assumed organisation hierarchy, artifacts are grouped within collections managed by *collection peers*. Collection peers offer functionality relating to collection organisation as for example provision of a common classification scheme. A digital library can combine a number of different collections and is associated with a *digital library peer*. A digital library peer supports integration of different collections, for example by offering merging services for different classification schemes [6]. Furthermore, it manages access to digital library artifacts, for example by ensuring a certain mode of payment [19]. Person peers and collection peers can also exist independently from a superior peer and autonomously offer artifacts.

Person peers, collection peers, and digital library peers, as in the approach for distributed software development presented above, are clustered. The clusters again are connected via super-peers in the described manner (cf. sect. 5) and searched via super-peers and superior peers.

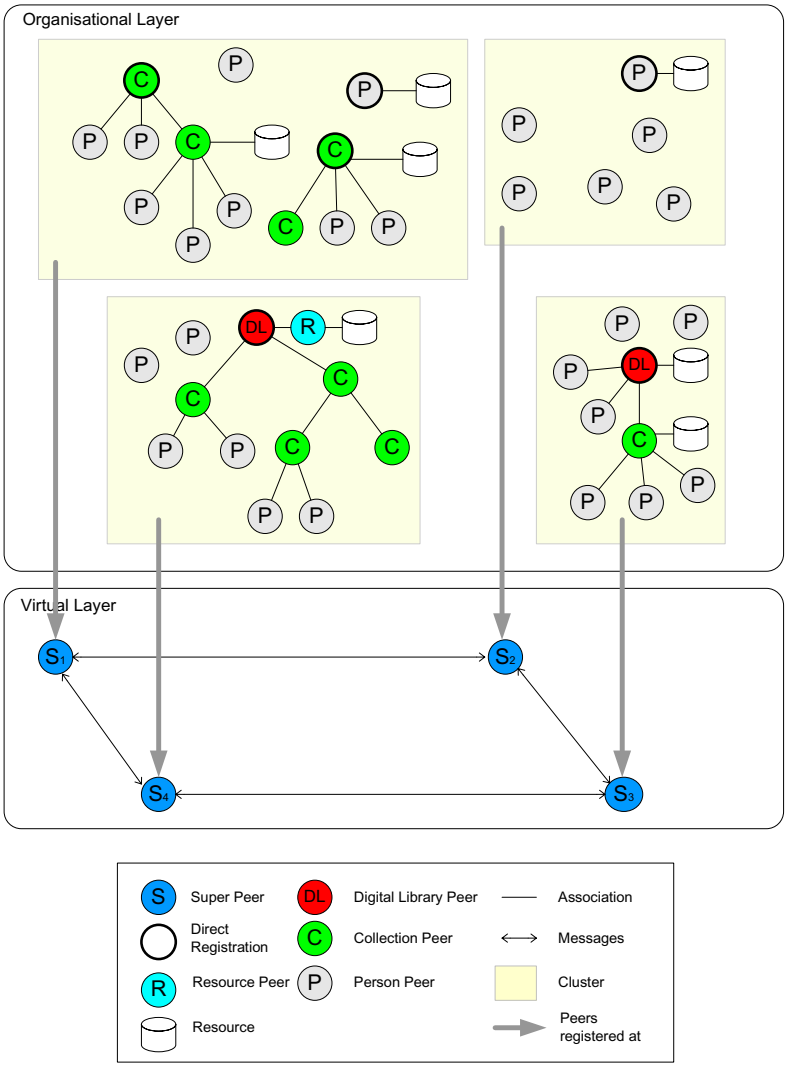


Fig. 7. An organisation-oriented super-peer network for digital libraries

The described network represents a first step in order to capitalise on the advantages of peer-to-peer technology for digital libraries. For personal or project reference libraries most of the upcoming traffic will remain within subareas of the network where co-workers cooperate intensely. For specialised collections which focus on special topics or special media types queries can be routed directly to selected collections or even library experts without flooding the entire network. Precision and query performance can hence be improved. Additionally, a self-organisation of collections and libraries is possible. Scalability and administrative autonomy are also ensured.

8 Related Work

Enabling interoperability among heterogeneous, widely distributed, autonomous digital library services is also the purpose of some other projects as described for example in [3]. The goal of establishing a manageable system of personal and project reference libraries as pursued in [13] also calls for flexibility which can be achieved by the use of a super-peer network.

Virtual networks organised as super-peer networks, which are used as a basis for the approach presented in this paper, have been described in a number of publications. The design of a super-peer network is e.g. presented in [20]. Costs and benefits of a new hybrid approach called structured super-peers is explored in [7]. It partially distributes lookup information among a dynamically adjusted set of high-capacity super-peers to provide constant-time lookup. Super-peer based routing strategies for RDF-based peer-to-peer networks are described in [8].

In hierarchical peer-to-peer systems, peers are organised into groups, and each group has its autonomous intra-group overlay network and lookup service. A general framework and scalable hierarchical overlay management is provided in [5].

None of the approaches above offers the ability to map arbitrary, even non-hierarchical organisational structures to an underlying p2p network as is achieved by introducing an organisational layer as presented in this paper.

9 Conclusions and Future Work

This paper has presented a three-layered organisation-oriented super-peer network approach for autonomous and self-organising digital libraries and artifact collections and has substantiated it by describing both a network instance for special libraries dedicated to software development tasks as well as a network instance for general distributed digital libraries.

One future challenge with regard to organisation-oriented super-peer networks is to analyse the dynamic behavior of the network, particularly if peers fail. Enhancing availability of artifacts and peer services by replication seems to be one promising approach to solve this problem [4].

Handling cyclic references within non-hierarchical organisational structures is another unsolved problem which has to be considered in order to ensure proper query routing within organisation-oriented p2p networks.

Furthermore, methods are needed to achieve a complete overview over the charcteristics of all organisational peers in order to enable valuable look-up services. Here again, considering temporary absence of peers is one of the major challenges.

Another issue more closely related to the area of digital libraries is to gain further understanding on how the presented approach can be refined against the background of reference libraries. The use of project peers as they have already been introduced for distributed software development could be an option. Yet, the project peers have to be adequately fit into the overall structure of the network.

References

1. M. Agosti, N. Ferro, I. Frommholz, and U. Thiel. Annotations in Digital Libraries and Collaboratories - Facets, Models and Usage. In *Proc. of the 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2004)*, pages 244–255, Bath, Great Britain, 2003. Springer.
2. H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, 46:43–48, 2003.
3. M. Baldonado, C.K. Chang, L. Gravano, and A. Paepcke. The Stanford Digital Library Metadata Architecture. *Journal on Digital Libraries*, 1(1):108–121, 1997.
4. L. Bischofs, S. Giesecke, W. Hasselbring, H. Niemann, and U. Steffens. Adaptive replication strategies and software architectures for peer-to-peer systems. In *8th International Workshop of the DELOS Network of Excellence on Digital Libraries on Future Digital Library Management Systems (System Architecture and Information Access)*, Schloss Dagstuhl, Germany, 2005.
5. L. Garces-Erice, E.W. Biersack, K.W. Ross, P.A. Felber, and G. Urvoy-Keller. Hierarchical Peer-to-peer Systems. In *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, Klagenfurt, Austria, 2003.
6. F. Matthes, C. Niederée, and U. Steffens. C-Merge: A Tool for Policy-Based Merging of Resource Classifications. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 5th European Conference, ECDL 2001*, Darmstadt, Germany, September 2001. Springer.
7. A.T. Mýzrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In *The Third IEEE Workshop on Internet Applications*, San Jose, California, 2003.
8. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-Peer-Based Routing Strategies for RDF-Based Peer-to-Peer Networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):177–186, 2004.
9. C. Niederée, U. Steffens, and M. Hemmje. Towards digital library mediation for web services. In *EurAsian Workshop on Knowledge Foraging for Dynamic Networking of Communities and Economies 2002, Shiraz*, October 2002.
10. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.
11. A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
12. M.T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In *Agents and Peer-to-Peer Computing, First International Workshop, AP2PC 2002, Revised and Invited Papers*, pages 112–124, Bologna, Italy, July 2002. Springer.
13. J.W. Schmidt, G. Schröder, C. Niederée, and F. Matthes. Linguistic and Architectural Requirements for Personalized Digital Libraries. *International Journal of Digital Libraries*, 1(1), 1997.
14. J.W. Schmidt, H. Sehring, M. Skusa, and A. Wienberg. Subject-Oriented Work: Lessons Learned from an Interdisciplinary Content Management Project. In *Proceedings of the Fifth East-European Conference on Advances in Databases and Information Systems*. Springer, 2001.

15. R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *1st International Conference on Peer-to-Peer Computing (P2P 2001)*, Linköping, Sweden, 2001. IEEE Computer Society.
16. C. Shirky. What Is P2P...And What Isn't. O'Reilly Network, 2001. http://www.openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html.
17. R. Steinmetz and K. Wehrle. Peer-to-Peer-Networking und -Computing. *Informatik-Spektrum*, 27(1):51–54, 2004.
18. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord - A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160. ACM Press, 2001.
19. R. Weber. Chablis - Market Analysis of Digital Payment Systems. Technical report, Technische Universitaet Muenchen, Institut fuer Informatik, 1998.
20. B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *IEEE International Conference on Data Engineering, 2003*, San Jose, California, 2003.
21. B.Y. Zhao, L. Huang, S.C. Rhea, J. Stribling, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE J-SAC*, 22:41–53, 2004.

A Combined Hyperdatabase and Grid Infrastructure for Data Stream Management and Digital Library Processes

Manfred Wurz, Gert Brettlecker, and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology,
Eduard-Wallnöfer–Zentrum 1, A–6060 Hall in Tyrol,
Austria
{manfred.wurz, gert.brettlecker, heiko.schuldt}@umit.at

Abstract. Digital libraries in healthcare are hosting an inherently large and continually growing collection of digital information. Especially in medical digital libraries, this information needs to be analyzed and processed in a timely manner. Sensor data streams, for instance, providing continuous information on patients have to be processed on-line in order to detect critical situations. This is done by combining existing services and operators into streaming processes. Since the individual processing steps are quite complex, it is important to efficiently make use of the resources in a distributed system by dynamically parallelizing operators and services. The Grid vision already considers the efficient routing and distribution of service requests. In this paper, we present a novel information management infrastructure based on a hyperdatabase system that combines the process-based composition of services and operators needed for sensor data stream processing with advanced grid features.

1 Introduction

Digital libraries in healthcare are increasingly hosting an inherently large and heterogeneous collection of digital information, like electronic journals, images, audios, videos, biosignals, three dimensional models, gene sequences, protein sequences, and even health records which consist of such digital artefacts. Medical digital libraries therefore have to organize repositories managing this medical information [17] and to provide effective and efficient access to it. In addition, a central aspect is the collection, aggregation, and analysis of relevant information.

Due to the proliferation of sensor technology, the amount of continuously produced information (e.g., biosignals or videos) in medical digital libraries will significantly grow. These data streams need sophisticated processing support in order to guarantee that medically relevant information can be extracted and derived for further storage, but also for the on-line detection of critical situations. Biosignals, like ECG recordings, contain relevant information derived from the evaluation of characteristic parameters, e.g., the heart rate, and their deviation from average. In some cases, even the combination of different biosignals is needed for the extraction of relevant information, such as a comparison of heart rate and blood pressure. *Data stream management* (DSM) addresses the continuous processing of streaming data in real-time. *Hyperdatabases* [22],

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 63–79, 2005.

in turn, provide a flexible and reliable infrastructure for data stream management [5]. Therefore, because of the streaming origin of parts of the information stored in medical digital libraries, the latter will significantly benefit from hyperdatabase infrastructures incorporating DSM.

Due the service-orientation and the distributed nature of digital libraries (i.e., information is made available by means of services), *grid infrastructures* are very well suited as basis for digital library applications. The composition of services and DSM operations can be realized by means of processes. The grid then has to support the efficient routing of service requests among different service providers. Considering the heterogenous and ever-changing nature of such environments, the need for dynamic binding of services during runtime is essential to achieve efficient resource usage [32]. Focusing on efficient routing and usage of available resources, it appears appealing to have services available that are able to split incoming requests and parallelize them according to the current status of the grid. In this paper, we introduce an approach to enable existing services to do so, without even changing existing and thoroughly tested functionality. By attaching two additional services, a *'Split Request'* and a *'Merge Result'* service, and by using an infrastructure component termed *'Dynamizer'*, a behavior as described can be realized.

A very challenging aspect in process-based service composition on top of a grid environment is that processes itself can be seen as services and therefore can be used within other processes again. This, in a way, adds recursive nature to processes and implements the well known composite pattern [14] for processes on the grid. Moreover, also the runtime support for process execution can be considered as a special, inherently distributed grid service.

In this paper, we introduce an integrated hyperdatabase and grid infrastructure that supports the processing of continuous data streams and that is able to distribute the processing of computationally expensive services within a grid. By this, the requirements of efficiently processing continuous data that can be found in digital medical library applications can be seamlessly supported.

The paper is structured as follows. Section 2 introduces a sample telemonitoring application to show the need for a joint hyperdatabase and grid environment. Section 3 gives a brief overview on the hyperdatabase and grid infrastructure. In Section 4, we present a process-based approach to data stream management. The dynamic process parallelization by using grid concepts is introduced in Section 5. Section 6 discusses related work and Section 7 concludes.

2 A Sample Application in a Digital Healthcare Library

In this section, we introduce a sample healthcare application to motivate the need for a flexible and reliable information management infrastructure that supports process management, data stream processing and management, and that provides grid computing capabilities.

The left hand side of Figure 1 illustrates a *telemonitoring system* which takes care of elderly patients suffering from chronic diseases (e.g., diabetes, heart diseases, or other age related problems like Alzheimer). This telemonitoring system is one of the informa-

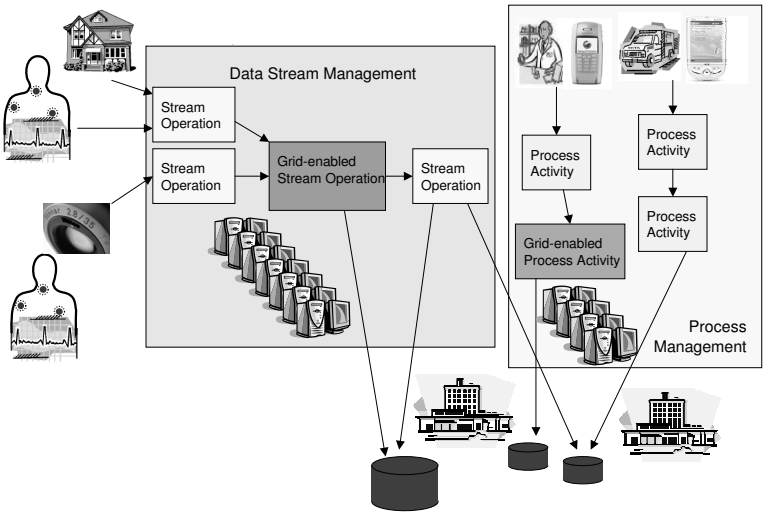


Fig. 1. Data Stream and Process Management in a Medical Digital Library

tion providers of the underlying medical digital libraries. Patients are equipped with an array of sensors, as for example the LifeShirt-System [30], that continuously measure the patient's body signals (e.g., ECG). Additionally, sensors integrated in the patient's home are detecting context information that describes what the patient is currently doing (e.g., if the patient is sleeping). This information is important to evaluate the medical meaning of vital signs — for example, the ECG signal has to be interpreted differently when a person is sleeping, compared to the case where he is active. In addition to medical monitoring, context information is also used to integrate a patient support system in this scenario. Patients can be remembered to turn off the oven or take their pills. In order to make use of the vast amount of sensor information, the incoming sensory data has to be processed in real-time. Medically relevant results may be stored in a digital library containing the patient's health record. Results with unknown characteristics are stored in repositories to support medical research. Critical results may request immediate intervention by the caregiver. In this case, appropriate processes (e.g., calling the emergency service or contacting a physician) have to be triggered.

Access to the contents of a medical digital library is supported by special services and user-defined processes that combine several of these services (illustrated on the right hand side of Figure 1). As described above, processes for contacting the caregiver (e.g., by sending a SMS to a mobile device of a physician), or even for triggering some rescue activities in case of critical situations have to be invoked if necessary. If the physician needs more detailed information or wants to request data on previous treatments or prescriptions, she has to be served with the data in a timely fashion. For all these purposes, appropriate processes have to be available (or have to be defined) and to be executed efficiently by the underlying infrastructure.

3 Architecture of Our Hyperdatabase and Grid Digital Library Infrastructure

Our infrastructure for telemonitoring applications is based on a combination of a hyperdatabase system [25] and a service grid environment as illustrated in Fig. 2. From hyperdatabases, we take the support for the definition and execution of processes on top of (web) services but also the possibility to implement continuously running processes for analyzing, processing, and managing data streams in real-time. Since processing data streams for evaluating the patient's health state requires the invocation of computationally intensive services, grid concepts are exploited to support the distributed computation on top of heterogenous resources. Therefore, the different data streams coming from the various sensors of a patient are dynamically distributed within the grid for parallel processing. Finally, the streams have to be joined in order to combine different sensor signals for rating medical relevance. The combination of process management and grid concepts allows for the composition of existing services and for the efficient distribution of single service invocations within the grid.

In the following, Chapter 4 introduces Data Stream Management within our hyperdatabase infrastructure OSIRIS-SE whereas Chapter 5 discusses how grid standards and dynamic service composition are incorporated into our integrated hyperdatabase and grid infrastructure.

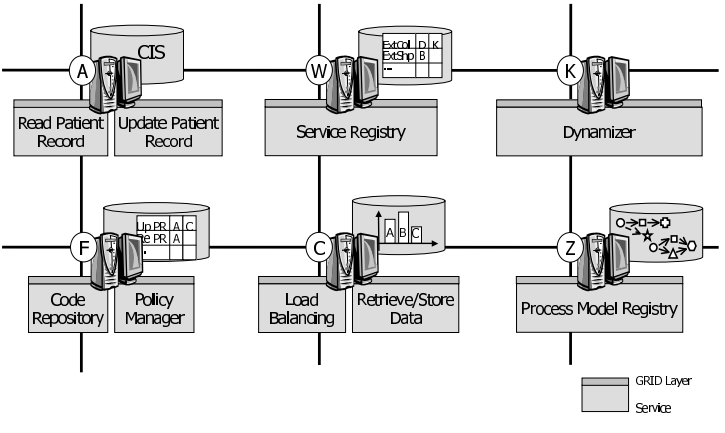


Fig. 2. Grid Infrastructure of a Medical Digital Library

4 Data Stream Management for Medical Digital Libraries

In this section, we introduce an extended hyperdatabase system for the support and management of continuous data streams.

4.1 Challenges in Data Stream Management

The main challenges in *data stream management* (DSM) are imposed by the large number of sensors, components, devices, information systems, and platforms connected by different network technologies, and by the vast amount of continuously generated data. For processing this data, existing systems and components are well in place and need to be incorporated into digital libraries. Reliability and provable correctness are new challenges that are of utmost importance particularly in healthcare applications, where failures may have perilous consequences. As described in Section 2, DSM has to interact with traditional process management in order to react to certain results (e.g., calling the ambulance) or to offer the user appropriate processes for the evaluation of DSM results. These challenges necessitate an infrastructure that combines the processing of data streams and process management, i.e., the possibility to combine services (conventional services as offered by digital libraries and services operating on data streams produced by sensors) and to execute composite services in a reliable way. Therefore, we propose an integrated information management infrastructure supporting user-defined processes, both conventional and processes performing DSM. *Hyperdatabase (HDB)* systems already provide an infrastructure for reliable process execution, which we have extended to enable DSM processes.

4.2 Peer-to-Peer Process Execution in the Hyperdatabase OSIRIS

A *hyperdatabase (HDB)* [22] is an infrastructure that supports the definition and reliable execution of user-defined processes on top of distributed components using existing services. Characteristic features of HDB's are the possibility to i.) support reliable peer-to-peer execution of processes without global control, thereby supporting a high degree of availability and scalability, ii.) add transactional guarantees to the execution of processes [24], and iii.) apply decentralized process execution in areas of intermitted connectivity.

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) is a prototype of a hyperdatabase, that has been developed at ETH Zurich and that is used as a starting point of our joint HDB and grid infrastructure. OSIRIS follows a novel architecture for distributed and decentralized process management. OSIRIS supports process execution in a peer-to-peer style based on locally replicated metadata, without contacting any central instance *(Peer-to-Peer Execution of Processes, P2PEP)*. With P2PEP, a component works off its part of a process and then directly migrates the instance data to nodes offering a suitable service for the next step(s) of the process according to its control flow specification. This is achieved by implementing two layers: the *HDB-layer*, a small software layer that is installed on each component providing a service and a set of global HDB repositories. These HDB repositories collect metadata on the processes to be executed, on the available components, and on their load. This meta information is smoothly distributed to the individual HDB layers – only metadata needed locally is actually replicated (e.g., only information on services and providers which might be invoked in some process are required at the local HDB layer of a component). More information on hyperdatabases and OSIRIS can be found in [22,23,25].

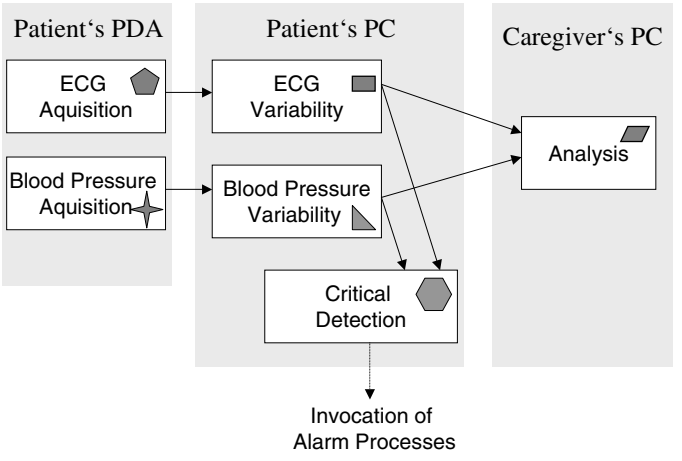


Fig. 3. Stream Process Processing ECG and Blood Pressure

4.3 OSIRIS-SE Infrastructure

HDB's have to be extended in order to enrich their benefits with the capabilities for DSM [5,6]. This extended infrastructure is called *OSIRIS-SE* (OSIRIS Stream Enabled). We consider *stream-processes*, which perform continuous processing of data streams. The requirements for the execution of these stream processes are similar to those of conventional processes with respect to important aspects like distributed execution, load balancing, meta information distribution, or fault tolerance. Figure 3 illustrates a stream-process, which continuously processes patient's ECG and blood pressure. Sensor signals are recorded and preprocessed by patient's PDA, which is wirelessly connected to patient's PC. The PC does further processing and detects critical health conditions. Processed sensor information is continuously forwarded to the caregiver for further analysis.

Operators are the processing units of DSM. Operators perform stream operations on incoming data streams and produce outgoing data streams. For this reason, operators have input and output *ports* for data streams. Sensors are the primary sources of data streams and can be considered as operators without incoming data streams. DSM is done by combining operators, similar to the combination of activities (service invocations) in traditional process management. A stream-process is such a well defined set of logically linked operators continuously processing the selected input data streams, thereby producing results and having *side effects*. Side effects are effects on external systems imposed by processing results (e.g., feeding a digital library with medical relevant information gained by the stream process). Stream-processes are defined by users with graphical design tools based on tools used in traditional process management. Our process design tool O'GRAPE (OSIRIS GRAphical Process Editor) [31] is extended to design also stream-processes. Additionally to the process activation flow, which describes the links between activity execution needed for controlling stream-processes,

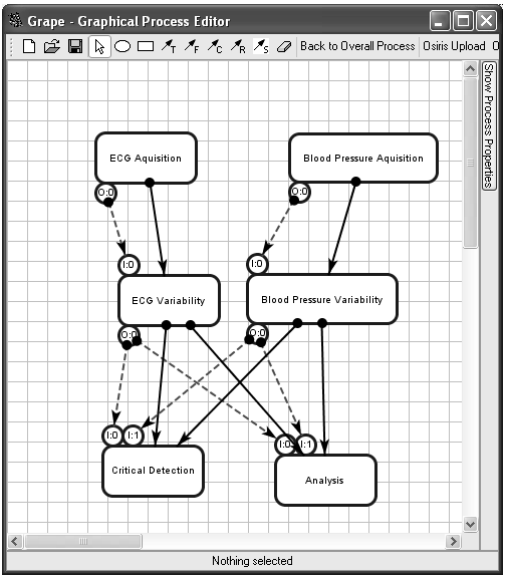


Fig. 4. Stream-Process Design with O'Grape

stream-processes have a second flow, called *data flow*. Whereas the process activation flow of a stream process describes how the operators are activated, the data flow describes how the data stream ports of the operators are interconnected. The extended O'GRAPE tool for stream processes can also explicitly graphically model this data flow, which does not necessarily comply with the process activation flow. In contrast to the process activation flow, the data flow combines ports of operators and not process activities. Figure 4 shows the stream-process of Figure 3 in O'GRAPE. Ports of operators are modeled as circles attached to the process activities. The solid edges indicate the process activation flow and the dashed edges indicate the data flow between ports.

Based on the OSIRIS approach to fault-tolerant distributed peer-to-peer process execution, we need to distribute necessary meta information on stream processes for DSM in the same way this is done also for process management.

Figure 5 illustrates the architecture of OSIRIS-SE. The process repository keeps definitions of stream processes and traditional processes. Locally needed pieces of the global process definition are published for replication on each corresponding node. Additionally, the service repository offers a list of available stream operators of components, which are also subject for smooth distribution among the suitable components offering the corresponding stream operators. A stream process is set up by sending an activation message to the HDB-layer of the component hosting a source operator (e.g., the component is attached to a sensor or has a data stream input). By making use of locally available metadata, the local HDB-Layer knows the subsequent stream operator and components in the process activation flow, which offer these operators and is able to make the routing decision. Then, the component sends an activation message to the selected subsequent component(s) and provides them with needed data streams.

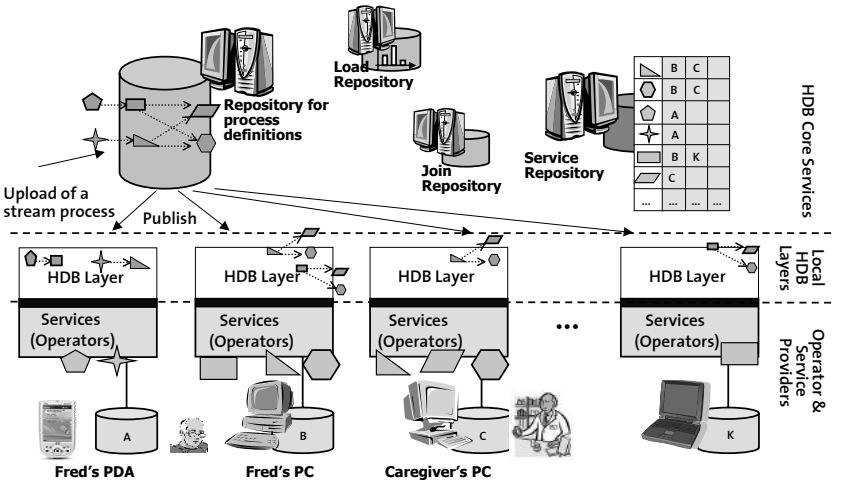


Fig. 5. Architecture of OSIRIS-SE

Special treatment is needed for join operators, which have more than one preceding operation (e.g., the analysis operator of Figure 3 which combines ECG variability with blood pressure variability of the patient). Due to the fact that preceding operations may run on different components, the infrastructure has to agree on one distinguished preceding operator. The component hosting the distinguished operator has unique responsibility for process routing, as needed for activation, load balancing, and failure handling. The routing decision is published as metadata via the join repository to non-distinguished components that are used in parallel.

OSIRIS-SE also allows for load balancing during the execution of stream processes. Therefore, the distribution of metadata on the load of components that are able to host stream operators needs to be published. This load information is used to choose the best component during the stream-process activation. In case of high load, the overloaded component is able to transfer a running stream operator instance to a component with less load. This is called *operator-migration*. When stream operations are affected that accumulate an internal state during their execution, this state has to be managed and transferred to the new host. Therefore, components make a backup of internal state of running stream-operators at a regular coordinated basis controlled by OSIRIS-SE. Information about the backup location address is metadata, which is also smoothly distributed via the operator backup repository.

The previous techniques are also responsible to allow for sophisticated failure handling. In case a component hosting a stream operator fails, components hosting preceding parts of the same stream process will recognize the failure because the transmission of their outgoing streams is no longer acknowledged. The infrastructure distinguishes between four failure cases:

1. The failed component recovers within a certain timeout (temporary failure). Then, processing is continued in the state before the failure. This is possible since output

queues of preceding components are used to buffer the data streams until they are acknowledged.
2. The failed component does not recover within the timeout period (permanent failure). In this case, the preceding component is in a similar situation as during the setup phase of the process. The component has to find suitable components that are able to perform subsequent stream operators. In addition to normal activation, operator migration is needed to initialize the newly generated operator instance with the existing operator backup. Due to local metadata, the new component is able to find the backup location and to load the old internal state for the continuation of stream processing. If the failed component recovers after the timeout, it has to be informed that its workload moved and that it is no longer in charge.
3. The failed component does not recover and there is no other suitable component. In this case, the stream process may have an alternative processing branch (defined in the streaming process), which is now activated by the preceding component.
4. There is no recovery and no possibility to continue stream processing. If so, a conventional process can be invoked to handle the failure situation (e.g., calling an administrator to fix the problem).

More on reliability of DSM with OSIRIS-SE can be found in [6]. OSIRIS-SE is capable of supporting telemonitoring applications by providing reliable integrated process and data stream management in peer-to-peer style. Furthermore, it allows to seamlessly cooperate with digital libraries, e.g., by making use of the services that are provided to access information.

5 Digital Libraries on the Grid

An important challenge when dealing with service composition, especially with computationally complex services, is the efficient routing of service requests among a set of providers. OGSA (Open Grid Services Architecture) [12] compliant grid systems are rapidly emerging and are widely accepted. These grid systems provide support for the efficient invocation and usage of individual services in the grid in a request/reply style. However, they do neither support service composition nor process execution. In contrast, the focus of state-of-the-art process support systems is not at all or only marginally oriented towards a tight integration into a grid environment. In what follows, we introduce an approach that combines (data stream) processes and (service) grid environments.

5.1 Bringing Service Composition to the Grid

Although OSIRIS, the starting point of our integrated *DSM* and *grid infrastructure*, is quite powerful in doing distributed process management, it does not yet follow OGSA or WS-RF [21], the de facto standard for grid environments. It does also not make use of the enhanced features offered in the globus toolkit [28] (the reference implementation of OGSA) like, for example, resource management and security. In our current work, we aim at bringing support for service composition to the grid, which is done by extracting some of the ideas that can be found in OSIRIS, and integrate those with current

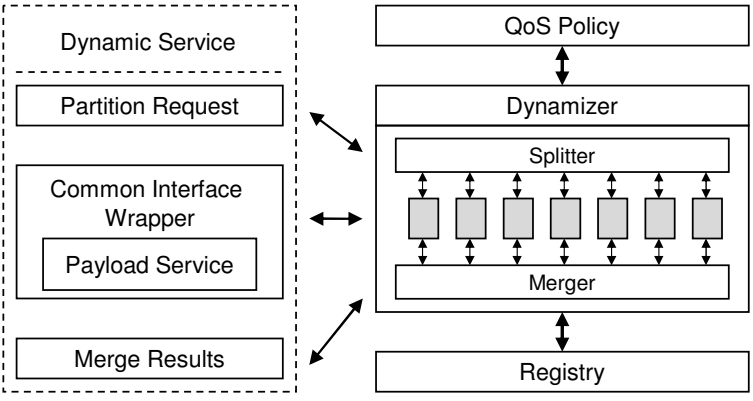


Fig. 6. Architectural Overview: Dynamic Parallelization of Grid-enabled Web Services

standards and services which have recently emerged in the grid community. This will result in a set of new OGSA compliant services enhancing current grid infrastructures with the ability of recursive process composition.

There are several possibilities to decompose an application into smaller parts that can then be executed in parallel. The most important ones are master/slave type of applications, as well as the divide and conquer or branch and bound paradigms. The applicability of these paradigms of course strongly depends on the semantics of the application to be parallelized. Especially the master/slave paradigm is very suitable to grid-enable applications [13], and is therefore widely used. In case of master/slave parallelization, the main prerequisites are:

- few or *no communication* among the sub parts
- work is dividable among *identical* sub parts
- work can be dis- and reassembled in a central point
- work can be parameterized and parallelized and does not need serial iterative processing.

Since the potential for master/slave parallelization can be found in several applications, we have started to apply this paradigm to enhance the efficiency, the creation, and the ease-of-use of services in the grid. Using the master/slave paradigm, application developers can focus on the implementation of the problem-specific subparts of the service as well as on the split into and merge of parallel subparts, but they do *not* need to take care about the distribution of subparts. This is particularly important since the latter requires dynamic information on the currently available resources which is not available at build-time, when the services are defined, as well as the way calls to these services are dynamically split and merged.

5.2 The Framework Architecture

To ease the creation of services for tomorrow's grid infrastructures, we developed a generic framework to handle master/slave applications where a single master process,

labeled 'Dynamizer' in Figure 6, controls the distribution of work to a set of identically operating slave processes. This framework is designed to accept ordinary web/grid services as destinations for calls, as well as composite services. The framework enables application developers to port new master/slave type of applications to the grid by enhancing a pre-existing web service with two additional services, one to split/partition an incoming request into parts for parallel execution, and one to merge and reintegrate sub-results to meet the original request. The service enhanced in such a kind is then named a 'Dynamic Service'.

The overall architecture is illustrated in Figure 6. The box in the center of the left hand side, labeled 'Payload Service', represents the actual service. It is responsible for providing application semantics, e.g., a complex computation or a database lookup. This is usually a piece of business logic that has existed beforehand, which is now supposed to be opened to the grid and enabled for parallel execution. To achieve this goal, it is surrounded by another box, labeled 'Common Interface Wrapper', which encapsulates the 'Payload Service' and enhances it with a common interface.

On top, 'Partition Request' encapsulates knowledge on how incoming parameters for the 'Payload Service' have to be partitioned, so that the original request can be decomposed into numerous new sub-requests. Each of these sub-requests can than be distributed on the grid, and be processed by other instances of the originally targeted service. The box at the bottom ('Merge Results') integrates (partial) results returned from the grid to provide the original service requester with a consolidated result. It can therefore be seen as the reverse operation to the 'Partition Request' service. The combination of these elements is referred to as 'Dynamic Service'.

To find the instances of the originally targeted service (e.g., services where the functional description equals the one of the 'Payload Service'), a registry is used (depicted in the lower right corner of Figure 6). This registry provides information on which services are available, how they can be accessed, and what their properties are (in terms of CPU load, connection bandwidth, access restrictions, etc).

5.3 Use of the Framework

The 'Dynamizer', depicted on the right hand side, makes use of the services mentioned above. It glues together the previously described services by making the parallel calls and by coordinating incoming results. It has also to provide appropriate failure handling. It is, in contrast to 'Partition Request' and 'Merge Results', application independent and generally usable. The 'Dynamizer' can interact with all services that adhere to a common interface, as ensured by the 'Common Interface Wrapper'. It can be integrated in environments able to call and orchestrate services, or it can be packaged and deployed together with specific services.

To make the best possible use of the 'Dynamizer', the user can send a description of the desired service quality along with the mandatory parameters needed to process the request. In this QoS (Quality of Service) policy, the user can, for example, describe whether the request should be optimized in terms of speed (select high performance nodes, and partition the input parameters accordingly), in terms of bandwidth (try to keep network usage low) or if it should aim for best accuracy (important for iterative approaches or database queries, where there is an option to use different data sources).

Since these specifications can be contradictory, adding preferences to rank the users requirements is important.

To better illustrate the functionality of the 'Dynamizer' regarding the user specified QoS policy, we reconsider the scenario from Section 2: A physician wants to use the digital library provided to gain some additional data on the patient she got sensor data from. In the QoS policy file, she specifies that she only wants to have data that can be viewed on her mobile device, a smart-phone, and as a second preference to have her call optimized in terms of speed since treatment for this patient is urgent. The 'Dynamizer' has only three services at hand, which are able to deliver data preprocessed for viewing it on mobile devices, all of them on rather slow computers, or alternatively access to 5 computationally powerful mainframes in the hospitals the patient was treated before, but without the capability to render their output for mobile devices. In this case, a choice can only be made if the user also specifies priorities for his preferences (e.g. consider first preference as more important than the second one). The algorithms needed to reconcile the user specifications, the details of the QoS description language and how to integrate this best with our existing implementation is currently under investigation.

The framework developed is based on web services. The core part consists of a set of classes building the 'Dynamizer' and the implementations of application specific slave services. These can be evolved to be OGSA-compliant grid services [12] bundled with corresponding stubs and some supporting classes for specialized exceptions and encapsulating the input and output parameters passed around. The work left to the application programmer is to implement the services 'Partition Request', 'Merge Results' and the 'Common Interface Wrapper' which are responsible for the application specific part. In addition, a Web Service deployment descriptor (WSDD) has to be written, as specified by the Axis framework [3], which GT3 is partly based on. At run-time, the framework determines which slaves to use, out of the set of all slaves registered to provide the appropriate service. This is done by accessing a global 'Registry' available in the grid. The request is then forwarded to all the slaves, after being divided into subtasks. This is shown in the upper right corner of Figure 7 where the service depicted as cross is provided by a set of slave services executing in parallel.

The current implementation can easily be adopted to more sophisticated distribution mechanisms based on the Service Data Elements (SDE's) [28] provided by each grid service. There might be more specialized implementations that distribute to slaves based on current workload, cost, or other metrics available. After having distributed the work, the 'Dynamizer' registers for notifications from the slaves and waits for results. After all slaves have returned, the 'Dynamizer' generates the final result by merging the results of the subparts and returns the completed result to the requestor. An important aspect here is to provide sophisticated failure handling that allows the 'Dynamizer' to re-distribute requests when slaves have failed during the execution of their subpart. On the slaves side, in addition to the implementation of the actual application logic, a deployment descriptor is needed that specifies where to register this particular slave service.

In the scenario described in Section 2, there is one dynamic service (bundled with a 'Dynamizer') which accepts streamed data from the patients life vest and ECG. This service acts, from the point of view of the process management system, as an ordinary step in the process chain. However, in the background, it re-directs the data stream to

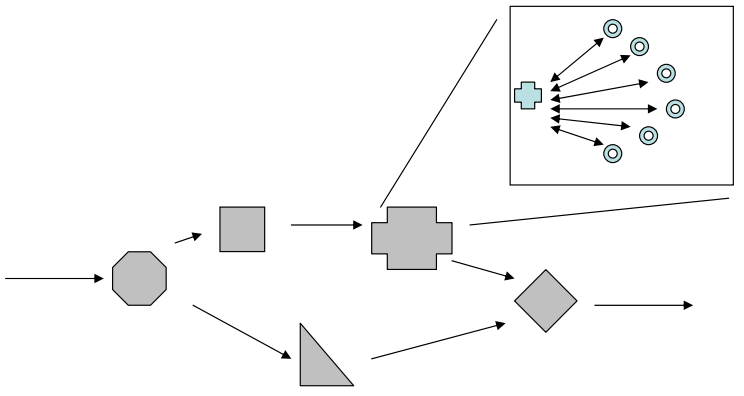


Fig. 7. Process containing a dynamically acting Grid Node

the slaves available in the system and checks the data against local replicas of digital libraries holding characteristic pathological and non-pathological data. The time intensive comparison of the stream data with entries in the digital library is done in a distributed way on the grid. The slaves report the result of their search back to the 'Dynamizer' who is then able to store the data for further usage and to trigger subsequent services or processes when needed (e.g., in critical situations).

5.4 From Master/Slave to Process Execution

A dynamic service can generally be seen as a grid service that controls the execution and dataflow among a set of services whose availability, number, and distribution is only known at runtime and subject to frequent changes. Since, from the point of view of the OSIRIS process execution engine, it acts just as any other operator or service, the dynamics of request distribution as well as the distribution pattern itself are transparent to the process execution engine. Figure 7 illustrates a process schema as executed by OSIRIS including a dynamically acting grid node. One step in this process, shown as a cross, is dispatching the request to various nodes in the grid and awaits their feedback. The process execution engine is not aware of this dispatching behind the scenes. This leads to the more general idea that the 'Dynamizer' can be seen as a process execution service itself, calling arbitrary grid services — either in parallel, sequentially, or in any other pattern available to the system.

These process execution services can be deployed to the grid as highly dynamic components. The distribution pattern of an algorithm can be determined at runtime based on some QoS information provided through the caller or can be hard-wired to a special distribution pattern. First results on performance and usage can be found in [32].

In order to avoid a centralized process execution service that could lead to a single source of failure, we are currently integrating the distributed process execution engine

described in OSIRIS. In OSIRIS, the execution plan for a process (determined by the control flow) is, prior to its invocation, split up into several execution steps. Each step consists of a service invocation, and information of all its successors. This allows to move the control from a centralized component to the responsibility of each node participating in the process. Therefore, this approach is much more robust to the failure of single nodes than centralized solutions.

6 Related Work

6.1 Data Stream Management

DSM aspects are addressed by various projects like NiagaraCQ [9], STREAM [4], and COUGAR [33]. The main focus of these projects is on query optimization and approximate query results and data provided by sensor networks. Aurora [7] allows for user-defined query processing by placing and connecting operators in a query plan. Aurora is a single node architecture, where a centralized scheduler determines which operator to run. Extensions like Aurora* and *Medusa* [10] also address DSM in distributed environments. TelegraphCQ [8] is a DSM project with special focus on adaptive query processing. Fjords allow for inter-module communication between an extensible set of operators enabling static and streaming data sources. Flux [26] provides load balancing and fault tolerance. PeerCQ [15] is a system that offers a decentralized peer-to-peer approach supporting continual queries running in a network of peers. The DFuse [19] framework supports distributed data fusion. Compared to other projects in this field, our integrated hyperdatabase and grid infrastructure offers two unique characteristics. Firstly, dynamic peer-to-peer process execution where local execution is possible without centralized control. Secondly, the combination of DSM and transactional process management enables sophisticated failure handling.

6.2 Grid Infrastructure

The master/slave paradigm is commonly agreed as valuable asset for the development of grid applications [13]. The master-worker tool [16] provides the possibility to integrate applications in the grid by implementing a small number of user-defined functions concentrating on the applications main purpose. It is applied to complex problems from the field of numerical optimization [2]. While it is tightly integrated into a former grid environment, the Globus Toolkit 2, our approach uses more recently emerged technologies and focuses on evolving into a more generally useable distributed process execution engine.

A similar approach is taken in AppLeS Master-Worker Application Template [27] where the main emphasis is on scheduling issues and a workflow model to select the best locations for the master and worker services. Other Approaches focusing on other task-parallel models can be found in [11,29] for the divide-and-conquer distribution pattern, and [18] for branch-and-bound.

In [20], BPEL4WS, the Business Process Execution Language for Web Services [1] is evaluated for the use within transactional business processes on the grid. The authors

point out that the usage of single, non-orchestrated web services is limited, and that there is a need for reliable and coordinated process execution on the grid.

7 Conclusion and Outlook

The proliferation of ubiquitous computing and the huge amount of existing information sources is leading towards a world where sophisticated information management is becoming a crucial requirement. A digital library for medical applications not only has to manage discrete data, it has also to support the acquisition, processing, and storage of streaming information that is continuously produced by sensors. Essentially, both streaming and non-streaming processes and applications have to be supported. Moreover, due to the complex processing operators that are used within stream processes, the distribution of work is a major requirement to efficiently process continuous data streams. By exploiting the features of a grid infrastructure, subparts can be executed in parallel by making use of the resources that are available at run-time. As a paradigm for the distribution of work within the grid, we have integrated a master/slave type of interaction into a stream-enabled HDB system.

Due to the distributed nature of this architecture, a special focus has to be set on failure handling and a high degree of reliability. There are numerous possibilities for the overall process to fail, not at last because of the missing control over the participating nodes within a grid infrastructure: single nodes can be disconnected without prior notification, wide area connections can be interrupted or significantly slowed down. Although some sophisticated transaction models have been integrated into OSIRIS [24], we continue to investigate mechanisms for a higher degree of reliability of the infrastructure that are oriented towards the special nature of grid infrastructures.

Based on this extended HDB system, we are currently building a comprehensive infrastructure that jointly addresses process-based service composition and streaming processes, and that is enriched by features from an existing grid infrastructure. In terms of the distribution paradigms supported, we are currently extending the master/slave type of distribution to allow for arbitrary execution plans. The goal is to define a generic, distributed and OGSA compliant process execution engine. This engine has to support different control flow specifications for composite services that are controlled by the grid-enabled process execution services so that it can be exploited for process-based applications on top of medical digital libraries.

References

1. Tony Andrews et al. *Business Process Execution Language for Web Services (BPEL4WS) 1.1.* BEA, IBM, Microsoft, SP, Siebel, 03 2003.
2. Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving Large Quadratic Assignment Problems on Computational Grids. In *Mathematical Programming 91(3)*, pages 563–588, 2002.
3. Apache WebServices Project. AXIS. http://ws.apache.org/axis/, 2004.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. of PODS Conf.*, pages 1–16, Madison, WI, USA, 2002.

5. G. Brettlecker, H. Schuldt, and R. Schatz. Hyperdatabases for Peer–to–Peer Data Stream Management. In *Proceedings of the International Conference on Web Services (ICWS'04)*, pages 358–367, San Diego, USA, July 2004.
6. G. Brettlecker, H. Schuldt, and H.-J. Schek. Towards Reliable Data Stream Processing with OSIRIS-SE. In *Proc. of the German Database Conf. (BTW'05)*, Karlsruhe, Germany, March 2005.
7. D. Carney, U. Centintemel, M. Cherniack, et al. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of VLDB Conf.*, pages 215–226, Hong Kong, China, 2002.
8. S. Chandrasekaran, O. Cooper, A. Deshpande, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of CIDR Conf.*, Asilomar, CA, USA, 2003.
9. J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. of SIGMOD Conf.*, pages 379–390, Dallas, TX, USA, 2000.
10. Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Ugur Cetintemel, Ying Xing, and Stan Zdonik. Scalable Distributed Stream Processing. In *Proc. of CIDR Conf.*, Asilomar, CA, USA, 2003.
11. I. Foster. Automatic Generation of Self - Scheduling Programs. In *IEEE Transactions on Parallel and Distributed Systems 2(1)*, pages 68–78, 1991.
12. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, January 2002.
13. Ian Foster and Carl Kesselman, editors. *The Grid 2, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2004.
14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
15. B. Gedik and L. Liu. PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proc. of Distributed Computing Systems Conf.*, pages 490–499, Providence, RI, USA, 2003.
16. Jean-Pierre Goux, Sanjeev Kulkarni, Jeff Linderoth, and Michael Yoder. An Enabling Framework for Master - Worker Applications on the Computational Grid. In *9th IEEE Int'l Symp. on High Performance Dist. Comp.*, pages 43–50, Los Alamitos, CA, 2000. IEE Computer Society Press.
17. R. Haux and C. Kulikovski. Digital Libraries and Medicine. *Yearbook of Medical Informatics*, 2001.
18. Adriana Iamnitchi and Ian Foster. A Problem-specific Fault-tolerance Mechanism for Asynchronous Distributed Systems. In *Int'l Conference on Parallel Processing*, 2000.
19. Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: A Framework for Distributed Data Fusion. In *Proc. of SensSys Conf.*, pages 114–125, Los Angeles, CA, USA, 2003.
20. F. Leymann and K. Güntzel. The Business Grid: Providing Transactional Business Processes via Grid Services. In *Proc. of ICSOC 2003*, pages 256 – 270, Trento, IT, 2003.
21. Daniel Sabbah. Bringing Grid & Web Services Together. Presentation, IBM Software Group, January 2004. http://www.globus.org/wsrf/sabbah_wsrf.pdf.
22. H.-J. Schek, K. Böhm, T. Grabs, U. Röhm, H. Schuldt, and R. Weber. Hyperdatabases. In *Proc. of WISE Conf.*, pages 28–40, Hong Kong, China, 2000.
23. H.-J. Schek, H. Schuldt, C. Schuler, and R. Weber. Infrastructure for Information Spaces. In *Proc. of ADBIS Conf.*, pages 22–36, Bratislava, Slovakia, 2002.
24. H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and Isolation for Transactional Processes. *ACM Transactions on Database Systems*, 27(1):63–116, 2002.
25. C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek. Peer-to-Peer Process Execution with OSIRIS. In *Proceedings of ICSOC 2003*, pages 483–498, Trento, Italy, December 2003. Springer LNCS, Vol. 2910.

26. M. Shah, J. Hellerstein, S. Chandrasekaran, and M. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In *Proc. of ICDE Conf.*, Bangalore, India, 2003.
27. G. Shao. *Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources*. PhD thesis, University of California - San Diego, 2001.
28. The Globus Alliance. The Globus Toolkit Version 3. http://www-unix.globus.org/toolkit/, 2003.
29. Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal. Efficient Load Balancing for Wide - Area Divide - And - Conquer Applications. In *8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 34–43, 2001.
30. VivoMetrics. VivoMetrics – Continuous Ambulatory Monitoring. http://www.vivometrics.com/site/system.html, 2003.
31. R. Weber, C. Schuler, P. Neukomm, H. Schuldt, and H.-J. Schek. Web Service Composition with OGrape and OSIRIS. In *Proc. of VLDB Conf.*, Berlin, Germany, 2003.
32. M. Wurz and H. Schuldt. Dynamic Parallelization of Grid-Enabled Web Services. In T. Priol A. Reinefeld M. Bubak P. Sloot, A. Hoekstra, editor, *Advances in Grid Computing - EGC 2005*, pages 173–184, Amsterdam, The Netherlands, June 2005.
33. Y. Yao and J. Gehrke. Query Processing for Sensor Networks. In *Proc. of CIDR Conf.*, Asilomar, CA, USA, 2003.

The MINERVA[1] Project: Towards Collaborative Search in Digital Libraries Using Peer-to-Peer Technology

Matthias Bender, Sebastian Michel, Christian Zimmer, and Gerhard Weikum

Max-Planck-Institut für Informatik,
D–66123 Saarbrücken, Germany
{mbender, smichel, czimmer, weikum}@mpi-inf.mpg.de

Abstract. We consider the problem of collaborative search across a large number of digital libraries and query routing strategies in a peer-to-peer (P2P) environment. Both digital libraries and users are equally viewed as peers and, thus, as part of the P2P network. Our system provides a versatile platform for a scalable search engine combining local index structures of autonomous peers with a global directory based on a distributed hash table (DHT) as an overlay network. Experiments with the MINERVA prototype testbed study the benefits and costs of P2P search for keyword queries.

1 Introduction

The peer-to-peer (P2P) approach, which has become popular in the context of file-sharing systems such as Gnutella or KaZaA, allows to handle huge amounts of data in a distributed way. In such a system, all peers are equal and all of the functionality is shared among all peers so that there is no single point of failure and the load is balanced across a large number of peers. These characteristics offer potential benefits for building a powerful search engine in terms of scalability, resilience to failures, and high dynamics. In addition, a P2P search engine can potentially benefit from the intellectual input of a large user community, for example, prior usage statistics, personal bookmarks, or implicit feedback derived from user logs and click streams.

Our framework combines well-studied search strategies with new aspects of P2P routing strategies. In our context of digital libraries, a peer can either be a library itself or a user that wants to benefit from the huge amount of data in the network. Each peer is a priori autonomous and has its own local search engine with a crawler and a corresponding local index. Peers share their local indexes (or specific fragments of local indexes) by posting the meta-information into the P2P network, thus effectively forming a large global, but completely decentralized directory. In our approach, this directory is maintained as a distributed hash table (DHT). A query posed by a user is first executed on the user's own peer, but can be forwarded to other peers for better result quality. Collaborative search strategies use the global directory to identify peers that are most likely to hold relevant results. The query is then forwarded to an appropriately selected

[1] Minerva is the Roman goddess of science, wisdom, and learning.

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 80–95, 2005.

subset of these peers, and the local results obtained from there are merged by the query initiator. We have implemented the outlined method in the MINERVA testbed for P2P search.

The rest of this paper is organized as follows: after presenting related work in section 2, we introduce the Chord overlay network as our chosen distributed hash table. Section 4 presents the main design fundamentals of our system, and Section 5 formalizes the system model. The implementation of our prototype is presented in Section 6, and experimental results are presented in Section 6. The last section informs about future and ongoing work and concludes.

2 Related Work

Recent research on P2P systems, such as Chord [27], CAN [24], Pastry [26], P2P-Net [5], or P-Grid [1], is based on various forms of distributed hash tables (DHTs) and supports mappings from keys, e.g., titles or authors, to locations in a decentralized manner such that routing scales well with the number of peers in the system.

Typically, an exact-match key lookup can be routed to the proper peer(s) in at most $O(log\ n)$ hops, and no peer needs to maintain more than $O(log\ n)$ routing information. These architectures can also cope well with failures and the high dynamics of a P2P system as peers join or leave the system at a high rate and in an unpredictable manner. Earlier work on scalable distributed storage structures, e.g., [18,31], addressed similar issues. However, in all these approaches searching is limited to exact-match queries on keys. This is insufficient for text queries that consist of a variable number of keywords, and it is absolutely inappropriate when queries should return a ranked result list of the most relevant approximate matches [8]. Our work makes use of one of these systems, namely Chord, for efficiently organizing a distributed global directory; our search engine is layered on top of this basic functionality.

PlanetP [12] is a publish-subscribe service for P2P communities and the first system supporting content ranking search. PlanetP distinguishes local indexes and a global directory to describe all peers and their shared information. The global directory is replicated using a gossiping algorithm. The system, however, is limited to a few thousand peers.

Odissea [28] assumes a two-layered search engine architecture with a global directory structure distributed over the nodes in the system. A single node holds the entire index for a particular text term (i.e., keyword or word stem). Query execution uses a distributed version of Fagin's threshold algorithm [13]. The system appears to cause high network traffic when posting document metadata into the network, and the query execution method presented currently seems limited to queries with one or two keywords only.

The system outlined in [25] uses a fully distributed inverted text index, in which every participant is responsible for a specific subset of terms and manages the respective index structures. Particular emphasis is put on three techniques to minimize the bandwidth used during multi-keyword searches: Bloom filters [4], caching, and incremental result gathering. Bloom filters are a compact representation of membership in a set, eliminating the need to send entire index lists across servers. Caching reduces the

frequency of exchanging Bloom filters between servers. Incremental result gathering allows search operations to halt after finding a certain number of results.

[20] considers content-based retrieval in hybrid P2P networks where a peer can either be a simple node or a directory node. Directory nodes serve as super-peers, which may possibly limit the scalability and self-organization of the overall system. The peer selection for forwarding queries is based on the Kullback-Leibler divergence between peer-specific statistical models of term distributions. The approach that we propose in this paper also uses such statistical measures but applies them in a much more light-weight manner for better scalability, primarily using bookmarks rather than full index information and building on a completely decentralized directory for meta-information.

Strategies for P2P request routing beyond simple key lookups but without considerations on ranked retrieval have been discussed in [33,10,9], but are not directly applicable to our setting. The construction of semantic overlay networks is addressed in [19,11] using clustering and classification techniques; these techniques would be orthogonal to our approach. [29] distributes a global directory onto peers using LSI dimensions and the CAN distributed hash table. In this approach peers give up their autonomy and must collaborate for queries whose dimensions are spread across different peers.

In addition to this recent work on P2P Web search, prior research on distributed IR and metasearch engines is potentially relevant, too. [6] gives an overview of algorithms for distributed IR like result merging and database content discovery. [14] presents a formal decision model for database selection in networked IR. [23] investigates different quality measures for database selection. [15,21] study scalability issues for a distributed term index. GlOSS [16] and CORI [7] are the most prominent distributed IR systems, but neither of them aimed at very-large-scale, highly dynamic, self-organizing P2P environments (which were not an issue at the time these systems were developed).

A good overview of metasearch techniques is given by [22]. [32] discusses specific strategies to determine potentially useful local search engines for a given user query. Notwithstanding the relevance of this prior work, collaborative P2P search is substantially more challenging than metasearch or distributed IR over a small federation of sources such as digital libraries, as these approaches mediate only a small and rather static set of underlying engines, as opposed to the high dynamics of a P2P system.

3 Chord - A Scalable P2P Lookup Service

The efficient location of nodes in a P2P architecture is a fundamental problem that has been tackled from various directions. Early (but nevertheless popular) systems like Gnutella or KaZaA rely on unstructured architectures in which a peer forwards messages to all known neighbors. Typically, these messages include a Time-to-live (TTL) tag that is decreased whenever the message is forwarded to another peer. Even though studies show that this *message flooding* (or *gossiping*) works remarkably well in most cases, there are no guarantees that all relevant nodes will eventually be reached. Additionally, the fact that numerous unnecessary messages are sent interferes with our goal of a highly scalable architecture.

Chord [27] is a distributed lookup protocol that addresses this problem. It provides the functionality of a distributed hash table (DHT) by supporting the following $lookup$

operation: given a key, it maps the key onto a node. For this purpose, Chord uses consistent hashing [17]. Consistent hashing tends to balance load, since each node receives roughly the same number of keys. Moreover, this load balancing works even in the presence of a dynamically changing hash range, i.e., when nodes fail or leave the system or when new nodes join.

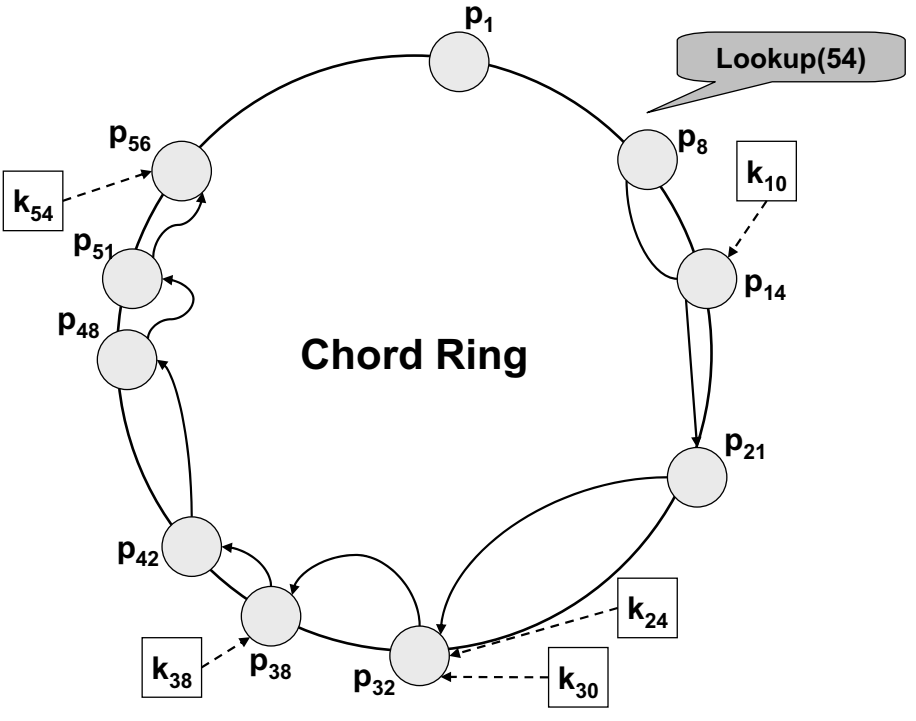


Fig. 1. Chord Architecture

Chord not only guarantees to find the node responsible for a given key, but also can do this very efficiently: in an N-node steady-state system, each node maintains information about only $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. These properties offer the potential for efficient large-scale systems.

The intuitive concept behind Chord is as follows: all nodes p_i and all keys k_i are mapped onto the same cyclic ID space. In the following, we use keys and peer numbers as if the hash function had already been applied, but we do not explicitly show the hash function for simpler presentation. Every key k_i is now assigned to its closest successor p_i in the ID space, i.e. every node is responsible for all keys with identifiers between the ID of its predecessor node and its own ID.

For example, consider Figure 1. Ten nodes are distributed across the ID space. Key k_{54}, for example, is assigned to node p_{56} as its closest successor node. A naive approach

of locating the peer responsible for the key is also illustrated: since every peer knows how to contact its current successor on the ID circle, a query for a key k_{54} initiated by peer p_8 is passed around the circle until it encounters a pair of nodes that straddle the desired identifier; the second in the pair (p_{56}) is the node that is responsible for the key. This lookup process closely resembles searching a linear list and has an expected number of hops of $O(N)$ to find a target node, while only requiring $O(1)$ information about other nodes.

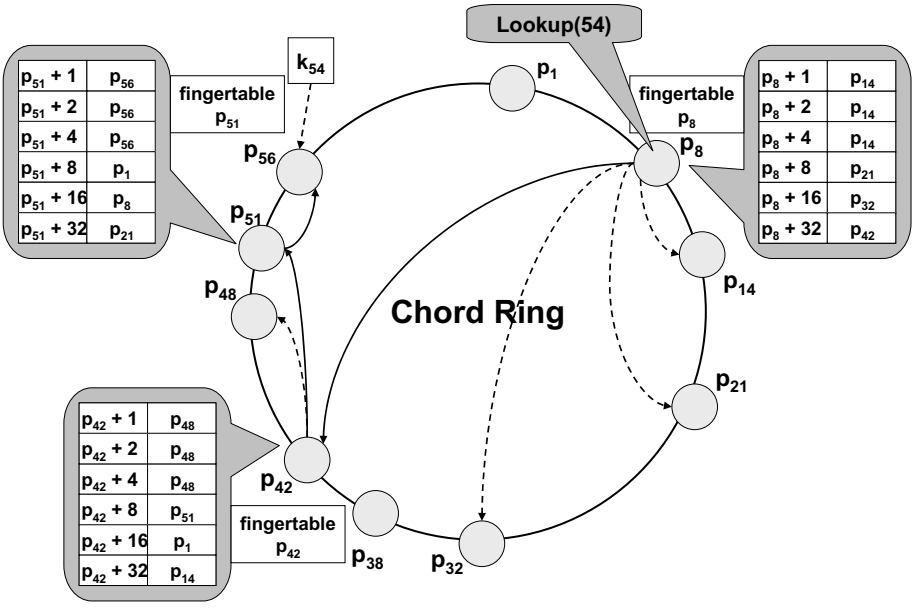


Fig. 2. Scalabe Lookups Using Finger Tables

To accelerate lookups, Chord maintains additional routing information: each peer p_i maintains a routing table called *finger table*. The m-th entry in the table of node p_i contains a pointer to the first node p_j that succeeds p_i by at least 2^{m-1} on the identifier circle. This scheme has two important characteristics. First, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. Secondly, a node's finger table does not necessarily contain enough information to *directly* determine the node responsible for an arbitrary key k_i. However, since each peer has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between itself and the target node. This property is illustrated in Figure 2 for node p_8. It follows that the number of nodes to be contacted (and thus the number of messages to be sent) to find a target node in an N-node system is $O(\log N)$.

Chord implements a stabilization protocol that each peers runs periodically in the background and which updates Chord's finger tables and successor pointers in order to ensure that lookups execute correctly as the set of participating peers changes. But even with routing information becoming stale, system performance degrades gracefully. Chord can also guarantee correct lookups if only one piece of information per node is correct [27].

Chord can provide lookup services for various applications, such as distributed file systems or cooperative mirroring. However, Chord by itself is not a search engine, as it only supports single-term exact-match queries and does not support any form of ranking.

4 Design Fundamentals

Figure 3 illustrates our new approach which closely follows a publish-subscribe paradigm. We view every library as autonomous. Peers, i.e. libraries acting as peers, can post meta-information at their discretion. Our conceptually global but physically distributed directory does not hold information about individual documents previously crawled by the peers, but only very compact aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose to other peers. We use a distributed hash table (DHT) to partition the term space, such that every peer is responsible for a randomized subset of terms within the global directory. For failure resilience and availability, the entry for a term may be replicated across multiple peers.

Every peer publishes a summary (*Post*) for every term in its local index to the underlying overlay network. A Post is routed to the peer currently responsible for the Post's term. This peer maintains a *PeerList* of all postings for this term from across the network. Posts contain contact information about the peer who posted this summary together with local IR-style statistics (e.g., TF and IDF values [8]) for a term and other quality-of-service measures (e.g., length of the index list for a given term, or average response time for remote queries).

Users wishing to pose a query are equally modeled as peers. Their potential input to the global directory consists of local bookmarks that conceptually represent high-authority documents within the overall document space.

The querying process for a multi-term query proceeds as follows: First, the querying peer retrieves a list of potentially useful libraries by issuing a *PeerList request* for each query term to the global directory. Next, a number of promising libraries for the complete query is selected from these PeerLists (e.g., based on the quality-of-service measures associated with the Posts). Subsequently, the query is forwarded to these carefully selected libraries and executed based on the their local indexes. Note that this communication is done in a pairwise point-to-point manner between the peers, allowing for efficient communication and limiting the load on the global directory. Finally, the results from the various libraries are combined at the querying peer into a single result list.

We have chosen this approach for the following reasons:

- The goal of finding high-quality search results with respect to precision and recall cannot easily be reconciled with the design goal of unlimited scalability, as the

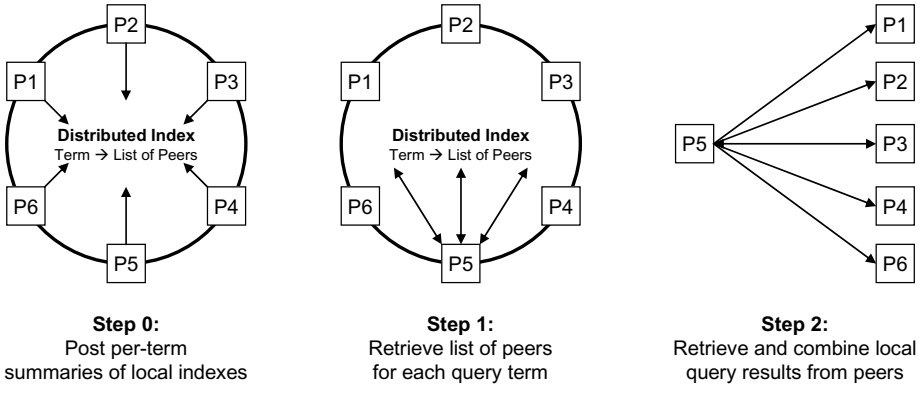


Fig. 3. P2P Query Routing

best information retrieval techniques for query execution rely on large amounts of document metadata. In contrast, posting only aggregated information about local indexes and executing queries at carefully selected peers exploits extensive local indexes for good query results while, at the same time, limiting the size of the global directory and, thus, consuming only little network bandwidth.

– If each peer were to post metadata about each and every document it has crawled, the amount of data moved across the network and, thus, the amount of data held by the distributed directory would increase drastically as more and more peers enter the network. In contrast, our design allows each peer to publish merely a concise summary per term representing its local index. As new peers enter the network, we expect this approach to scale very well as more and more peers jointly maintain this moderately growing global directory.

This approach can easily be extended in a way that multiple distributed directories are created to store information beyond local index summaries, such as information about user bookmarks or relevance assessments derived from peer-specific query logs, click streams, or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

5 System Model

In this section we formalize the design that we have previously presented. Let $P := \{p_i | 1 \leq i \leq r\}$ be the set of peers currently attached to the system. Let $D := \{d_i | 1 \leq i \leq n\}$ be the global set of all documents; let $T := \{t_i | 1 \leq i \leq m\}$ analogously be the set of all terms.

Each peer $p_i \in P$ has one or more of the following local data available:

– Local index lists for terms in $T_i \subseteq T$ (usually $|T_i| \ll |T|$).
The local index lists cover all terms in the set of locally seen documents $D_i \subseteq D$ (usually $|D_i| \ll |D|$).

- Bookmarks $B_i \subseteq D_i$ ($|B_i| \ll |D|$)
  Bookmarks are intellectually selected links to selected documents or other peer profile information and, thus, are a valuable source for high-quality search results as well as for the thematic classification of peers.
- Cached documents $C_i \subseteq D$
  Cached documents are readily available from a peer.

Separate hash functions can be used in order to build conceptually global, but physically distributed directories that are well-balanced across the peers in the ID space ($hash_{terms} : T \rightarrow ID$, $hash_{bm} : D \rightarrow ID$, and $hash_{cached} : D \rightarrow ID$).

Given hash functions that assign identifiers to keys using $id_{k,j} := hash_j(k)$ with $j \in \{terms, bm, ...\}$, the underlying distributed hash table offers a function $lookup : ID \rightarrow P$ that returns the peer p currently responsible for an id.

Building on top of this basic functionality, different PeerList requests plr_j can be defined as functions $plr_{terms} : T \times P \rightarrow 2^P$, $plr_{bm} : D \times P \rightarrow 2^P$, and $plr_{cache} : D \times P \rightarrow 2^P$ that, from a peer p previously determinded using $lookup$, return lists of peers that have posted information about a key id in dimension j. Note that $id_{k,j}$ for a specific key k is unambiguously defined across the directory using $hash_j$(k).

In order to form a distributed directory, each peer p_i at its own discretion globally posts subsets $T_i' \subseteq T_i$, $B_i' \subseteq B_i$, and $C_i' \subseteq C_i \subseteq D$ (potentially along with further information or local QoS statistics) forming the corresponding global directories:

- $systerms : T \rightarrow 2^P$ with $systerms(t) = plr_{terms}(t, lookup(hash_{terms}(t)))$
  This directory provides a mapping from terms to PeerLists and can be used to identify candidate peers that hold index information about a specific term.
- $sysbm : D \rightarrow 2^P$ with $sysbm(d) = plr_{bm}(d, lookup(hash_{bm}(d)))$
  This function provides information about which peers have bookmarked specific documents and is a combination of the above methods analogously to $systerms$.
- $syscd : D \rightarrow 2^P$ with $syscd(d) = plr_{cached}(d, lookup(hash_{cached}(d)))$
  This function provides information about the document availabilities in the caches of local peers, which is a valuable information for the efficient gathering of results.

We consider a query q as a set of $(term, weight)$-pairs and the set of available queries as $Q := 2^{T \times \mathbb{R}}$. In order to process a query q, first a candidate set of peers that are confronted with the query has to be determined. This can be done using the functions $select_{terms} : Q \rightarrow 2^P$, $select_{bm} : 2^D \rightarrow 2^P$, and $select_{cached} : 2^D \rightarrow 2^P$ that select candidate subsets for each dimension by appropriately combining the results returned by $systerms$, $sysbm$, and $syscd$, respectively. These candidate subsets are combined (e.g., by intersection or union) using a function $comb : 2^P \times 2^P \times 2^P \rightarrow 2^P$.

Combining the above, the final candidate set is computed using a function

$$selection : Q \times 2^D \times 2^D \rightarrow 2^P$$

$$selection(q, B_0'', C_0'') := comb(select_{terms}(q), select_{bm}(B_0''), select_{cached}(C_0''))$$

where $B_0'' \subseteq B_0$ and $C_0'' \subseteq C_0$ are the bookmarks and cached documents, respectively, that the querying peer has chosen to support query execution. For example, a

peer may choose its own bookmarks and a sample of its cached documents as B_0'' and C_0'', respectively.

The execution of a query is a function $exec : 2^P \times Q \rightarrow 2^D$ that combines the local results returned by the peers that are involved in the query execution into one single final result set. Finally, we can define the global query execution function $result$: $Q \times 2^D \times 2^D \rightarrow 2^D$ that is evaluated as

$$result(q, B_0'', C_0'') := exec(selection(q, B_0'', C_0''), q)$$

$$= exec(comb(select_{terms}(q), select_{bm}(B_0''), select_{cached}(C_0'')), q)$$

6 Implementation

Figure 4 illustrates the architecture of a single library peer of the MINERVA prototype system. Each peer works on top of our globally distributed directory which is organized as a distributed hash table (DHT) that provides a mapping from terms to peers by returning a *PeerDescriptor* object representing the peer currently responsible for a term. A *Communicator* can be established to send messages to other peers. Every peer has an *Event Handler* that receives incoming messages and forwards them to the appropriate local components.

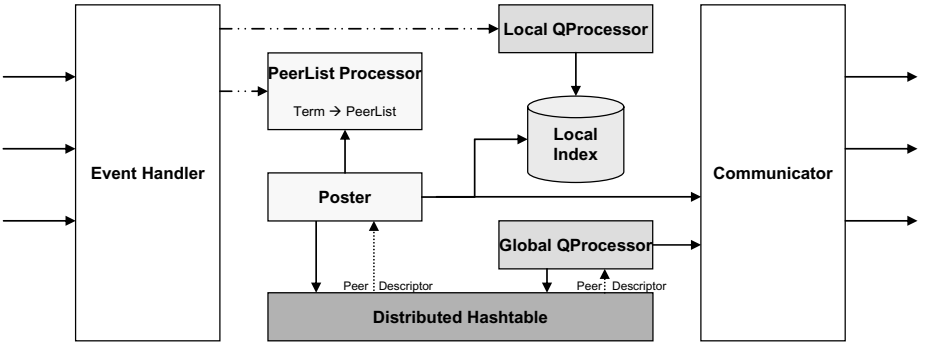


Fig. 4. System Architecture

Every peer has its own local index that can be imported from external crawlers and indexers. The index is used by the *Local QueryProcessor* component to answer queries locally and by the *Poster* component to publish per-term summaries (*Posts)* to the global directory. To do so, the Poster uses the underlying DHT to find the peer currently responsible for a term; the *PeerList Processor* at this peer maintains a PeerList of all Posts for this term from across the network. When the user poses a query, the *Global QueryProcessor* component analogously uses the DHT to find the peer responsible for each query term and retrieves the respective PeerLists from the PeerList Processors using Communicator components. After appropriately processing these lists, the Global

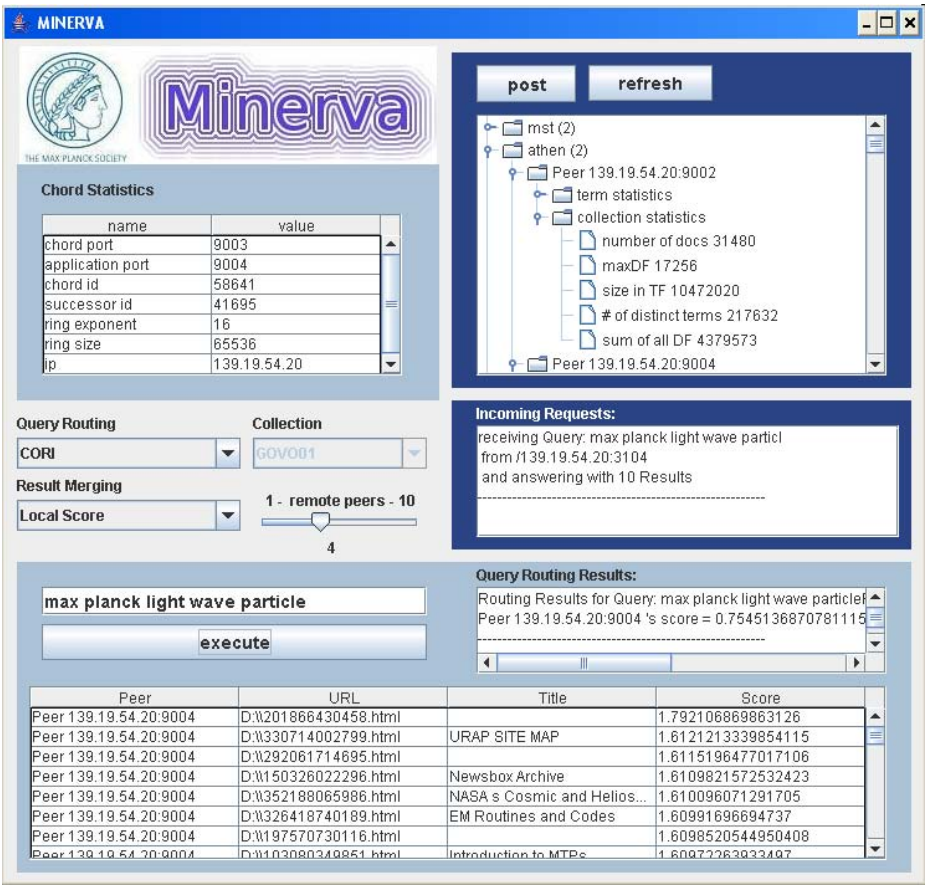


Fig. 5. Prototype GUI of MINERVA

QueryProcessor forwards the complete query to selected peers, which in turn process the query using their Local QueryProcessors and return their results. Finally, the Global QueryProcessor merges these results and presents them to the user.

We have built a prototype system that handles the above procedures. Our system uses a Java-based reimplementation of Chord [27] as its underlying DHT, but can easily be used with other DHT's providing a $lookup(key)$ method. Communication is conducted socket-based, but Web-Service-based [2] peers can easily be included to support an arbitrarily heterogeneous environment. The local index is stored in a database. It consists of a collection of standard IR measures, such as TF and IDF values. Result ranking is based on a smoothed TF*IDF quality measure. Figure 5 shows a screenshot of the user interface of our prototype. The user creates a peer by either creating a new Chord ring or by joining an existing system. Both actions require the specification of a local Chord port for communication concerning the global directory and a local application port for direct peer-to-peer communication. The join operation requires additional information on how to find an already existing peer. Status information regarding the

Chord ring is displayed. The Posts section provides information about the terms that a peer is currently responsible for, i.e., for which it has received Posts from other peers. The button *Post* posts the information contained in the local index to the DHT. The Queries section can be used to execute queries. Similar to Google, multiple keywords can be entered into a form field. After query execution, the results obtained from the system are displayed.

7 Experiments

This section discusses experimental results. We divide the evaluation part into 3 subsections: the experimental setup, performance results and results about query result quality.

7.1 Experimental Setup

One pivotal issue when designing our experiments was the absence of a standard benchmark. While existing a number of benchmark collections for (centralized) Web search, it is not clear how to apply these collections to our scenario. While other studies partition these collections into smaller, disjunctive pieces we do not think this is an adequate approach. In contrast, we expect a certain degree of overlap among the collections, with popular documents being indexed by a substantial fraction of all peers, but, at the same time, with a large number of documents only indexed by a tiny fraction of all peers.

Our group has performed extensive Web crawls to gather real-world experimental data. In the absence of a standard benchmark for our scenario we test MINERVA with collections that have been created by Web crawls originating from manually selected crawl seeds on the topics Sports, Computer Science, Entertainment, and Life, leading to 10 thematically focused collections. Additionally, one reference collection was created by combining all collections and eliminating duplicates. Table 1 gives details about the collections. Note the overlap between the 10 original collections.

Table 1. Collection Statistics

Collection	# Docs	Size (MB)
Computer Science	10459	137
Life	12400	144
Entertainment	11878	134
Sport	12536	190
Computer Science mixed	11493	223
Computer Science mixed	13703	239
CS Google	7453	695
Sport Google	33856	1,086
Life Google	16874	809
Entertainment Google	18301	687
Σ	168953	4,348
Combined Collection	142206	3,808

Table 2. Queries (*: *not* from WordTracker)

Max Planck Light Wave Particle*	Einstein Relativity Theory*
Lauren Bacall	Nasa Genesis
Hainan Island	Carmen Electra
National Weather Service	Web Search*
John Kerry	George Bush Iraq*

For the query workload we took the 6 most popular queries on AltaVista, as reported by http://www.wordtracker.com for September 21, 2004, and 4 additional queries that were specifically suitable for our corpus. Table 2 lists all queries.

The query is executed on the reference collection using state-of-the-art top-k techniques [30] to obtain a reference query result (*ideal document ranking*). A CORI-style [6] peer selection approach is used to rank the 10 collections according to their expected result quality; the quality of this approach and the importance of a powerful peer selection strategy in general has been assessed in previous work [3]. Subsequently, the query is sent to an increasing number of collections according to the ranking previously determined. These local results are returned to the query initiator and merged into one global result list (*obtained document ranking*) which is compared to the ideal document ranking to obtain recall statistics.

The experimental results are highly influenced by the following parameters:

- Number of documents retrieved from each collection
- Number of documents retrieved from combined collection (size of ideal document ranking)

For our experiments, we retrieve 30 documents from the reference collection and 30 documents from each peer. The *obtained document ranking* is again limited to a size of 30 documents to match the size of the ideal document ranking.

All experiments have been conducted using 10 Peers running as separate processes on a single notebook with a Pentium M 1.6GHz processor and 1GB main memory. All peers share a common Oracle $10g$ database that is installed on a Dual-Pentium Xeon 3GHz processor with $4GB$ main memory. The peers are connected to the database through a 100MBit network.

7.2 Performance

In our experiments we observed the following performance characteristics. The overall amount of data that one peer has to send across the network during a complete posting process turned out to be about 650KB for a collection containing 45000 distinct terms. Note that we use a standard compression technique (gzip) to reduce the size of the messages. We expect each peer to receive a share of Posts of similar size from the network (note that each peer spreads its Posts across n peers, i.e., sending $\frac{|T|}{n}$ Posts to every remote peer). Even if we assume collections containing more distinct terms, this leads to a very reasonable space requirement at each participating peer.

One PeerList request accounts for one message of about 150 bytes and an answer of about 1000 bytes (for a list of 10 promising peers together with their IR statistics, which is well above the number of peers suggested in Section 7 required to expect good recall). Note that such a PeerList request is necessary for each keyword of a query. Sending a query to remote peers accounts for a message of approximately 250 bytes for a typical two-keyword query (including term weights), which is sent to a limited number of peers (e.g., 2 or 3) directly, i.e., not stressing all other peers or the directory. Each of these remote peers answers with a limited-size list of local results together with some light-weight statistics about the results, which in our experiments triggered response messages of typically 2500 bytes for 30 results. Note that we currently do not use all statistics included in these messages, i.e., the sizes could easily be reduced even further.

Keeping in mind the high computational load during the experiments conducted on a single notebook, the complete process of identifying promising peers, sending the query and merging the results is executed in a reasonable time of about 5 seconds per queried peer, which is mainly the time needed to execute the query locally at this peer; the system overhead is negligible. Thus, in our experimental setup, the execution time for one query increases linearly with the number of queried remote peers. In a real-world scenario, the execution time is expected to remain nearly constant as the load is spread over a number of independent remote processors. However, this selfish observation is a naive assessment and makes our strong case that identifying a small number of promising peers for a query is a key issue of making P2P Web search feasible.

7.3 Query Result Quality

Using the above set of collections and queries, we studied the recall relative to the top-30 documents from the reference collection when we query the peers in descending

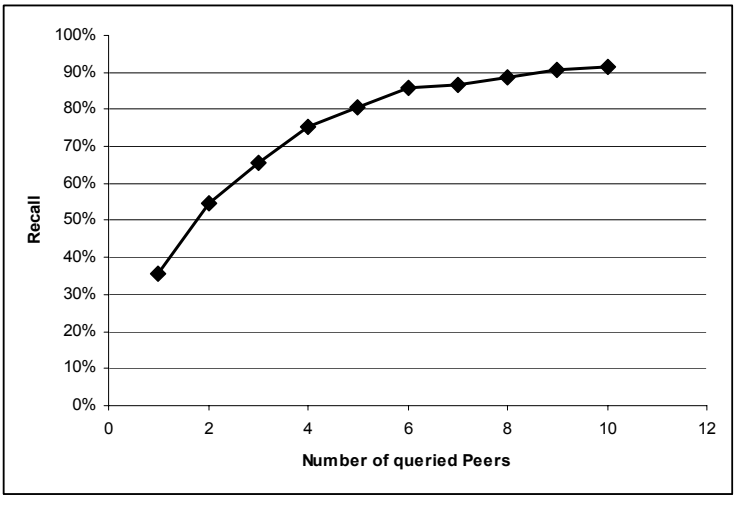


Fig. 6. Recall

order of their positions in the peer ranking. Figure 6 shows the recall of our approach in percent as the number of queried peers increases. Sending the query to the two best remote peer only already yielded an average of almost 60% of all relevant documents, whereas the inferior peers typically do not contribute many new documents to the obtained query result. Note that this does not mean they do not *contain* any relevant documents, but rather that their relevant documents have already been contributed by other peers before. So taking into account the overlap of the peers' local contents is a crucial issue for future work.

8 Ongoing and Future Work

Our prototype implementation of MINERVA allows the easy exchange of strategies for query routing (i.e., selecting the peers to which the query is sent), and for merging the results returned by different peers. We are currently analyzing different strategies and are preparing more extensive comparative experiments. We want to contact as few peers as possible to retrieve the best possible results, i.e., we want to estimate a $benefit/cost$ ratio when deciding on whether to contact a specific peer. While a typical cost measure could be based on expected response time (network latency, current load of remote peer), meaningful benefit measures seem harder to find. Possible measures could follow the intuition that good answers are expected to come from peers that are similar to the query, but at the same time have only little overlap with our local index and, thus, can potentially contribute new results. We also take a closer look at existing strategies for combining local query results from metasearch engine research and try to fit those with our P2P environment.

We are also investigating the trade-offs of not storing *all* Posts for a term, but only the top-k posts (based on some quality measure) to reduce space consumption of the global directory. While this seems intuitive at first sight (good results should come from good peers), early experiments indicate that this strategy might be dangerous for multi-term queries, as good combined results are not necessarily top results for any one of the search terms.

Due to the dynamics typical for P2P systems, Posts stored in the PeerLists become invalid (peers may no longer be accessible, or the responsibility for a specific term may have moved to another peer). A possible mechanism to handle these problems is to assign a TTL (Time-to-live) stamp to every Post in the list. Every peer periodically revalidates its Posts. Stale Posts will eventually be removed from the PeerList. We address the question of choosing a good time period for refreshing the Posts and compare this strategy to a strategy of actively moving Posts to other peers as responsibilities change.

References

1. Karl Aberer, Magdalena Punceva, Manfred Hauswirth, and Roman Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1):58–67, 2002.
2. Gustavo Alonso, Fabio Casati, and Harumi Kuno. *Web Services - Concepts, Architectures and Applications*. Springer, Berlin;Heidelberg;New York, 2004.

3. Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. The minerva project: Database selection in the context of p2p search. *BTW*, 2005.
4. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
5. Erik Buchmann and Klemens Böhm. How to Run Experiments with Large Peer-to-Peer Data Structures. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA*, April 2004.
6. J. Callan. Distributed information retrieval. *Advances in information retrieval, Kluwer Academic Publishers.*, pages 127–150, 2000.
7. James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28. ACM Press, 1995.
8. Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, 2002.
9. Edith Cohen, Amos Fiat, and Haim Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of the IEEE INFOCOM'03 Conference, April 2003*, April 2003.
10. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.
11. Arturo Crespo and Hector Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, October 2002.
12. Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Rutgers University, September 2002.
13. Ronald Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
14. N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.
15. Torsten Grabs, Klemens Böhm, and Hans-Jörg Schek. Powerdb-ir: information retrieval on top of a database cluster. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 411–418. ACM Press, 2001.
16. Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.
17. David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
18. Witold Litwin, Marie-Anna Neimat, and Donovan A. Schneider. Lh* – a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, 1996.
19. Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003 (DBISP2P 03)*, pages 33–47.
20. Jie Lu and Jamie Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 199–206. ACM Press, 2003.
21. Sergey Melnik, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. Building a distributed full-text index for the web. *ACM Trans. Inf. Syst.*, 19(3):217–241, 2001.
22. Weiyi Meng, Clement T. Yu, and King-Lup Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

23. Henrik Nottelmann and Norbert Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 290–297. ACM Press, 2003.
24. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172. ACM Press, 2001.
25. Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.
26. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
27. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
28. T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasunderam. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. Technical report, Polytechnic Univ., 2003.
29. Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.
30. Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. *VLDB*, pages 648–659, 2004.
31. Radek Vingralek, Yuri Breitbart, and Gerhard Weikum. Snowball: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.
32. Zonghuan Wu, Weiyi Meng, Clement T. Yu, and Zhuogang Li. Towards a highly-scalable and effective metasearch engine. In *World Wide Web*, pages 386–395, 2001.
33. Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE Computer Society, 2002.

Distribution Alternatives for Superimposed Information Services in Digital Libraries

Sudarshan Murthy, David Maier, and Lois Delcambre

Department of Computer Science, Portland State University,
PO Box 751 Portland, OR 97207-0751 USA
{smurthy, maier, lmd}@cs.pdx.edu
http://datalab.cs.pdx.edu/sparce

Abstract. Component-based, service-oriented digital library architectures are being used to provide superimposed information services such as annotations. Although much attention has been paid to the issues in building components for these services, not enough attention has been paid to their deployment—specifically to distribution. We believe that matching the location of executable and data components to the needs of patrons and digital libraries can improve the overall system performance. We describe five distribution alternatives for providing superimposed information services in a digital library and discuss the trade-offs for each alternative. We define some metrics to compare the performance of the alternatives, and use the metrics in a qualitative evaluation of the alternatives. We also discuss potential barriers for performance and means of improving performance. We use our middleware architecture for superimposed information management, called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)*, for illustration.

1 Introduction

Our research on superimposed information focuses on allowing users to *superimpose* [6] new information such as annotations and summaries on top of existing *base* information such as web pages and PDF documents. In addition to superimposing annotations, a user may select parts of existing information and create new linkages among those selections. For example, a user may create an alternative organization of sections in a PDF document, or the user may create a table of selected contents.

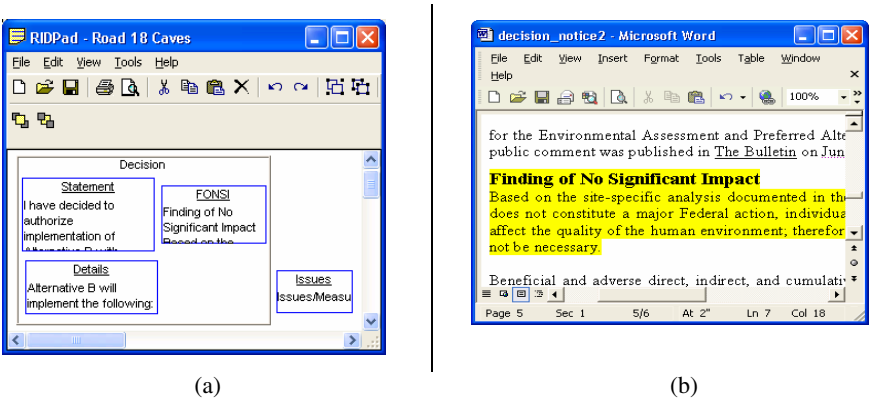
Figure 1 (a) shows superimposed information elements that were created and organized in one of our superimposed applications called *RIDPad* [8]. It shows four items labeled 'Statement', 'FONSI', 'Details' and 'Issues,' each linked to a selection in other documents such as spreadsheets and word processor documents. For example, the item labeled 'Issues' is linked to a selection in an MS Excel spreadsheet; the item labeled 'FONSI' is linked to a selection in a MS Word document (shown in Figure 1 (b)). The box labeled 'Decision' groups items. RIDPad works with a component-based, middleware architecture called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)* [8].

Some digital Library (DL) systems also use component-based architectures [13, 17] to support creation of annotations and metadata over information in a DL. Some

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation …, LNCS 3664, pp. 96 – 111, 2005.

of the benefits of component-based architectures are that architectural components may be replaced with alternatives and new components may be plugged in easily. They make it easier to build new services using component stacks.

Another benefit of component-based architectures, one that does not receive as much attention, is flexibility of deployment. With proper interface design and abstraction, components (both data and executable) may be either centrally-deployed or distributed, without affecting the services they provide. This flexibility is important because placing a component at the right location can improve performance, especially for frequently used services.



(a) (b)

Fig. 1. (a) Superimposed information organized using *RIDPad*, a superimposed application. (b) The MS Word selection linked to the item FONSI of the RIDPad document activated.

In this paper, we present five distribution alternatives and their trade-offs when providing superimposed information services, such as annotations, in a DL. We use our component-based middleware architecture SPARCE to illustrate the alternatives. We present the alternatives without a specific DL architecture in mind because they should apply to component-based DL architectures in general.

To motivate, we present a hypothetical annotation system in Figure 2 (fashioned after Open Digital Library [11]) and outline two distribution alternatives for it. This system has three components: user interface, annotation, and an archive. One distribution alternative, call it Alternative A, is to run the user interface and the annotation components on a patron's (client's) computer, and run the archive component within a DL server. Alternative B is to run the user interface on a patron's computer, and run the other two components within a DL server. These alternatives could differ significantly in maintainability and performance. For example, Alternative B may be more maintainable because few components run outside the DL server, but it has the potential to increase the load on the server. The alternatives differ also in the interface a DL server needs to provide to the outside world.

Alternative A requires a DL server to provide interface to the Archive component (the Annotate component connects from the patron's computer to the Archive component in the DL server), whereas Alternative B requires the server to provide interface to the Annotate component.

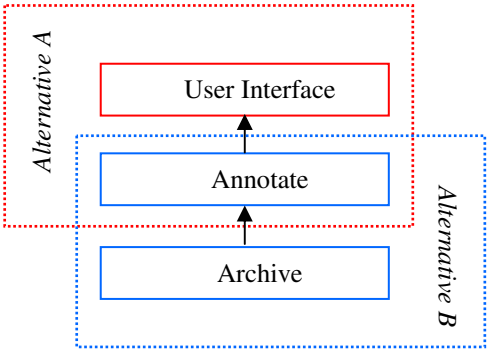


Fig. 2. An ODL-style annotation system. Each dotted rectangle contains components collocated in the distribution alternative called out.

Proximity of components can affect the overall system performance. To give an idea, we present results from a simple experiment we conducted. Table 1 shows the mean round-trip time for three web-service methods based on SOAP [15] bound to HTTP. From the table we see that the mean round-trip time over a WAN is about 60 times that over a LAN when sending and receiving 400 bytes. These numbers tell us that performance could be improved by placing components with a higher number of round trips between them closer to each other (based on network distance, not geographic distance). For example, if we know that the Annotate component makes many round trips to the Archive component to serve a single request from the user interface, we may benefit by collocating the Annotate and Archive components in a DL server as in Alternative B.

The numbers in the last column of Table 1 deserve some clarification, because they indicate rather large round-trip times. A WAN communication is heavily influenced by network conditions (for example, congestion), policies, and operations such as routing and filtering (for example, firewalls). Table 1 shows numbers for communication between a computer in a home office (client) and a computer inside OGI (server). In this setting, all communication passes through the policies and infrastructure of at least four parties: the client, the client's ISP, OGI, and us (the web service provider). Additionally, all packets are subject to inspection at each hop. Finally, one web service roundtrip might require more than one roundtrip at lower network layers.

The rest of this paper is organized as follows. Sections 2 and 3 give an overview of superimposed information and SPARCE, respectively. Section 4 defines some metrics, and details five distribution alternatives and their trade-offs. Section 5 discusses some issues in employing those distribution alternatives. Section 6 provides a brief overview of related work. Section 7 concludes the paper.

Table 1. Mean round-trip time for SOAP-based web service methods via HTTP. Columns *Input* and *Output* denote the number and type of inputs and outputs respectively for the methods; Column *Local* shows round-trip time when client and server are on the same computer, *LAN* shows round-trip time when client and server are connected over a LAN (100 MBPS, one hop), *WAN* shows round-trip time when the client and server are connected over a WAN (756 KBPS DSL connection, more than 18 hops).

		Mean round-trip time (milliseconds)		
Input	Output	Local	LAN	WAN
None	None	2.94	3.13	146.74
10 integers	10 integers	3.13	3.36	137.53
One 400-byte array	One 400-byte array	7.87	10.53	689.39

2 Superimposed Information

Superimposed information refers to data placed over existing information sources to help select, access, organize, connect, and reuse information elements in those sources [6]. Existing information sources reside in the *base layer*, and data placed over one or more base sources resides in the *superimposed layer* (see Figure 3). Word-processor documents, databases, and web pages are examples of base documents. A stand-off annotation is an example of superimposed information (because it is stored separately from the object of annotation, and it maintains a link to the object). An application that manipulates base information is called a *base application*; an application that manipulates superimposed information is called a *superimposed application*. Adobe Acrobat is a base application for PDF documents, and RIDPad (Figure 1 (a)) is a superimposed application.

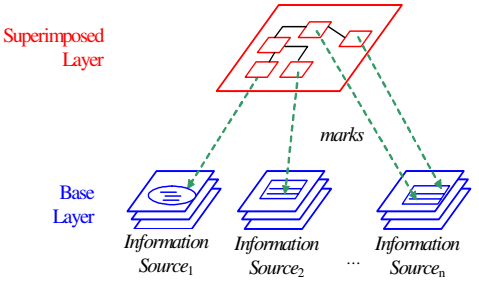


Fig. 3. Layers of information in a superimposed information management system. A *mark* connects a superimposed information element to the base layer.

2.1 Marks

A superimposed information element (such as an annotation) can reference a base information element (such as a selection in a spreadsheet) using an abstraction called

a *mark*. Figure 3 shows the superimposed layer using marks to address base elements. We have implemented the mark abstraction for base types such as PDF, HTML, and MS Word. The addressing scheme a mark implementation uses often depends on the base type(s) it supports. For example, an implementation for PDF documents may use page number and index of the first and last words in a text selection, whereas an implementation for XML documents may use XPath. All mark implementations provide a common interface to address base information, regardless of the base types or access protocols they support. A superimposed application can work uniformly with any base type by virtue of this common interface. For example, the items in RIDPad document of Figure 1 (a) refer to marks in MS Word and MS Excel documents, but the RIDPad application works uniformly with all marks. Figure 1 (b) shows the MS Word mark of the 'FONSI' item activated.

2.2 Excerpts and Contexts

Superimposed applications may sometimes need to incorporate the content of base-layer elements in the superimposed layer. For example, an application might use the extracted base-layer content as the label of a superimposed element. We call the contents of a base-layer element an *excerpt*. An excerpt can be of various types. For example, it may be text or an image. An excerpt of one type might also be transformed into other types. For example, formatted text in a word processor could also be seen as plain text, or as a graphical image.

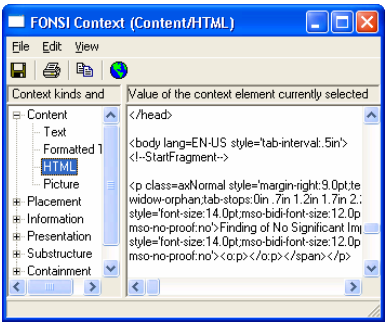


Fig. 4. The context of the MS Word mark of the item FONSI of the example RIDPad document in the Context Browser. The browser is showing part of the HTML markup required to display the excerpt for the mark formatted as is in the base layer (the highlighted region of Figure 1 (b)).

In addition to excerpts, superimposed applications may use other information related to base-layer elements. For example, an application may group superimposed information by the section in which the base-layer elements reside. To do so, the application needs to retrieve the section heading (assuming one exists) of each base-layer element. We call information concerning a base-layer element, retrieved from

the base layer, its *context*. Presentation information such as font name and location information such as line number might be included in the context of a mark. Each such piece of information is a *context element*, and context is a collection of context elements. Because we use the same mechanism to support both contexts and excerpts, we often use the term "context" broadly to refer to both kinds of information about a base-layer element. Figure 4 shows the context of a MS Word mark in a browser (for the mark activated in Figure 1 (b)).

3 SPARCE

The *Superimposed Pluggable Architecture for Contexts and Excerpts* (*SPARCE*) is a middleware-based approach for mark and context management [8]. It is designed to be extensible in terms of supporting new base-layer types and context-element types, without adversely affecting existing superimposed applications. Figure 5 shows the SPARCE reference model. The Mark Manager provides operations such as creating marks and storing them in a marks repository that it maintains. The Context Manager retrieves context information for a mark. Superimposed applications use the managers to create marks and access context (which in turn use base applications).

The Mark Manager supports three operations for marks: creation, retrieval, and activation. *Mark creation* is the operation of generating a new mark corresponding to a selection in a base layer. This operation consists of three steps: generating the address of base information (and other auxiliary information), using the information generated to create a mark object, and storing the mark object in the mark repository. Details of each mark, such as the address of the base document and the selection inside it, are stored as an XML file. The mark repository is a database of such XML files.

The *mark retrieval* operation returns a mark from the mark repository. *Mark activation* is the operation of navigating to a location inside the base layer, using the information supplied by a mark.

SPARCE uses mediators called *context agents* to retrieve context information for a mark from the base layer. A context agent interacts with a base application to retrieve context. The name of the context agent to use for each mark instance is one of the details stored in the mark repository. SPARCE uses this information to instantiate an appropriate context agent for a mark instance. A superimposed application receives a reference to the instance of context agent from SPARCE, and then works directly with the agent instance to retrieve context.

The components of SPARCE (see Figure 5) may be mapped to those of the ODL-style annotation system we introduced in Figure 2. Superimposed applications (patron applications) provide the user interface. These could be desktop applications or browser-based applications (for example, Java applets) running on a patron's computer. The Mark Manager, the Context Manager, and the base applications constitute the Annotate component. The base documents and a DL's interface to access them (if required) roughly constitute the Archive component.

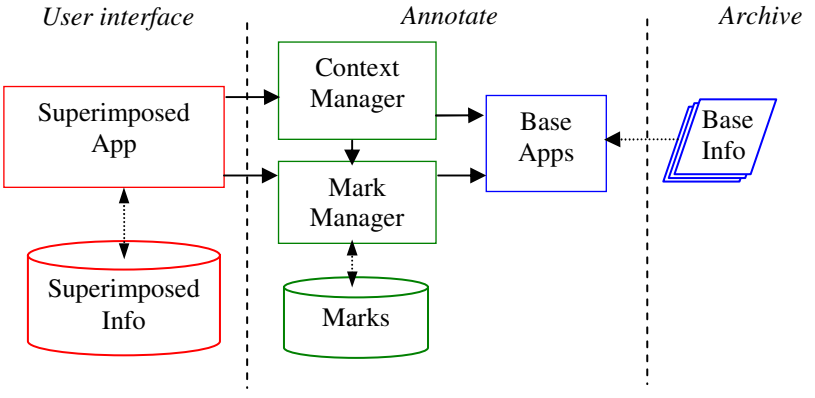


Fig. 5. The SPARCE reference model. Solid arrows show dependency, dotted arrows show data flow (not all data-flows shown). The dashed lines partition components to correspond to the blocks in Figure 2.

4 Distribution Alternatives

All components of SPARCE shown in Figure 5 are candidates for distribution, but we consider only four components: the patron (superimposed) application, the Mark Manager, the Context Manager, and base applications. We vary the location of these components and describe five distribution alternatives for SPARCE (see Figures 7, 8, and 9) when providing superimposed information services in a DL. For simplicity, we assume the following configuration for all alternatives:

- The DL server contains base documents (and that an appropriate interface in the DL server is used to access the documents).
- The two manager modules (the Mark Manager and the Context Manager) run on the same computer.
- The mark repository is stored wherever the Mark Manager is deployed.
- The patron application always runs on a patron's computer.
- The superimposed information is stored on a patron's computer.

We first present a goal for distribution and some related metrics. We do not provide experimental results based on these metrics, but estimate trends based on a few rules of thumb (see Section 4.7). We assume a patron uses a high-speed Internet connection (such as broadband) to a DL server. We also assume that the ratios of the mean round-trip times shown in the third row (for 400-byte array input and output) of Table 1 hold.

A note on terminology: the term "patron's computer" in the rest of this paper may mean a stand-alone computer or a computer in a network local to the patron. Our description of alternatives would be valid for either interpretation of the term.

4.1 Goals and Metrics

Several metrics such as latency (for example, time to serve a request), load (for example, the number of active processes) and throughput (for example, the number of requests processed per unit time) should be considered for a thorough analysis of the alternatives. However, we discuss only latency in detail. Section 5 touches on issues such as load.

In distributing the architectural components, one of our goals is to minimize the latency of a patron application's request (T_{pm})[1]. This latency is the duration between the patron initiating a request in the patron application (to a manger module) and the patron receiving a corresponding response. It is made up of the following components (see Figure 6):

- t_{bb}: The time taken by a base application to complete a requested operation. An example is the time to retrieve a context element's value.
- t_{mb}: The round-trip time between a manager module and a base application. This time measures the duration between a manager module receiving a request from a patron application and the manger module returning a corresponding response, after discounting the time the base application takes to complete its work.
- t_{pm}: The round-trip time between a patron application and a manager module. This time measures the duration between the patron initiating a request in the patron application (to a manger module) and the patron receiving a corresponding response, after discounting the time the manager module and the base application need to complete their work.

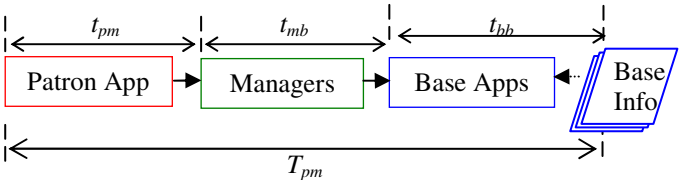


Fig. 6. Components of latency of a patron application's request. The subscripts *p*, *m*, and *b* stand for patron application, manager module, and base application respectively. They identify the pair of architectural components with which a latency term is associated.

Because the latency of a patron application's request (T_{pm}) is the sum of the times t_{pm}, t_{mb}, and t_{bb}, our sub-goals are to minimize these terms. The distribution alternatives we describe vary the location of architectural components to highlight the affect of each alternative on these latency terms.

4.2 Distribution Alternative A

Alternative A is to run the patron applications, the manager modules, and the base applications on a patron's computer (Figure 7). That is, a DL server only supplies base

[1] This is the only goal we consider in this paper.

information. The patron opens base documents using appropriate base applications running on his or her computer. Marks and superimposed information are stored on the patron's computer. Because all components run within the patron's computer, the round-trip times between them would be quite low. Base documents would still be accessed over the network because they reside within the DL server. This cost is likely incurred only once per document per session, because documents accessed over the Internet are usually (automatically) first downloaded to the client's computer. Base applications then access the documents locally. This alternative requires that each patron have all base applications locally, even for rarely encountered base types.

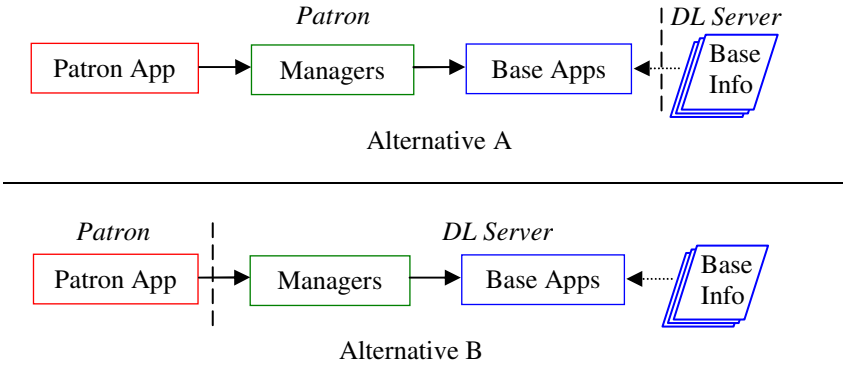


Fig. 7. Distribution Alternatives A and B. The dashed lines denote network boundaries.

4.3 Distribution Alternative B

Alternative B is to run the patron applications on a patron's computer, and run the manager modules and the base applications within a DL server (Figure 7). Marks are stored in the DL server, and superimposed information is stored on the patron's computer. Because the base applications operate within the DL server, the patron is able to view base documents in their native applications only if those applications are also available locally on his or her computer. However, because the manager modules run inside the DL server, the mark activation operation would be unable to exploit any base application available on the patron's computer. When the patron activates a mark (for example, the MS Word mark as in Figure 1 (b)), the DL server prepares the context elements needed to display the excerpt and sends it to the patron application (possibly in HTML as in Figure 4). The patron may request additional context elements to view as needed.

Because the manager modules and the base applications run within a DL server, the round-trip time between them would be low, but the round-trip time between a patron application and the manager modules (t_{pm}) would be high. Consequently, patron applications must strive to minimize the number of round-trips to the manager modules. For example, combining requests for context elements can reduce the number of round trips.

4.4 Distribution Alternative C

Alternative C is to run the patron applications and the two manager modules on a patron's computer, but run the base applications within a DL server (Figure 8). As in Alternative A, marks and superimposed information are stored on the patron's computer. Like Alternative B, the patron would be able to view base documents in their native applications only if those applications are also available locally. Unlike Alternative B, because the manager modules run on the patron's computer, the mark activation operation would be able to exploit any base application available on the patron's computer. In other cases, the Context Manager module would have to retrieve the necessary context elements to provide a view of marked regions. Because the manager modules run on the patron's computer, but the base applications run within the DL server, the round-trip time between them (t_{mb}) would be high.

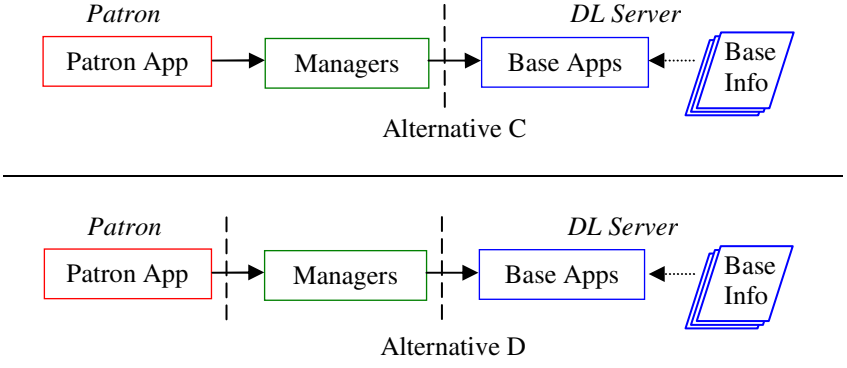


Fig. 8. Distribution Alternatives C and D. The dashed lines denote network boundaries.

4.5 Distribution Alternative D

Alternative D is to run the patron applications on a patron's computer, the two manager modules in a *middle tier*, and run the base applications within a DL server (Figure 8). As in Alternative A, the superimposed information is stored on the patron's computer, but marks are stored in the middle tier. The capabilities of a patron application are similar to those in Alternative B. The performance of this alternative is also similar to that of Alternative B, except the round-trip time t_{mb} would be higher because the manager modules run in the middle tier. Finally, with the manager modules running in a middle tier, they can connect to more than one DL server.

4.6 Distribution Alternative E

Alternative E is similar to Alternative D except that the two manager modules *and* the base applications run in a middle tier (Figure 9). The performance of this alternative is

also similar to that of Alternative D, except that the round-trip time t_{mb} would be lower because the manager modules and the base applications run in the same tier. The base documents are accessed over the network because they reside within the DL server, but the base applications are in a middle tier. That is, this alternative is similar to Alternative A with respect to base applications accessing base documents. However, the cost of accessing base documents could be less than that in Alternative A, if the bandwidth between the middle-tier and the DL server is better than that between the patron's computer and the DL server.

As with Alternative D, the manager modules can connect to more than one DL server. Further, the same installation of base applications may also work with information on more than one DL server.

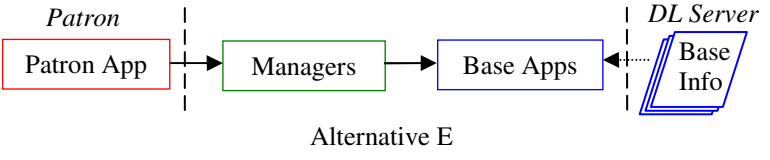


Fig. 9. Distribution Alternative E. The dashed lines denote network boundaries.

4.7 Summary of Distribution Alternatives

Table 2 provides a summary of the location of components, the role of the DL server, and the profile of the components on the patron's computer for each alternative. A DL server operating as an *information server* is similar to a file server, whereas an *application server* runs applications on behalf of clients. A *thin* client profile means a minimal amount of code runs on the patron's computer. Patron applications tend to be browser-based (applets for example) in this case. A *fat* client profile means large amounts of code run on the patron's computer. Patron applications tend to be desktop applications in this case, and are often richer in functionality than browser-based applications.

Table 3 summarizes the *trend* we expect for maintenance cost and performance of the resulting systems from the alternatives. Two rules of thumb drive our expectation of maintenance cost. First, a thin client is less expensive to maintain than a fat client because fewer components run outside a DL server in a thin client scenario. For components that run outside the DL server, changes made to a component need to be propagated to all locations where that component is deployed.

Second, the load on a DL server increases as the number of components running within the server increases. The trend *Medium* for Alternatives C and D indicates that the load on the DL server would be greater than that for Alternative A, but less than that for Alternative B. (The manager modules run outside the DL server in Alternatives C and D.)

Table 2. Summary of alternatives. *Client profile* is the profile of the components on the patron's computer.

	Location of components				
Alternative	Patron apps	Managers	Base apps	DL Server Role	Client profile
A	Patron	Patron	Patron	Info server	Fat
B	Patron	DL	DL	App server	Thin
C	Patron	Patron	DL	App server	Fat
D	Patron	Middle tier	DL	App server	Thin
E	Patron	Middle tier	Middle tier	Info server	Thin

Two rules of thumb guide our expectation of round-trip times: 1) placing components "closer" to each other reduces the round-trip time between them; 2) the reduction is greater if the number of round-trips between them is large (especially when the components exchange large amounts of data).

Based on these rules of thumb alone, Table 3 indicates that Alternative E has the best potential performance. The high round-trip time between a patron application and the manager modules (t_{pm}) appears to be its only weakness.

Table 3. Summary of maintenance cost and performance. Columns t_{pm} and t_{mb} are round-trip times as defined in Section 4.1.

	Maintenance Cost				Round-trip time	
Alternative	Patron apps	Managers	Base apps	DL Server Load	t_{pm}	t_{mb}
A	High	High	High	Low	Low	Low
B	Low	Low	Low	High	High	Low
C	High	High	Low	Medium	Low	High
D	Low	Low	Low	Medium	High	High
E	Low	Low	Low	Low	High	Low

5 Discussion

In reality, DL systems are likely to employ a mixture of distribution alternatives. For example, some DL providers might wish to support patron applications of different capabilities (for example, fat clients and thin clients). Doing so requires the DL server to provide many kinds of interfaces (expose manager modules *and* base applications) for patron applications to choose from. Also, a patron may work with more than one DL, and those DLs may employ different distribution alternatives. In this case, patron applications must be able to discover the alternative a DL system employs.

Needs of patrons and patron applications, and access patterns also influence the choice of a distribution alternative. For example, Alternatives B and D are similar to each other except that the latter uses a middle tier. Though Alternative B causes a greater load on the DL server (see Table 3), its round-trip time between the manager modules and base applications (t_{mb}) is lower. If an analysis of the access pattern shows that the number of interactions between the manager modules and base applications is low, then the round-trip time t_{mb} might not be critical. Also, Alternative B could be

less expensive than Alternative D because the middle-tier would be eliminated from the configuration. Unfortunately, choosing an alternative is rarely this easy, because the needs of patrons and patron applications vary (from time to time, patron to patron, and application to application). Fortunately, we can exploit some patterns to build flexibility in to the system. For example, Alternatives B, D, and E have the common pattern that the manager modules are not resident on the patron's computer. That is, patron applications cross network boundaries to connect to the manager modules in any of these alternatives, and the location of components beyond the manager modules should not matter to patron applications. Consequently, one could start with any of these three alternatives and migrate to another alternative as needs and access patterns change. The switching could even be transparent to patrons and patron applications.

The presence of a middle tier gives Alternatives D and E some advantages over the other alternatives. The infrastructure of the middle tier can be designed with DL services in mind. Also, there are likely to be far fewer middle-tier servers than patron computers. These alternatives also create the possibility for a federation of DLs to maintain a middle tier (or tiers), thereby distributing some of the cost of operations.

Distribution alternatives that deploy the manager modules and base applications outside a patron's computer could also be helpful in serving requests from patron applications running on diverse operating systems. For example, SPARCE is currently implemented on the MS Windows® operating systems. This implementation could potentially be accessed using a web service to retrieve context for use in a patron application running on a Macintosh or UNIX operating system. (We are currently evaluating such a system of providing superimposed information services.)

Hosting base applications outside a patron's computer may cause some problems. Without the necessary base application available locally, a patron will be unable to see a marked region in its original context (as in Figure 1 (b)). In such cases, the context manager would have to retrieve the context elements necessary to provide a "broad enough" view of the selection, but the combined size of the context elements could be excessively large. Alternatively, the context manager could retrieve the context elements needed to display just the excerpt of the mark (but nothing surrounding it). In either case, the context manager needs to transform context elements to a format such as HTML or GIF, so the patron application may render the view. This transformation may not be easy for some base types. This problem is more likely to occur with Alternatives B, D, and E, because the manager modules run outside the patron's computer, and they are unable to "call back" base applications on a patron's computer. Mechanisms do exist for manager modules to call back applications on patron's computers, but they present some concerns (for example, security). Also, calling back may require the manager modules to cope with many versions of the same base application across patrons' computers.

Commonality and licensing are important issues related to base applications. Some base applications, such as an HTML browser, may be expected on all patrons' computers, whereas an application such as Adobe FrameMaker® is unlikely to be available on all patrons' computers. Some base-application vendors may disallow patrons from using installations resident on a DL server (or may require large license fees).

We have thus far discussed distribution and sharing of only executable code, but not the distribution and sharing of data. Sharing annotations is an emerging need among DL patrons. When sharing superimposed information, the corresponding marks may be shared or replicated. It is also possible to share just the marks, but not the superimposed information that uses them. In reality, we envision that some marks and superimposed information may be shared, and some marks may be replicated.

Distributing components and sharing information could each increase security risks. DL systems may need to implement more than one alternative to balance security and performance. A small number of DL server interface points, and narrow functionality of those interface points can help reduce risk. The number of interface points in the SPARCE manager modules is far fewer than that in most base applications. They are also narrower in functionality. For example, The Mark Manager module has fewer than twenty *methods*, whereas the MS Word 10.0 type library has more than 30 top-level *objects* (each object has many methods) relevant to this discussion [7]. The MS Word Range object alone has 15 methods that return a "primitive" value and 21 methods that return complex objects or collections. It may be better to present an interface to the manager modules rather than to base applications if a patron connects to a DL server over a public network such as the Internet, but base applications may be exposed to a patron connecting over a trusted intranet.

In discussing the distribution alternatives, we mentioned that combining requests to retrieve context can help reduce latency. Such intelligence may be added to the Context Manager, thus benefiting all patron applications. Caching context may also be useful for some applications. Cached context could be used to minimize the number of round trips to a base application. Cached context could be useful when a base application or base document is inaccessible (ignoring issues of cache consistency). The location of the cached context may be chosen based on needs. A context cache placed within a DL server can help with requests from many patrons, whereas a cache on a patron's computer can help with requests from only that patron. Alternatives D and E provide an excellent location (the middle tier) for such a cache.

Caching, replication and pre-fetching of base documents can also reduce latency in some cases. Caching could also enable offline access to base documents. The location of a cached document (or a replicated or pre-fetched base document) has some unique constraints compared to those for location of cached context. Unlike cached context, cached documents need to be accessible to base applications if a patron wishes to navigate to the document or retrieve context from it. Consequently, the document cache must reside on the patron's computer in Alternative A[2]. A document cache would probably not help in Alternatives B, C, and D because the base applications and base documents are collocated (on the DL server). However, Alternative E provides an excellent location (the middle tier) for a document cache. In general, a document cache could help when the DL server functions as an information server.

DL-server load balancing is another aspect that deserves consideration, especially in Alternatives B, C, and D. (It deserves greater consideration in Alternative B

[2] Other reasons, such as offline access by locally installed base applications, may motivate caching base documents on a patron's computer even when they are inaccessible to base applications controlled by the manager modules.

because both manager modules and the base applications run inside a DL server.) A DL server may replicate instances of the manager modules and the base applications to handle large number of requests. Replicated modules can serve concurrent patron requests, but in practice there is an upper limit to the concurrency obtained. The stress on system resources (for example, available main memory) will eventually prevent further replication, or even slow down existing replicas.

Dynamically choosing a distribution alternative provides an interesting means to reduce load on a DL server. For example, when using Alternative C, the manager modules could instantiate a base application on a patron's computer if it is locally available (thus switching to Alternative A), and use base applications on the DL server in other cases. Alternatives D and E could be mixed similarly (in the middle tier). Distribution architectures could also be different for different base types.

6 Related Work

The DELOS-NSF Working Group's report on Digital Library Information-Technology Infrastructures [1] highlights the importance of services and infrastructure for metadata and annotation services in DLs. OAI-PMH [13] and its extension XOAI-PMH [12] have demonstrated the feasibility of component-based architectures for metadata and annotation services respectively in a DL.

The UC Berkeley Digital Library Project uses *superimposed behaviors* in multivalent documents to support annotations [17]. That work facilitates distributed annotation and base data, but not distribution of executables. The Stanford InfoBus [9] defines a mechanism for interaction among UI clients, proxies, and repositories. (SPARCE's manager modules may be viewed as proxies.) It also defines service layers that are available to clients, proxies, and repositories. However, it does not consider distribution of executables.

FEDORA [10] and XOAI-PMH [12] provide promising frameworks for integration of superimposed information services with other DL services. The *parameterized disseminators* of FEDORA could be used to address (access) parts of documents. XOAI-PMH does not explicitly specify sub-document objects, but its extensibility mechanism could be used to create *item* instances that serve a similar purpose.

Caching, replication, and pre-fetching of DL documents have been researched more extensively [2, 3, 4] than caching of context information. Past work on caching metadata in hierarchical harvesting systems [5] provides useful insight into issues related to caching context information.

Component-based systems and distributed systems have been used in many domains. Wang and Fung [16] discuss the influence of architecture choices on component-based systems. Verissimo and Rodrigues [14] describe models for distributed systems and provide some case studies.

7 Summary

We have described five distribution alternatives to provide superimposed information services in DLs and defined some metrics to compare the performance of the

alternatives. We have illustrated the distribution alternatives using SPARCE, our middleware architecture for superimposed information management. We have also discussed some of the advantages and disadvantages of employing the distribution alternatives.

References

1. DELOS-NSF Working Group on Digital Library Information-Technology Infrastructures: Report (2002). Available online: http://www-rocq.inria.fr/~abitebou/pub/DELOS-ITI.pdf
2. Hollmann, J., Ardö, A., Stenström,. P.: Prospects of caching in a distributed digital library.Technical Report 03-04. Department of Computer Engineering, Chalmers University of Technology (2003)
3. Hollmann, J., Ardö, A., Stenström,. P.: An evaluation of document prefetching in a distributed digital library. In: Proceedings of the 7th European Conference on Research and Advanced Technology for Digital Libraries. Lecture Notes in Computer Science, Vol. 2769, Springer (2003) 276-287
4. Kovács, L., Micsik, A.: Replication within Distributed Digital Document Libraries. In: Proceedings of the 8th EDRG Workshop, Trondheim, Norway (1995)
5. Liu, X., Brody, T., Harnad, S., Carr, L., Maly, K., Zubair, M., Nels, M,L.: A Scalable Architecture for Harvest-Based Digital Libraries. D-Lib Magazine. Vol. 8, Number 11 (2002)
6. Maier, D., Delcambre, L.: Superimposed Information for the Internet. In: Informal proceedings of WebDB 1999, Philadelphia, PA (1999) 1-9
7. Microsoft Corporation: Microsoft Word Visual Basic Reference. Available online from: http://www.msdn.microsoft.com/library/
8. Murthy, S., Maier, D., Delcambre, L., Bowers, S.: Putting Integrated Information in Context: Superimposing Conceptual Models with SPARCE. In: Proceedings of the First Asia-Pacific Conference of Conceptual Modeling, Dunedin, New Zealand (2004) 71-80
9. Roscheisen, M., Baldonado, M., Chang, C., Gravano, L., Ketchpel, S., Paepcke, A.: The Stanford InfoBus and Its Service Layers: Augmenting the Internet with Higher-Level Information Management Protocols. Digital Libraries in Computer Science: The MeDoc Approach. Lecture Notes in Computer Science, Vol. 1392, Springer (1998)
10. Staples, T., Wayland, R., Payette S.: The Fedora Project: An Open-source Digital Object Repository Management System. D-Lib Magazine. Vol. 9, Number 4 (2003)
11. Suleman, H., Fox, E.A.: A Framework for Building Open Digital Libraries. D-Lib Magazine. Vol. 7, Number 12 (2001)
12. Suleman, H., Fox, E.A.: Designing Protocols in Support of Digital Library Componentization. In: Proceedings of ECDL 2002. Rome, Italy (2002)
13. The Open Archives Initiative: Protocol for Metadata Harvesting, Version 2.0. (2002)
14. Verissimo, P., Rodrigues, L.: Distributed Systems for System Architects. Kluwer Academic Publishers, Massachusetts (2000), ISBN: 0-7923-7266-2
15. W3C XML Protocol Working Group: Simple Object Access Protocol. (2003)
16. Wang, G., Fung, C.K.: Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Hawaii (2004)
17. Wilensky, R.: Digital library resources as a basis for collaborative work. Journal of the American Society of Information Science. Vol. 51, Number 3 (2000) 228-245

Towards a Service-Oriented Architecture for the Adaptive Delivery of Heterogeneous Cultural Resources

Jérôme Godard[1,⋆], Frédéric Andrès[2], Elham Andaroodi[1], and Katsumi Maruyama[2]

National Institute of Informatics, The Graduate School,
Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan
& SOKENDAI - The Graduate University of Advanced Studies,
Shonan Village, Hayama, Kanagawa 240-0193, Japan
[1]{jerome,elham}@grad.nii.ac.jp,[2]{andres,maruyama}@nii.ac.jp

Abstract. Adaptive systems are becoming essential for supporting the overload and the diversity of digital documents to be archived, retrieved and disseminated. They indeed represent the most promising solution for individuals to be able to handle the amount of data which they are daily flooded with during their professional activities or on their personal devices. However, providing adaptive management of heterogeneous resources remains an important research issue as it requires extensive and global environmental knowledge management to be effective. Communities, as they are sharing interests and accesses to resources, present very interesting characteristics that enable systems to deliver automated and personalized services; the scope of such processes being to take advantage of collaborative involvement in order to provide relevant knowledge management to users, to ensure the consistency of data manipulation, and to improve the distribution of resources within communities. We propose an architecture that complies with this vision and provide a case study based on the elaboration of a collaborative digital archive dedicated to the historical silk roads.

1 Introduction

This paper gives an overview of a generic architecture we are currently building as part of the the Digital Silk Roads project (DSR [16]). Initiated by UNESCO and NII, DSR aims at providing a wide area collaborative repository and portal for research and education in order to collect, to archive, to organize and to disseminate all the *relevant* information that can be gathered about the historical silk roads. This implies to deal with any kind of multilingual digital document; it goes from architecture-related pictures showing parts of buildings to records of traditional songs that characterize some social behaviors. This project involves more than 400 experts in many fields providing annotated *resources* (i.e. monotype multimedia digital documents) and has a set of end users that is indefinite. Therefore, DSR needs to provide adaptive services to users, by taking advantage of all the knowledge that is available on the *environment* (user

⋆ This research has been partially supported by a grant (*bourse Lavoisier*) from the French Ministry of Foreign Affairs (*Ministère des Affaires Etrangères*).

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 112–129, 2005.

himself, communities he is involved in, and device he is using). This vision requires to define an advanced model for the classification, the evaluation, and the distribution of multilingual multidisciplinary cultural *resources*. Our approach fully relies on state of the art knowledge management strategies. We define a global collaborative architecture that allows us to handle resources from the gathering to the dissemination.

In the following section, we introduce our testbed project and its portal. Then, after motivating our interest in ontology, we present our information management model in section 3. Based on this model, the fourth section describes and defines the personalized services we are providing for collaborative environments. The last section will summarize this overview and introduce some forthcoming issues.

2 DSR Framework

2.1 Description

We are involved in the Digital Silk Roads project (DSR [16]), which is focused on the collaborative management of digital multilingual cultural documents; it is handling all kinds of multimedia documents (including text-based, image, audio, video types) related to the historical silk roads. DSR aims at creating a global repository that enables us to collect, validate, preserve, classify and disseminate cultural resources. Building such a system is a great challenge, and requires to fulfill some commitments: first, it must provide an appropriate knowledge management framework that supports all the tasks it aims at covering. Then, documents and annotations have to be gathered and to be well structured. Thirdly, the data has to be stored safely. Afterwards, it is necessary to ensure an accurate access to the resources for each user. Finally, the distribution of the information has to be optimized in order to propose adaptive services. The global framework of DSR with its data distribution scheme is illustrated on Fig.1.

The main issue we are facing with DSR is heterogeneity; we are considering very varied data (as said before), users and devices. DSR users are divided into two categories: contributors and end users who are supposed to manipulate any kind of device (from mobile phones to mainframe computers). Since DSR is a collaborative project, it is important to register the users as members of communities (NB each community has an *access point*, which is a device being a kind of sub-server dedicated to the community); this enables us to increase the environmental knowledge that is required for performing adaptive services based on users' status and abilities.

2.2 The *myscoper* Platform, an Open Archives Initiative Based Digital Archive

Several research projects such as the arXiv e-print archive[1], the Networked Computer Science Technical Reference Library (NCSTRL)[2] or the Kepler project [14] in the field of digital libraries or digital research archives, tried to solve issues of sharing research information. They generally provide a common interface to the technical report collection based on the Open Archives Initiative (OAI) infrastructure[3]. This mechanism

[1] arXiv.org e-Print archive: http://arxiv.org/

[2] Networked Computer Science Technical Reference Library (NCSTRL): http://www.ncstrl.org/

[3] Open Archives Initiative http://www.openarchives.org/

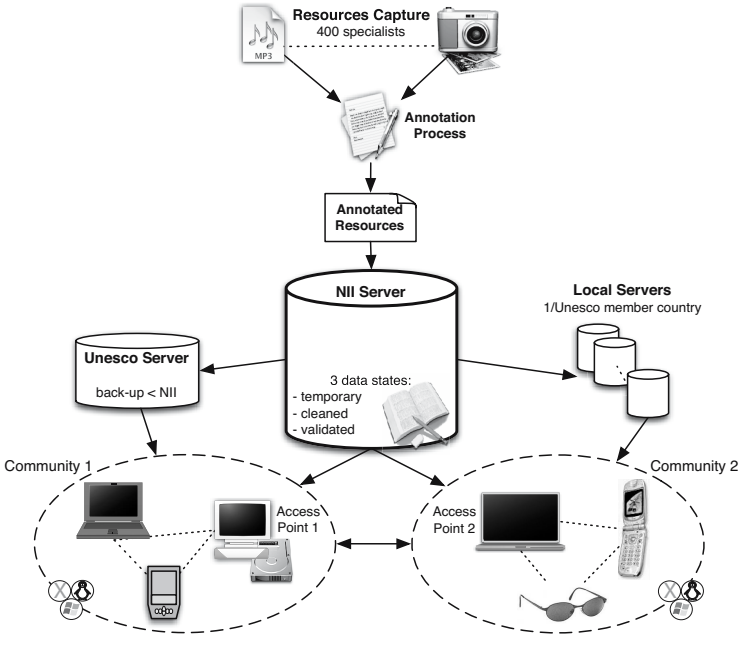


Fig. 1. DSR framework

enables interoperability among large scale distributed digital archives. In many cases, the network environment services include automated registration service, tracking of connected clients, and harvesting service of clients' metadata. Query service enables accesses to resources and to its related metadata. OAI has created a protocol (Open Archives Initiative Protocol for Metadata Harvesting, OAI-PMH) based on the standard technologies HTTP and XML as well as the Dublin Core metadata scheme[4]. OAI presently supports the multipurpose resource description standard Dublin Core which is simple to use and versatile. Shortcomings of such research projects generally include a too general metadata attributes schema for fine-grained information (e.g. cultural domains) and the non-support of community building. However, OAI-PMH itself has been created to provide an XML-wrapper for metadata exchange. It has been extended in the Digital Silk Roads project to support multi-disciplinary metadata schemas such as Object ID[5] for historical buildings, Categories for the Description of Works of Art (CDWA) for historical artifacts, or VRA[6] for visual resources. In order to avoid the different shortcomings and to provide a community framework for the research and education on Digital Silk Roads, we proposed and built the advanced scientific portal for international cooperations called *myscoper*[7]. *myscoper* is OAI-PMH 2.0 compliant

[4] Dublin Core: http://dublincore.org/
[5] Object ID http://www.object-id.com/
[6] Visual Resources Association: http://www.vraweb.org/
[7] *myscoper* will be soon accessible at: http://www.myscoper.org

as part of the distributed collaborative architecture as it is shown in Fig.2. The platform provides services for data handling, registration for identification, and metadata handling based on cross-disciplinary metadata schemas to create OAI-compliant metadata and resource management. Researchers can annotate resources according to their point of views and can share their comments according to cross-disciplinary and multi cultural backgrounds. Furthermore, the cultural resource server includes an ontology management service to support multi-lingual ontologies of cross-disciplinary metadata standards and multi-lingual ontologies in Digital Silk Roads related fields (e.g. architecture, history, geography, art...). We currently use Protégé 2000[8] as the Ontology manager. The start point of *myscoper*'s storage management has been the Dspace[9]. We also have extended Dspace core system to produce a multi-lingual platform and to support DSR metadata.

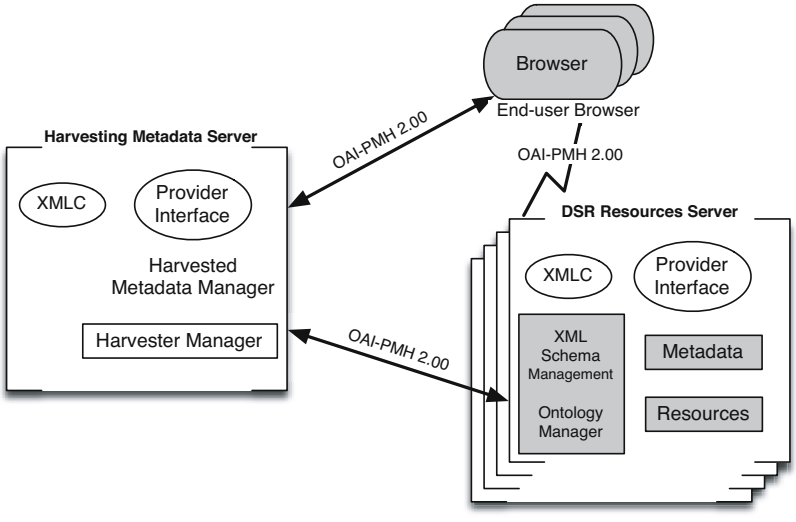


Fig. 2. *myscoper* architecture

3 Knowledge Management

3.1 Handling Ontologies

DSR's *myscoper* portal has to deal with large repository of multimedia of historical and cultural resources which come along with the web. For instance, it must provide access to databases containing cultural heritage photography. Meanwhile semantic understanding, access and usages of these materials are not fully possible due to the semantic gap for their annotation and retrieval [17]. There are still shortcomings of appropriate methods and tools for multimedia annotation, browsing and retrieval to help the users to

[8] http://protege.stanford.edu/
[9] DSpace Federation http://www.dspace.org/

find what they are really looking for. On the other hand, historical and cultural content of these databases make the process more complicated as there might be different semantic interpretations toward the subject of the visual information. Development and application of multi-lingual multimedia ontologies for the conceptual recognition of the content of cultural heritages of silk roads by using domain knowledge is the approach of *myscoper*'s to improve multimedia semantic annotation and retrieval. Ontology is defined as a specification of a conceptualization or as a set of concept-definition, a representational vocabulary[10]. Another definition of ontology which emphasizes the component-base recognition of a subject is a declarative model of the terms and relationships in a domain or, the theory of objects[11]. Based on these definitions ontology provides a hierarchical structured terminology of a domain and is completed by defining different relationships between term-sets. Ontology is explicitly declared to be helpful for knowledge representation, knowledge sharing and integration, portability issues... Ontology has application in artificial intelligence, natural language processing, multimedia database... Examples of ontology can widely be found in Biomedicine. Meanwhile recently in the field of multimedia enhanced annotation and retrieval, like photo annotation and ontology-based image retrieval and in some cases with relation to cultural heritage and art objects ontologies are designed and applied. In the field of cultural heritage, through application of domain ontology, the domain experts can develop semantic annotation for the multimedia data like images, and users can have access to a model of the vocabularies of the subject which will guide them for a more standard and intelligent search through database and will lead to a better retrieval.

DSR is directly involved in servicing ontology management on a case study of architectural cultural heritage named caravanserais[12]. This ontology tries to design a visual lexical model of terms or components in architectural relic and relationship between components based on the physical and spatial characteristics of the components. It also tries to design the ontology in different languages with the help of UNESCO expert team in order to exchange the content with experts and cover the needs of multilingual users [2]. This ontology, which is designed with Protégé 2000 environment (version 3.1) will be accesible as part of *myscoper* and will be used by domain experts in order to reach a consensus for its content to be extended to other languages and typologies of architectural heritage. Developing ontology on this case study as part of the portal is considered as a proper example for involvement of domain experts over internet in knowledge management and application of it can help enhanced access to large visual data which DSR is dealing with.

3.2 Architecture

Information modeling must be based on a coherent and powerful structure that can support all the layers of the architecture it is applied to (physical, logical, semanti-

[10] Gruber-Tom, "What is an ontology?" http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[11] Roberto Poli, "Framing Ontology - Second Part" http://www.formalontology.it/Framing_second.htm

[12] This multi-lingual ontology on architecture is constructed as part of a collaboration between National Institute of Informatics in Japan and the Architecture school of Paris Val de Seine in France under the *Digital Silk Roads Initiative Framework* in cooperation with UNESCO.

cal, transactional). In our framework, it has to handle physical entities (servers, *access points* being devices used as sub-servers for communities, and fixed and mobile devices; see the illustration on Fig. 1), *resources* (i.e. any mono-type multimedia document that can be related to at least one topic), knowledge management entities (*resource descriptions*, community's, user's, and device's *profiles*), and semantic elements (types of documents, i.e. *categories*, attributes, i.e. *descriptors*, and characteristics, i.e. *descriptors' values*). This information structure relies on contextual information for personalizing the retrieval and the distribution of multimedia documents. Since we want to provide a generic solution, our model is deeply related to XML; as a matter of fact, the Extensible Markup Language, with its ever-increasing number of extensions (and numerous drawbacks...), has become the standard for data analysis and exchange, and perfectly fits the requirements for handling annotation.

Transactions have to support distributed and heterogeneous constraints coming both from the users and the data. From our point of view, the best way to achieve efficiently this goal is to process database-like services on devices, with processes that behave quite autonomously. Many aspects have to be considered in order to provide such a complicated distributed document management; e.g. data model (supporting both storage and transaction processes), databases operators (indexing, querying...), transaction protocols (e.g. JXTA to provide mixed strategy: Peer to Peer for transactions between devices, and C/S for transactions between devices and DBMS server).

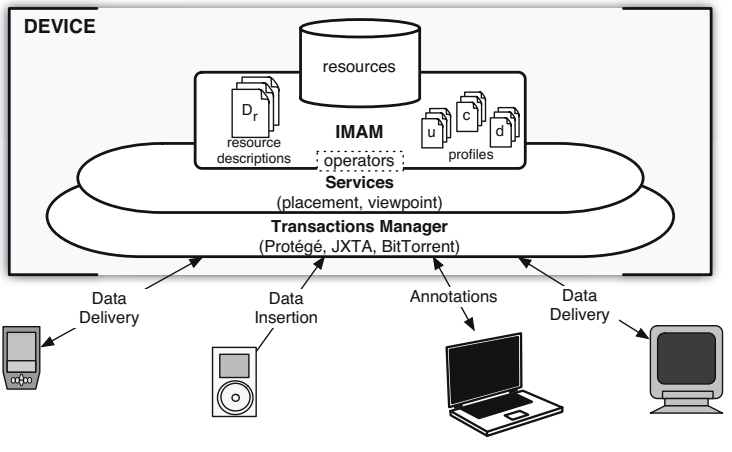


Fig. 3. Architecture

Thus, we propose an architecture (see Fig. 3) that can be applied to any device having some computing power and memory space. The main component of this framework is an information modeling that enables systems to structure and categorize any piece of knowledge, which might be involved in the access of shared *resources*. This modeling called IMAM (the following section presents it in details) relies on two knowledge structures: the *resource description* and the *profile*. These metadata sets are im-

plemented in XML and enable services to perform adaptive tasks (namely *placement* and *viewpoint*, see Sect. 4), which are processed through transactions based on a P2P protocol; the interface between the metadata and the transaction manager is performed by Protégé. The interactions between the devices are ideally fully decentralized. However, as we will explain it later, it is currently realistic to partially rely on a centralized network structure in order to ensure the safety of the resources and the reliability of the services.

3.3 IMAM, an Information Modeling for Adaptive Management

We have the great opportunity for DSR to be working with more than 400 specialists in various fields (using 21 languages) who are *motivated* and *able* to annotate documents very accurately. These annotations become very valuable once they are related to the knowledge structure presented above. Then, we need a unified model that enables us to capture this useful information about the documents and also the available knowledge about communities, users and devices.

Metadata is descriptive information that can be applied to data, environmental entities, or applications; it enables applications to capture and handle information embedded within the file and into a content management system. Metadata content goes from structural specifications to deep semantic descriptions. Most of current usage of metadata is related to the management of database schemas, interface definitions, or web pages layouts; it is in fact easy to notice that structural metadata is quite simple to produce and to manage, whereas semantic manipulation of information is a huge challenge. Relevant descriptions, searchable information, and up-to-date author and environmental information can be captured in a format that is eventually understood by users as well as by software applications, and hardware devices. We claim that information systems do not enough take advantage of metadata, and so are missing great opportunities to improve their knowledge management tasks. Therefore, two solutions appear for collaborative platforms that aim at manipulating complex sets of metadata:

- To build an application that integrates and exploits any kind of metadata structure; it implies to map all the structures to each others and to update the mapping each time a new structure appears.
- To define a generic metadata structure that supports any kind of multimedia documents.

The first solution generates many inconsistency issues and becomes very heavy as the number of mapped structures grows. The second solution first seems too restrictive and complex; but it is quite possible to make it more flexible by defining a core structure that can be extended for particular uses. The MPEG-7 standard also known as *Multimedia Content Description Interface*[13] aims at providing standardized core technologies allowing description of audiovisual data content in multimedia environments. MPEG-7, with its Description Schemes and Description Definition Language, has an interesting approach. Unfortunately, as its design goes deep into details and is quite complicated,

[13] http://www.itscj.ipsj.or.jp/mpeg7/

common users have been reluctant to use it. Adobe is providing an open source, W3C-compliant way of tagging files with metadata across products from Adobe and other vendors, called Extensible Metadata Platform[14] (XMP). XMP is extensible, meaning that it can accommodate existing metadata schemas; therefore systems do not need to be rebuilt from scratch. However, many companies and communities reject it because of its origins. Finally, W3C recently provided a recommendation describing CC/PP[15] (Composite Capabilities/Preference Profiles) which defines the description of device capabilities and user preferences as a profile. This structure, using RDF, is quite attractive as it contains very precise and coherent vocabularies. Unfortunately, its complex and very detailed structure makes it unusable for most of the users.

In order to categorize and describe *resources* (i.e. any mono-type multimedia document that can be related to at least one topic), we use a thesaurus-like knowledge tree called *Resource Categorization Tree* (RCT, [7]); a node of the RCT is denoted α_i. The knowledge management through the RCT is based on a contextual structure, which is made of a core structure (RCT_{core} enables communities to share a minimal knowledge structure that can categorize any *resource*) on which extensions can be fixed (making it possible for communities to extend and refine the branches of RCT_{core} they are interested in). Figure 4 shows an extract (one path) of the RCT in

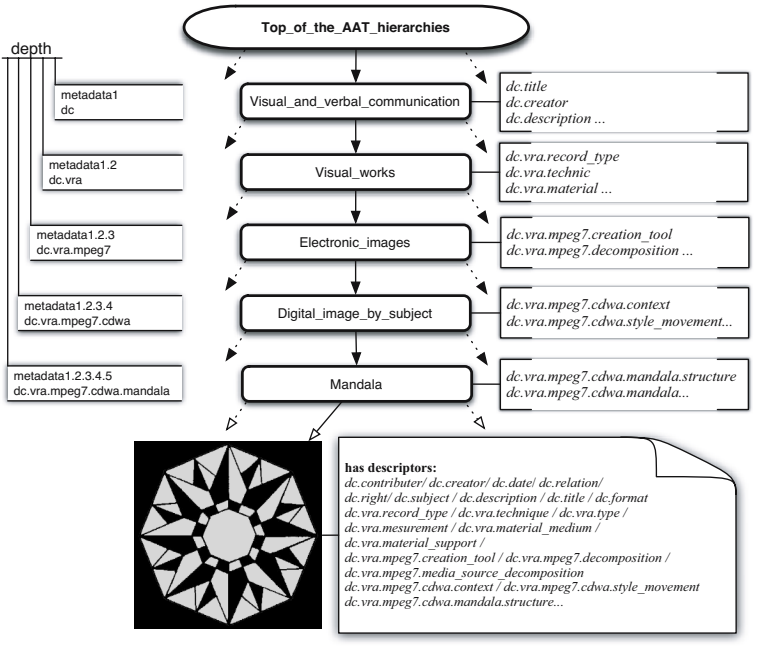


Fig. 4. RCT extract

[14] http://www.adobe.com/products/xmp/main.html
[15] http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/

java-like language by mapping metadata sets on the AAT hierarchy. First, AAT categorizes cultural resources by format and content criteria (e.g. visual works). Then, the hierarchical mapping provides a specialization of the metadata properties describing the resources along the hierarchical structure of AAT from generic nodes such as *Visual_and_verbal_communication* to community descriptive nodes such as *Mandala*. Finally the mapping of the metadata takes advantage of the hierarchical classes of metadata *dc.vra.mpeg7.dcwa.mandala* and the inheritance mechanism between each node and its sub-nodes.

Our model aims at describing as clearly as possible the information contained in the annotations provided on the *resources*. The primary element in this approach is the *descriptor* (δ), which is a contextual attribute. The value assigned to the j^{th} *descriptor* of the i^{th} node for a specific *resource* is denoted $\sigma_{i,j}$. The whole annotation about a *resource* is contained in a complete branch of the RCT called *Resource Description* and defined as follows: $D_r = \left(<\alpha_i>, <\delta_{i,j}>, <\sigma_{i,j}> \right)_{\substack{i=0,\ldots,m-1 \\ j=1,\ldots,p}}$. The information about communities, users, and devices is contained in a quite similar structure called *profile* (denoted π) that is composed of *descriptors* and *descriptors' values* (without nodes): $\pi = (<\delta_v>, <\sigma_v>)$. We specified DBMS-like operators [6] with the aim to powerfully manage the *resources* (e.g. *create, insert, edit, intersection, difference...*) by using IMAM.

4 IMAM's Services

4.1 Preamble

Services to be proposed to DSR users cover usual database functions, personalized automated processes, and management of transactions in order to ensure the capability of the system to work in heterogeneous distributed mobile environments. In fact, more and more people are showing strong interests in peer-to-peer [1] as a foundation for creating advanced distributed applications; moreover, innovative sharing strategies are implemented and used in peer-to-peer [13,15,3] and mobile systems [11,9]. But they are generally lacking in a unique generic basis for knowledge management that would allow us taking fully advantage of these powerful distributive environments. We agree that distributed knowledge management has to assume two principles [4] related to the classification: autonomy of classification for each knowledge management unit (such as community), and coordination of these units in order to ensure a global consistency. Distributed adaptive services require to exploit all the environmental knowledge that is available about the elements involved. An important category of this knowledge is related to devices' states; indeed, knowing if a device is on, in sleep mode, off, if its battery still has an autonomy of five minutes or four days, or if it has a wired or wireless connection, etc. helps adapting services that can be delivered to this device. For each device, we consider a state control that is part of the device's *profile*. And of course we use the information contained in communities' and users' *profiles*. The information that can be gathered in collaborative environments (i.e. people sharing interests and resources) shall increase the ability to create new kinds of services.

The number of applications using metadata to improve information retrieval processes and to deal with semantic heterogeneity is growing very fast; web services in particular already have several standards (e.g. DAML/OIL, ebXML) to describe service related information. Web-based Information systems have a typical structure which consists of three layers: semantic, application, and presentation. The Hera design methodology [19] considers integration and user support as aspects to be included within these three layers, which is a very relevant strategy according to us. Nevertheless, Hera uses a RDF-based ontology model which is very convenient but lacks context management support. Metadata is also part of many semantic management frameworks for multimedia documents (e.g. audiovisual resources [18]); but most of the time, these frameworks are dedicated to a precise type of data and/or to a specific domain.

Knowledge sharing is a wide area made up of many fields; moreover it covers different kinds of application, going from common memory space access to collaborative project management. It has been deeply investigated for many years and a lot of work has been produced (e.g. for software development teams [5]). As a matter of fact, most of the ontology-based applications are influenced by initiatives for the definition of interoperable metadata standards (such as Dublin Core). We also would like to point out that XML, with its large set of tools and extensions is commonly recognized as the best framework to contain metadata. The distribution of data within communities can be partially automated in order to reduce the query workload [8]; indeed, it is possible to evaluate what kind of data might be interesting or useful for a class of users (community), and for a specific user. It is very important to choose an appropriate heuristic [10] depending on the requirements related to users and environments in order to perform data placement. Then it becomes realistic to create automated data placement processes; an example of scheduled data placement [12] indicates that much benefit can be obtain without any human interaction.

Personalized services finally depend on user-related contexts such as localization, birth date, languages abilities, professional activities, hobbies, communities' involvement, etc. that give clues to the system about users' expectations and abilities. All this information is quite easy to extract and to manipulate through IMAM; in the remainder of this section, we present the two main adaptive services based on our model.

4.2 Authoritarian Data Placement

The main motivation for the data placement is to automatically copy resources that seems to be very relevant to a user or a community on the appropriate devices. This operator relies on memory spaces that are allocated on each device for the server to place the data. Each time a *resource* is added to the server, its *Resource Description* is used for analyzing the possible correlations with the communities and users interests. The strategy we are using to evaluate the significance of a *resource* placement on a device is quite similar to the one used for the operator *sim* (which evaluates the similarity between two *resources*, see [6]). But in the case of the placement (operator denoted *disp*), the *descriptors* are replaced by the *descriptors' values*. we extract the ratio of common *descriptors values* where the *descriptors* are similar by using the function ρ_D:

$$\rho_D(A, B) = \frac{\mid \textsc{Tinter}(A, B) \mid}{\mid \textsc{Tunion}(A, B) \mid} \in [0, 1], \text{ with:}$$

- TINTER(A,B) = $\{<\sigma_{inter}> \mid \sigma_{inter} = \sigma_{i,j} = \sigma_{k,l}, (\sigma_{i,j} \in A) \wedge (\sigma_{k,l} \in B)\}$
- TUNION(A,B) = $\{<\sigma_{union}> \mid (\sigma_{union} \in A) \veebar (\sigma_{union} \in B)\}$

where A and B contain ordered families of labels, which are lists of descriptors with associated values (there can be only one label, in the case of a profile for instance). $\veebar$ denotes operator *exclusive-or*.

The *disp* operator first applies ρ_D to communities. Depending on two threshold values s_{c_1} and s_{c_2} ($s_{c_1} > s_{c_2}$), we decide if the resource has to be placed on all the devices used in the community (Case 1 on Fig.7) or only on the community's *access point* (Case 2 on Fig.7); the operator dispatches the *resource* on all devices of a community for which the value returned by ρ_D is higher than s_{c_1}, and if the *resource* seems to be quite relevant only for a community (i.e. the returned value is between s_{c_1} and s_{c_2}), the operator copies the *resource* on the *access point* only. The last option for *disp*, when the *resource* does not seem to be relevant for a whole community (Case 3 on Fig.7), is to apply ρ_D on each user in this community; again, this is done by using a threshold value s_u. If the value returned by the function is higher than s_u, then the resource is placed on the user's device that is the most able to get it. The selection of the device is processed by the function SELECTDEV(i, j) i and j being integers, the function returns the device (profile) used by the j^{th} member of the i^{th} community that has the largest storage capacity on its placement area (see Fig.5). We have to mention that each time a

```
SELECTDEV(i, j)
1  device ← ∅
2  a ← 0
3  for k ← 1 to K_{i,j}
4       do if FSPACE(π_{d_{i,j,k}}) > a and STATE(d_{i,j,k}) = TRUE
5            then a ← FSPACE(π_{d_{i,j,k}})
6                 device ← π_{d_{i,j,k}}
7  return device
```

Fig. 5. The device selection function pseudo-algorithm

resource is supposed to be placed on a device, *disp* first checks the ability of the device to store the *resource* and if there is not enough space for it, the operator compares the new *resource* to the *less interesting resource* that is on the placement area of the device. If the new *resource* is more *interesting*, then it shall replace the other one. This is recursively done by the function PROEMIN(D, π, ρ), D being a resource description, π a device profile, and ρ a value between 0 and 1 (see Fig.6).

Before copying a *resource* on a device, *disp* checks if the device is online and if it has enough free space on its placement area (limited predefined space) for the *resource* to be stored. The storage capacity (full capacity and empty space) of a device is defined in order to ensure limits (depending on a minimum and a ratio) that cannot be passed over; the available space dedicated to automated services on the device must be precisely defined (default ratio or user's choice) in order to keep enough memory space for

```
PROEMIN(D, π, ρ)
 1  proemin ← TRUE
 2  if STATE(π) = TRUE
 3     then if ASPACE(π) > SIZE(D)
 4             then PUT(D, π)
 5             else  t ← GETWR(π)
 6                   if ρ > t[1, 1] and FSPACE(π) > SIZE(D)
 7                     then DELETE(t[1, 2], π)
 8                          PROEMIN(D, π, ρ)
 9     else  proemin ← FALSE
10  return proemin
```

Fig. 6. The Proemin function pseudo-algorithm

the user's *manual* activities. *disp* gets this states' information from the device *profile* via several functions:

- FSPACE(π_d) returns the full space allocated for placed data on the device d (in KB).
- ASPACE(π_d) returns the available space in the placement area on d (in KB).
- SIZE(D_{r_i}) returns the size of Resource r_i (in KB).
- STATE(π_d) returns FALSE if the device d is off, and TRUE if it is on.

Thus appears the update problem: variables we need to handle can change at any time very irregularly (frequency might anyway be taken into account); for instance, it is necessary to record the new locations of the *resource* in its *resource description* and devices' states. Indeed, to be able taking advantage of the dispatched *resources* for the query management, we have to keep a record of all the locations a *resource* is stored at. So each PUT and DELETE (see below) implies that the *Resource Description* (which contains all these locations within the *locations descriptor*) is updated. The new version of the *Resource Description* is first saved on the server, and then it overwrites the other copies that are on the devices containing the *resource*. The updates processes have to take into account the possibility for a device to be offline, and so to ensure that the update can be performed as soon as the device becomes available. Following the same strategy, when a device is switched on, it updates its IP address in its *profile*, which is copied on the server and related *access points* (we do not address here the case of connection loss because of space limitation). We also have to consider the creation of new communities: each time a community is created, the placement operator must be applied on the server to check what *resources* should be dispatched on the devices of this community. The function UPDATEPROF provides the support described above for every *Resource Description* and *profile* that has to be updated; it moreover propagates the new location where a resource is stored after a placement: the information is first recorded on the concerned resource description on the server, which then dispatches the update on all the copies of the resource descriptions.

We finally declare all the functions that *disp* uses in order to manipulate the *resources* and their *profiles*:

```
DISP(D_r)
 1  disp ← FALSE
 2  for i ← 1 to C
 3       do π_{c_i} ← GETPROFCOM(i)
 4          ρ_1 ← ρ_D(D_r, π_{c_i})
 5          device1 ← GETAC(π_{c_i})
 6          if ρ_1 ≥ s_{c_1}
 7            then for j ← 1 to U_i                                          ▷ Case 1
 8                      do π_{u_{i,j}} ← GETPROFUSE(i, j)                   ▷ Case 1
 9                         device2 ← SELECTDEV(i, j)                        ▷ Case 1
10                         if PROEMIN(D_r, device2, ρ_1) = TRUE             ▷ Case 1
11                           then UPDATEPROF()                              ▷ Case 1
12                                disp ← TRUE                               ▷ Case 1
13          elseif s_{c_2} ≤ ρ_1 < s_{c_1} and PROEMIN(D_r, device1, ρ_1) = TRUE   ▷ Case 2
14            then UPDATEPROF()                                            ▷ Case 2
15                 disp ← TRUE                                             ▷ Case 2
16          else for j ← 1 to U_i                                          ▷ Case 3
17                      do π_{u_{i,j}} ← GETPROFUSE(i, j)                   ▷ Case 3
18                         ρ_2 ← ρ_D(D_r, π_{u_{i_j}})                      ▷ Case 3
19                         device3 ← SELECTDEV(i, j)                        ▷ Case 3
20                         if ρ_2 ≥ s_u and device3 ≠ ∅                     ▷ Case 3
                           and PROEMIN(D_r, device3, ρ_2) = TRUE            ▷ Case 3
21                           then UPDATEPROF()                              ▷ Case 3
22                                disp ← TRUE                               ▷ Case 3
23  return disp
```

Fig. 7. The placement pseudo-algorithm

- PUT(r, d) accesses the placement area on the device d and pastes the Resource r there.
- GETPROFCOM(x) (x being the number (i) of the i^{th}community, or the community's identifier *comID*) returns the profile of the related community. GETPROFUSE(x) works in the same way for a user.
- GETAC(π_c) returns the profile of the *Access Point* of the community π_c.
- DELETE(x, π_d) deletes the resource identified by x on the placement area of the device d.
- Each device's profile contains a table $[r_i, \rho_i]_{i=1...n}$ made of n columns (n being the number of resources stored on the device) and two rows (resource identifier and related ρ_D values) such as ρ_D values are increasingly ordered. The function GETWR(π_d) returns this table for the device d.
- SELECTDEV(i, j) i and j being integers, the function returns the device (profile) used by the j^{th} member of the i^{th} community that has the largest storage capacity on its placement area (see Fig.5).

NB: some variables are shared and are accessible from all the functions dedicated to the services; it consists in all the profiles (communities ($< \pi_{c_j} >_{j=1,...,\mathcal{C}}$), users ($< \pi_{u_{j,k}} >_{k=1,...,\mathcal{U}_j}$), and devices ($< \pi_{d_{j,k,l}} >_{l=1,...,\mathcal{L}_{j,k}}$)), sets' number of elements ($\mathcal{C}$

is the total number of communities, $\mathcal{U}_j$ is the total number of users involved in the j^{th} community, $\mathcal{K}_{i,j}$ is the total number of devices used by the j^{th} user of the i^{th} community), and threshold values (s_{c_1}, s_{c_2}, s_u).

The *disp* placement operator has been introduced in [6]; we proposed here a full description of the pseudo-algorithm (see Fig.7) that moreover takes into account new features such as checking devices activity and storage capacity.

4.3 Adaptive Query Management

A major benefit of the RCT is to allow us giving an appropriate *viewpoint* (denoted ν) to each user for a same set of *resources* (taking the user's characteristics and environment into account). In fact, our *viewpoint* can be seen as a query optimizer, since it clears and modifies an initial set of *resources*. The *viewpoint* operator has initially been defined in [6] with only two sets of rules (re-ordering was integrated within the two other sets). This strategy was lighter and seemed more optimized. However, after simulating some simple tests, we realized that the two steps approach might generate incoherent resources management. Therefore, we added to the *viewpoint* a third function using re-ordering rules only. We provide here the updated full characterization of ν:

$$
\begin{array}{rrcl}
\nu = p \circ t \circ g : & \Lambda^p \times \Pi & \longrightarrow & \Lambda^q \\
& \left(<D_i>_{i=1,\ldots,p},\, \pi_e\right) & \overset{\Psi \circ \Theta \circ \Gamma}{\longmapsto} & <D'_k>_{k=1,\ldots,q}
\end{array}
$$

$$
\begin{array}{rrcl}
with \quad g : & \Lambda^p \times \Pi & \rightarrow & \Lambda^q \times \Pi \\
& \left(<D_i>_{i=1,\ldots,p},\, \pi_e\right) & \overset{\Gamma}{\mapsto} & \left(<D_j>_{j=1,\ldots,q}, \pi_e\right) \\
t : & \Lambda^q \times \Pi & \rightarrow & \Lambda^q \times \Pi \\
& \left(<D_j>_{j=1,\ldots,q},\, \pi_e\right) & \overset{\Theta}{\mapsto} & \left(<D'_j>_{j=1,\ldots,q}, \pi_e\right) \\
p : & \Lambda^q \times \Pi & \rightarrow & \Lambda^q \\
& \left(<D'_j>_{j=1,\ldots,q},\, \pi_e\right) & \overset{\Psi}{\mapsto} & <D'_k>_{k=1,\ldots,q}
\end{array}
$$

where p is the number of considered Resource Descriptions and q the number of returned Resource Descriptions ($q \leq p$), π_e is the *profile* of the *environment e* (with $\pi_e = \pi_u \cup \pi_d$, u denotes a user, and d a device), and Γ, Θ, and Ψ are three sets of selective rules:

- Γ contains acceptation rules denoted γ. If a descriptor value of the resource description D_λ does not respect a rule $\gamma_i \in \Gamma$, then the set returned by g does not contain D_λ.
- Θ contains transformation rules denoted θ. If a descriptor of the resource description D_λ is involved in any rule $\theta_i \in \Theta$ and if the corresponding value σ does not satisfy this rule, a new resource λ' (with the related $D_{\lambda'}$) will be created as the result of a modification applied to the resource λ by t according to instructions contained in θ_i.
- Ψ contains re-ordering rules denoted ψ. If a descriptor value of the resource description D_λ does not respect a rule $\psi_i \in \Psi$, then the position of D_λ in the set of

resource descriptions returned by p. This re-ordering depends on the result of the rules and then on the existing order in the original set of resource descriptions: any D_λ that does not respect rules in Ψ will be pushed behind the resources descriptions that respect all or more rules than D_λ (i.e. p rearranges in order the resource description sets by classifying decreasingly the elements respecting the larger amount of rules).

Each set of rules is deeply dependent on the domain the viewpoint is applied to. It is obvious that rules must be defined according to communities' and users' interests. Moreover, the rules rely on the available applications (especially for transformation rules). Each rule used by the viewpoint is a test on a pair of descriptor values; one from the *resource description* and the other one from the profile. Thus the syntax for each rule is very simple and relies on the fact that the rule is true or false for each test. Note that it is imperative to keep the order of the compound functions when applying ν; indeed, another order would generate inconsistencies in the management of resources as it might for instance create new resources and reject them afterwards.

The sets of rules, which the viewpoint is using (Γ, Θ, and Ψ), are contained in two different categories: we can consider Θ's rules results as commands for *resources* themselves to be adapted, when the other sets of rules adapt the already returned sets of resources; for instance, an image, that has a bigger resolution than the one of the user's screen, would be reduced to the screen resolution. This strategy is very useful for distributed systems and heterogeneous environments since it reduces the bandwidth consumption and fits devices characteristics (especially mobile devices). It is imperative to define the transformation rules according to the server software environment; in the DSR case, we use some applications providing image management, text summarization... Then it becomes trivial to manage the information, and to apply the modifications depending on the descriptor values.

As an illustration, we consider the scenario of an researcher being a member of DSR, who looks for resources that contain maps of the historical silk roads. A typical query in that case would be a set of terms such as $<$ *maps silk roads* $>$; the query is directly sent to the server with the IDs of the user and of the device, so the server can select their profiles from its own memory. This kind of query on a repository which is dedicated to the silk roads would of course return a very large set of resources. Let us just consider a small set of resource descriptions $S = < D_{r_1}, D_{r_2}, D_{r_3} >$ (in order to make the example simple and short) where:

- r_1 is a high resolution map covering the whole Asia and showing the main historical silk roads with comments written in English.
- r_2 is a movie file containing a short documentary on the silk roads in Iraq.
- r_3 is a low resolution satellite picture of Iraq where silk roads have been drawn with comments written in Arabic.

As it has been explained above, the viewpoint is a compound of three functions so the refined selection process is done in three steps:

1. Selection. The first set of rules applied by function g might remove resources descriptions which do not respect at least one rule. g is considering descriptors in the

environmental profile that are involved in at least one rule. In our example, one rule checks the size of the resources and the tuple (available memory space, bandwidth) from the profile. The movie, with a size of 30MB, exceeds both limits from the tuple as the user is processing his query from his mobile phone. So g returns the set: $< D_{r_1}, D_{r_3} >$ (note that it means that both resources passed the tests).

2. Modification. The second set of rules applied by function t can modify some resources (and then creates a new resource description) if a resource of a certain type exceeds a threshold defined in the rule or in the profile for this type of resource. Then the rule calls an application which is able to modify the resource so it would not exceed the threshold anymore. This is the case with r_1, which resolution is very high and exceeds the resolution of the mobile phone screen. Thus t calls a resampling application that reduces the resolution of r_1 until it reaches the screen resolution; then, a new resource r'_1 is created and replaces r_1. t returns $< D'_{r_1}, D_{r_3} >$.
3. Reordering. Finally, the last set of rules applied by p can reorder the set if it considers that a resource with a higher priority (or value) is behind a resource with a lower priority. For this kind of rule, a very appropriate evaluation relies on the languages the user can understand. Here, as the researcher is an Iraqi, and so is fluent in Arabic whereas he has a poor English level (reminder: the values for the language descriptor are ordered). Then, p will return $< D_{r_3}, D'_{r_1} >$, which will be the result of the *viewpoint*.

In fact, our query optimization strategy, as a distributed and decentralized operation, would require a large CPU contribution from the devices as they have to apply the *viewpoint* on all the *Resource Descriptions* that they are receiving from other devices. This would be especially true for re-ordering rules as they compare each rersource description to all the others in the considered set and require to add some temporary factors (e.g. the number of rules, which the resource description follows). Moreover, since we are dealing with high resolution multimedia *resources* and as we are reasonably convinced that portable devices processing capacity will soon drastically increase, we claim that the benefits of the *resources*' selection worth the overload on devices CPU and primary memory. We are still investigating the best solution for such a decentralized distributed query management to be effective.

However, the architecture proposed for DSR in Fig. 1 relies on a partial centralized strategy for the processing of queries; therefore, the ability to perform a reliable query optimization (regarding time consumption) depends on the complexity of the rules that are used by the *viewpoint*. The main constraint here is the capacity of the server to process the operations. Thus, communities' servers, which are dedicated to the usage of IMAM services must be able to cover the operative costs that the viewpoint requires.

5 Conclusion

The framework described in this document corresponds to a long-term vision. Indeed, it requires much time to define the knowledge management structure (i.e. multilingual ontologies and interrelationships) and to implement the portal that has to gather all the

information and the system that provides the services. At least, we now have a solid basis for the management of the resources that allow us to design innovative services.

The interesting features we are getting from the *disp* operator and the *viewpoint* can then be enhanced by using an appropriate query management based on our three-layers architecture (server, *access point*, normal device). We are currently designing a distributed query manager based on JXTA and BitTorrent[16] (P2P delivery system); in fact *resource descriptions* can partially seen as BitTorrent *trackers*, as they contain all the locations of the *resources*. We now just have to take advantage of IMAM's support to provide appropriate *resources* to users in the best conditions. Following BitTorrent strategy, we can provide distributed query processing by using the placed and indexed data; then a device can access all copies of a *resource* (even not complete ones).

A simple example of what we want achieve with IMAM: let us consider a class studying caravans in Iraq with a focus on a the 14^{th} century and looking for information about people exchanging specific goods. It would be interesting and useful for the students to get on their laptop maps, pictures, videos that are related to their topic. This could be done by creating the community some time before the class starts this lesson. The placement would be then restricted by the viewpoint, being used as a filter for the sets of resources to be sent on each device. Then the distribution should be improved (regarding time and bandwidth consumption) by a BitTorrent-like community P2P shared access on the resources. Moreover, a unified protocol based on JXTA can enable the whole process to be more efficient and safer.

However, many issues are remaining; we need to implement the transaction and query managers in order to enable users to start using all the portal functionalities. Then, we have to define some policies to evaluate the relevance and efficiency of adaptive services (e.g. how to fix the threshold values). We also would like to point out that the significance of this architecture is not restricted to collaborative cultural projects; we see valuable possible application in the management of companies' resources.

Acknowledgements

We would like to thank NII for the support under the International Cooperation Framework and MEXT for the Scientific Research Grant-in-Aid to the Geomedia project.

References

1. Serge Abiteboul. Managing an XML Warehouse in a P2P Context. In *Proc. of CAiSE*, pages 4–13, Klagenfurt, Austria, June 16-18 2003.
2. Elham Andaroodi, Frédéric Andrès, Kinji Ono, and Pierre Lebigre. Ontology for caravanserais of Silk Roads: Needs, Processes, Constraints. In *Proc. of the Nara Symposium for Digital Silk Roads*, pages 361–367, Nara, Japan, December 10-12 2003.
3. Neal Arthorne, Barbak Esfandiari, and Aloke Mukherjee. U-P2P: A Peer-to-Peer Framework for Universal Resource Sharing and Discovery. In *Proc. of the FREENIX Track: USENIX Annual Technical Conference*, pages 29–38, San Antonio, Texas, USA, June 9-14 2003.

[16] http://www.bittorrent.com/

4. Matteo Bonifacio, Paolo Bouquet, and Roberta Cuel. The Role of Classification(s) in Distributed Knowledge Management. In *Proc. of KES*, Podere d'O., Italy, Sept. 16-18 2002.
5. Thomas Chau, Frank Maurer, and Grigori Melnik. Knowledge Sharing: Agile Methods vs. Tayloristic Methods. In *Proc. of WETICE*, pages 302–307, Linz, Austria, June 9-11 2003.
6. Jérôme Godard, Frédéric Andrès, William Grosky, and Kinji Ono. Knowledge Management Framework for the Collaborative Distribution of Information. In *Current Trends in Database Technology - EDBT 2004 Workshops (Revised Selected Papers), LNCS 3268*, pages 289–298, Heraklion, Greece, March 14 2004.
7. Jérôme Godard, Frédéric Andrès, and Kinji Ono. Management of Cultural Information: Indexing Strategies for Context-dependent Resources. In *Proc. of the Nara Symposium for Digital Silk Roads*, pages 369–374, Nara, Japan, December 10-12 2003.
8. Steven D. Gribble, Alon Y. Halevy, Zachary G. Ives, Maya Rodrig, and Dan Suciu. What Can Database Do for Peer-to-Peer? In *Proc. of WebDB*, pages 31–36, Santa Barbara, California, USA, May 24-25 2001.
9. Matthias Grimm, Mohammed-R. Tazari, and Dirk Balfanz. Towards a Framework for Mobile Knowledge Management. In *Proc. of PAKM*, pages 326–338, Austria, Dec. 2-3 2002.
10. Magnus Karlsson and Christos Karamanolis. Choosing Replica Placement Heuristics for Wide-Area Systems. In *Proc. of ICDCS*, pages 350–359, Tokyo, Japan, March 23-26 2004.
11. Panu Korpipää and Jani Mäntyjärvi. An Ontology for Mobile Device Sensor-Based Context Awareness. In *Proc. of CONTEXT*, pages 451–458, Stanford, CA, USA, June 23-25 2003.
12. Tevfik Kosar and Miron Livny. Scheduling Data Placement Activities in Grid. Technical Report 1483, Computer Sciences Department, University of Wisconsin, USA, July 2003.
13. Yugyung Lee, Changgyu Oh, and Eun Kyo Park. Intelligent Knowledge Discovery in Peer-to-Peer File Sharing. In *Proc. of CIKM*, pages 308–315, McLean, USA, Nov. 4-9 2002.
14. Kurt Maly, Mohammad Zubair, and Xiaoming Liu. Kepler - An OAI Data/Service Provider for the Individual. *D-Lib Magazine*, 7(4), April 2001.
15. Aloke Mukherjee, Babak Esfandiari, and Neal Arthorne. U-P2P: A Peer-to-Peer System for Description and Discovery of Resource-Sharing Communities. In *Proc. of ICDCS Workshops*, pages 701–705, Vienna, Austria, July 2-5 2002.
16. Kinji Ono, editor. *Proceedings of the Tokyo Symposium for Digital Silk Roads*, Tokyo, Japan, December 11-13 2001. UNESCO & National Institute of Informatics.
17. D.V. Sreenath, William Grosky, and Frédéric Andrès. *Intelligent Virtual Worlds: Technologies and Applications in Distributed Virtual Environments*, chapter Metadata-Mediated Browsing and Retrieval in a Cultural Heritage Image Collection. World Scientific Publishing Company, Singapore, 2002.
18. Chrisa Tsinaraki, Eleni Fatourou, and Stavros Christodoulakis. An Ontology-Driven Framework for the Management of Semantic Metadata Describing Audiovisual Information. In *Proc. of CAiSE*, pages 340–356, Klagenfurt, Austria, June 16-18 2003.
19. Richard Vdovjak, Peter Barna, and Geert-Jan Houben. Designing a Federated Multimedia Information System on the Semantic Web. In *Proc. of CAiSE*, pages 357–373, Klagenfurt, Austria, June 16-18 2003.

Analysis and Evaluation of Service Oriented Architectures for Digital Libraries

Hussein Suleman

Department of Computer Science, University of Cape Town,
Private Bag, Rondebosch, 7701, South Africa
hussein@cs.uct.ac.za

Abstract. The Service Oriented Architecture (SOA) that underlies the Web Services paradigm of computing is widely regarded as the future of distributed computing. The applicability of such an architecture for digital library systems is still uncertain, as evidenced by the fact that virtually none of the large open source projects (e.g., Greenstone, EPrints, DSpace) have adopted it for internal component structuring. In contrast, the Open Archives Initiative (OAI) has received much support in the DL community for its Protocol for Metadata Harvesting, one that in principle falls within the scope of SOA. As a natural extension, the Open Digital Library project carried the principles of the OAI forward into a set of experimental derived and related protocols to create a testbed for component-based digital library experiments. This paper discusses a series of experiments with these components to confirm that SOA and a service-oriented component architecture is indeed applicable to building flexible, effective and efficient digital library systems, by evaluating issues of simplicity and understandability, reusability, extensibility and performance.

1 Introduction

Service-oriented computing is a relatively new paradigm of computing where tasks are subdivided and performed by independent and possibly remote components that interact using well-defined communications protocols [24]. In particular, the Service-Oriented Architecture (SOA) refers to a framework built around XML and XML messaging, with standards for how messages are encoded, how protocol syntax is specified and where instantiations of services are to be located. These are exemplified by the SOAP [7], WSDL [3] and UDDI [9] specifications respectively. It is often argued that SOA can be adopted by an organisation to increase reuse, modularity and extensibility of code, while promoting a greater level of interoperability among unrelated network entities.

From a somewhat different perspective, the Open Archives Initiative (OAI) attempted to address interoperability first and foremost, by designing a protocol for the efficient incremental transfer of metadata from one network entity to another. This Protocol for Metadata Harvesting (PMH) [11] has since been adopted by many large digital archives and has become the primary mechanism for digital library interoperability in 2004. The OAI-PMH is very closely related to SOA as it adopts a Web-distributed view of individual systems, where independent components - listed on the OAI website - interact through the medium of a well-specified request/response protocol and XML-encoded messages. The OAI-PMH differs significantly from the SOA in that it is more

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 130–146, 2005.

concerned with a specific set of protocol and encoding semantics while SOA specifies only an underlying transport mechanism that could be applied to many different protocol suites. In this sense, a marriage of OAI-PMH and SOA is both possible and probable - initial experiments to verify the feasibility of this and expose possible areas of concern were carried out by Congia et al. [4].

In the interim, however, one of the reasons OAI-PMH has not as yet migrated to SOA is that the technology is not sufficiently well proven. In addition, OAI-PMH is aimed at interaction among entire systems, viewed as components of a super-system. SOA, however, is easily applied to a finer granularity, where reuse and modularity of components are crucial. To bridge this gap, and translate the core principles of OAI-PMH to fine-grained interaction among small components of a larger digital library, the Open Digital Library (ODL) project defined a suite of protocols based on the widely accepted principles of OAI-PMH, but aimed at the needs of digital library subsystem interaction [21][19]. There are some similar projects such as Dienst [10], which uses older technology, and OpenDLib [2], for which tools and reference implementations were not available for experimentation. These are discussed in detail in prior publications.

To maintain thematic consistency, the ODL protocols were designed in an object-oriented fashion, where each protocol built on a previous one, extending and overriding semantics as needed. A set of reference implementations of components were then created to provide the following services: **searching**; **browsing**; tracking of **new items**; **recommendation** by collaborative filtering; **annotation** of items as an independent service; numerical **ratings** for items in a collection; **merging** of sub-collections; and **peer review** workflow support.

A typical URL GET request to the search component, according to the ODL-Search protocol that is defined by ODL, is listed below:

```
http://www.someserver.org/cgi-bin/Search/instance1/search.pl?
  verb=ListRecords&metadataPrefix=oai_dc&
  set=odlsearch1/computer science/1/10
```

This request is for records 1-10 that match the query string “computer science”. The response is in a format very similar to that of the OAI-PMH, with records ranked according to probable relevance and one additional field to indicate the estimated total number of hits. This is the crux of the ODL-Search protocol, as implemented by the IRDB component. Other protocols and their associated component implementations use similar syntax and semantics.

The primary aim of the ODL project was to develop simple semantics for building experimental digital libraries - ODL was not meant for large scale production systems and there is no intention to standardise the protocols that were developed. Some users have noted that ODL protocols do not use SOAP/WSDL - this is largely because of their relationship to OAI-PMH and the fact that SOAP was not considered to be a W3C recommendation until mid-2003. However, the operation of ODL protocols is very much in keeping with the spirit of the SOA community and a move to SOAP/WSDL would involve only minor syntactic changes of negligible impact.

Ultimately, the purpose of the ODL framework was to serve as a testbed for experimental work. Many, if not all, of the experiments that were conducted have tested features of the SOA model applied to digital libraries. The components were analysed

and evaluated to determine how applicable a fine-grained component model is to the construction of digital libraries, and possibly expose problems and shortcomings to be addressed in future research. While the results of these experiments are generalisable to SOA, they are also indicators for the success of componentised Web-based systems in general, thereby blurring the already fuzzy line between Web-based information systems and digital libraries.

2 Experiments: Simplicity and Understandability

The first set of experiments aimed to determine if components with Web-based interfaces can be composed into complete systems with relative ease by non-specialists. Three different user communities were introduced to the underlying principles of OAI and the component architecture devised and were then led through the procedure of building a simple digital library system using the components.

2.1 OSS4LIB

The first group, at an ALA OSS4LIB workshop, provided anecdotal evidence that the component-connection approach to building DLs was feasible. The approximately 12 participants were largely technical staff associated with libraries and therefore had minimal experience with installation of software applications. They were carefully led through the process of configuring and installing multiple components and were pleasantly surprised at the ease with which custom digital libraries can be created from components. This led to a second, more controlled, experiment as detailed below.

2.2 Installation Test

The second group comprised 56 students studying digital libraries. The aim of this study was to gauge their level of understanding of OAI and ODL components and their ability to complete a component composition exercise satisfactorily. The objective was to install the following components and link them together to form a simple digital library, as illustrated also in Fig 1:

- XMLFile: a simple file-based OAI archive
- Harvester: an OAI/ODL harvester
- IRDB: an ODL search engine
- IRDB-ui: a simple user interface for IRDB

Students were given an hour-long introduction to OAI and ODL and then given detailed instructions to perform the component composition exercise. Upon completion, they were asked to fill out a questionnaire to evaluate the experience of building this system from components. Table 1 displays a summary of the responses to the core questions asked of users.

In addition to these questions, typical demographic information was collected to ascertain skills levels and exposure to the technical elements of the experiment. The

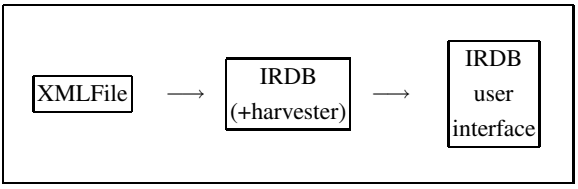


Fig. 1. Architecture of simple componentised digital library

Table 1. Summary of Responses to Component Composition Experiment

Question/Response	S. Agree	Agree	Neutral	Disagree	S. Disagree
Understand concepts of OAI-PMH.	9	38	9		
Understand concepts of ODL.	6	36	14		
Instructions were understandable.	35	20	1		
Installing components was simple.	36	18	2		
Configuration was simple.	33	21	2		
Connecting Harvester+XMLFile was simple.	28	25	3		
Connecting IRDB+XMLFile was simple.	26	25	5		
Understanding of OAI/ODL has improved.	13	25	12	6	
I will use ODL and OAI components.	12	33	10	1	

different backgrounds of participants evident from the responses to demographic questions makes it difficult to analyse these results without taking into account all of the interaction effects that result from past experience. Since OAI and ODL utilise various different Web technologies, it is non-trivial to enumerate all of the pre-requisites and determine their independent effects. It may be possible to construct an experimental model to minimise the interaction effects, but this will require finding unique participants, each with a very particular background and training. This may prove difficult because of the cutting-edge nature of Web-based technology. Taking these difficulties into account, any analysis of such an experiment cannot easily determine general trends.

Nine respondents who indicated that they did not know how Web Services worked answered affirmative when asked if they had done Web Services-related development. This may be because they interpreted the question as referring to Web-related services other than SOAP, WSDL, and UDDI, or because they had done development work without understanding the underlying standards and information model of Web Services. Either of these is consistent with the vague understanding many people have of Web Services.

Judging from the responses to the first two questions in Table 1, most participants appear to have grasped the basic concepts related to OAI and ODL. The fact that some participants were unsure indicates that an hour and 15 minutes may not be enough for a person building a digital library to learn enough about OAI and ODL. This raises the question of just how much training a person needs before being able to effectively use OAI and ODL technology (or any Web-service-related technology). Also, more of the participants were able to understand OAI rather than ODL; this is expected since ODL builds on OAI.

Most participants agreed (or strongly agreed) that the instructions were understandable. The instructions were very detailed so that even if participants did not understand one section of the exercise, they were still able to complete the rest of the steps.

Installation and configuration of individual components as well as interconnecting different components was deemed to be simple. As these are two basic concepts underlying ODL (that all services can be independent components and that systems are built by interconnecting service components), it supports the hypothesis of this experiment that ODL is simple to understand and adopt - which commutes to Web Services.

There was not much agreement about the ability of the exercise to improve the participants understanding of OAI and ODL. This can be attributed to the sheer volume of new concepts covered during the presentation and exercise. Given that approximately half of the participants had never created a CGI-based Web application before, the learning curve was quite steep. In practice, those who adopt OAI and ODL technology are usually digital library practitioners who already have experience with the construction of dynamic Web-based information systems.

In spite of all these factors, two-thirds of the participants indicated an interest in using similar components if they have a need for such services. Thus, even without a thorough understanding of the technology and other available options, the simple and reusable nature of the components seemed to appeal to participants.

Thirteen participants provided optional feedback in the survey, and these ranged from positive to somewhat skeptical. Eight of the reactions were positive, including the following comments:

- "I think the idea is very good and the approaches to build digital libraries is easy."
- "They provide a way to get up and running very quickly with a Web application."

Some participants were not sure about the workflow as indicated by the comment:

- "We have high level idea but detailed explanation will be great."

One comment in reference to the questions on simplicity of installation and config uration included:

- "I dont know; custom config might not be simple."

This summarises the notion that components should be simple enough to bootstrap a development process but still powerful enough to support a wide range of functionality. In particular, the above comment refers to the XMLFile component that is simple to install and use in its default configuration, but can be non-trivial to configure if the records are not already OAI-compatible. In such a situation, XSL transformations can

be used to translate the records into acceptable formats. However, irrespective of the complexity of configuration for a particular instance, the OAI/ODL interface to such components always is the same.

Some questions raised during the lab sessions revealed very important issues that need to be addressed in future development of ODL or related Web-based standards:

- Confusion over baseURLs
  - Some participants were confused regarding which baseURL to use in which instance. Since all URLs were similar, it was not obvious – this ought not to happen in practice with any system built on OAI, ODL or Web Services technology.
  - Entering URLs by hand resulted in many typographical errors. Ideally, such links must be made using a high-level user interface that masks complex details like URLs from the developers.
  - The user interface was sometimes connected to the wrong component. While it is possible for a user interface to Identify the service component before using it, this will be inefficient. As an alternative, user interfaces can themselves be components, with associated sanity tests applied during configuration.
- Failures during harvesting
  - Harvesting will fail if the baseURL is incorrect, but there are no obvious graceful recovery techniques. The components used in the experiment assume a catastrophic error and stop harvesting from the questionable archive pending user intervention. Better algorithms can be devised to implement exponential back-off and/or to trigger notification of the appropriate systems administrator.

2.3 Comparison Test

Finally, a third experiment was conducted to contrast the component approach to system building with the traditional monolithic system approach. 28 students in digital libraries were asked to install a system similar to the one in the previous experiment as well as a version of the Greenstone [23] system, and compare and contrast them from the perspectives of ease of use and installation.

The responses highlighted both positive and negative aspects of both systems. The majority of respondents indicated that Greenstone was easier to install, being a single cohesive package. However, it was also agreed by almost all respondents that the component approach was more flexible and powerful, and therefore applicable to a larger set of problem domains than the monolithic equivalent. There was tension between the higher degree of architectural control possible with ODL and the increase in complexity it introduced for those not wanting such control. The service-oriented approach was also preferred for its scalability, genericity and support for standards, which was not as evident in the monolithic approach. A number of respondents were undecided as to an outright preference, given that each solution had its advantages and disadvantages - leading to the conclusion that an ideal solution would capitulate on the strengths of both approaches, somehow giving end users the advantages of component-based customisation and flexibility as well as the advantages of cohesion and simplicity inherent in the non-component approach.

3 Experiments: Reusability and Extensibility

To test for reusability and extensibility, the suite of components was made available to colleagues for integration into new and existing systems. A number of digital library systems have since made use of the components, either directly, or composed/aggregated into other components. The following are a discussion of how some projects have integrated service-oriented components and protocols into their architectures.

3.1 AmericanSouth.org

AmericanSouth.org [8] is a collaborative project led by Emory University to build a central portal for scholarly resources related to the history and culture of the American South. The project was initiated as a proof-of-concept test of the metadata harvesting methodology promoted by the OAI. Thus, in order to obtain data from remote data sources, the project relies mainly on the OAI-PMH.

The requirements for a central user portal include common services such as searching and browsing. AmericanSouth.org used ODL components to assist in building a prototype of such a system. The DBUnion, IRDB and DBBrowse components were used in addition to XMLFile and other custom-written OAI data provider interfaces. Many questions about protocol syntax and component logic were raised and answered during the prototyping phase, suggesting that more documentation is needed. Alternatively, pre-configured networks of components can be assembled to avoid configuration of individual components. Both of these approaches are being investigated in the DL-in-a-Box project [14].

The production system for AmericanSouth.org still uses multiple instantiations of XMLFile but the ODL components have been replaced with the ARC search engine [13] largely because of concerns over execution speed of the IRDB search engine component. This in itself indicates the ease with which service-oriented components can be replaced in a system whose requirements change over time.

3.2 CITIDEL

CITIDEL - the Computing and Information Technology Interactive Digital Education Library [6] - is the computing segment of NSF's NSDL - the National Science, Technology, Engineering and Mathematics Digital Library [12]. CITIDEL is building a user portal to provide access to computing-related resources garnered from various sources using metadata harvesting wherever possible. This user portal is intended to support typical resource discovery services, such as searching and category-based browsing, as well as tools specific to composing educational resources, such as lesson plan editors.

From the initial stages, CITIDEL was envisioned as a componentised system, with an architecture that evolves as the requirements are refined. The initial system was designed to include multiple sources of disparate metadata and multiple services that operate over this data, where each data source and service is independent.

CITIDEL uses components from various sources. In terms of ODL, this includes the IRDB and Thread components to implement simple searching and threaded annotations, respectively. The IRDB component was modified to make more efficient use of the underlying database, but the interface was unchanged.

3.3 BICTEL/e

The BICTEL/e project, led by the Universite Catholique de Louvain, is building a distributed digital library of dissertations and e-prints within the nine French-speaking universities in Belgium. The project adopted use of OAI and ODL components to support dissertations and e-prints collections at each university and at a central site, alongside some non-ODL components.

3.4 Sub-classing

Some component implementations were created by sub-classing existing components. All of the component modules were written in object-oriented Perl, which allows for single inheritance, so this was exploited when possible. Since the DBRate and DBReview components also store the original transaction records submitted to them, they were derived from the Box component. In each case, some of the methods were overridden to provide the necessary additional functionality.

3.5 Layering: VIDI

The VIDI project [22] developed a standard interface, as an extension of the OAI protocol, to connect visualisation systems to digital libraries. A prototype of the VIDI reference implementation links into the search engine of the ETD Union Catalog [20] to obtain search results. The search engine used in the ETD Union Catalog understands the ODL-Search protocol. Thus, additional services are provided as a layer over an ODL component, without any reciprocal awareness necessary in the ODL system.

3.6 Layering: MAIDL

MAIDL, Mobile Agents In Digital Libraries [17], is a federated search system connecting together heterogeneous Web-accessible digital libraries. The project uses the "odlsearch1" syntax, as specified in the ODL-Search protocol, in order to submit queries to its search system. Further communication among the mobile agents and data providers transparently utilize the XOAI-PMH protocol [19].

4 Experiments: Performance

A number of performance tests were conducted to determine the effect of Web-based inter-component communication. The aim of these experiments was to demonstrate that the use of an SOA model would not have a significant adverse impact on systems in terms of performance. In addition, these experiments highlighted techniques that could be applied to ameliorate the effects that were noticed.

Measurements were taken for heavily loaded systems, systems that rely on multiple components to respond to requests (e.g., portals) as well as the contribution made to system latency by different layers in the architecture.

4.1 Application/Protocol Layering

The most critical of measurements looked at the effect of additional Web-application layering on the execution times of individual components of a larger system. The IRDB search engine component was used for this test because search operations take a non-trivial (and therefore measurable) amount of time and the pre-packaged component includes a direct interface to the search engine that allows bypassing of the ODL protocol layer.

For test data, a mirror of the ETD Union Archive was created and this then was harvested and indexed by an instance of the IRDB component. 7163 items were contained in this collection, each with metadata in the Dublin Core format.

The test was to execute a search for a given query. Three queries were used: "computer science testing", "machine learning", and "experiments". At most the first 1000 results were requested in each case. Each query was executed 100 times by a script to minimise the effect of the script on the overall performance. The first run of each experiment was discarded to minimise disk access penalties, and an average of the next 5 runs was taken in each case.

Six runs were made for each query:

1. Executing lynx to submit a ListIdentifiers query through the Web server interface.
2. Executing wget to submit a ListIdentifiers query through the Web server interface.
3. Using custom-written HTTP socket code to submit a ListIdentifiers query through the Web server interface.
4. Executing the search script directly from the command-line, thereby bypassing the Web server.
5. Executing testsearch.pl to bypass both the Web server and the ODL layer.
6. Using direct API calls to the IR engine, without spawning a copy of testsearch.pl in each iteration.

The time was measured as the "wall-clock time" reported by the bash utility program "time" from the time a run started to the time it ended. The script that ran the experiment controlled the number of iterations (100, in this case) and executed the appropriate code in each of the 6 cases above. In each case, the output was completely collected and then immediately discarded - thus, each iteration contributed the complete time between submitting a request and obtaining the last byte of the associated response, hereafter referred to as the execution time.

It was noticable from the measured times that execution time increases as more layers are introduced into the component. This increase is not always a large proportion of the total time, but the difference between Test-1 and Test-6 is significant. The time differences between pairs of consecutive tests is indicated in Table 2.

Test-1, Test-2 and Test-3 illustrate the differences in times due to the use of different HTTP clients. In Test-1, the fully-featured text-mode Web browser lynx was used. In Test-2, wget was used instead, and the performance improved because wget is a smaller application that just downloads files. Test-3 avoided the overhead of spawning an external client application altogether by using custom-written network socket routines to connect to the server and retrieve responses to requests. The differences are only slight but there is a consistent decrease for all queries.

Table 2. Time differences between pairs of consecutive tests

Query	Test1-2	Test2-3	Test3-4	Test4-5	Test5-6
"computer science testing"	4.52	1.04	0.67	0.33	8.57
"machine learning"	3.72	0.63	0.58	0.22	10.62
"experiments"	4.26	0.94	0.35	0.66	8.53

The difference between Test-3 and Test-4 is due to the effect of requests and responses passing through the HTTP client and the Web server. While no processes were spawned at the client side in Test-3, a process was still spawned by the Web server to handle each request at the back-end. This script was run directly in Test-4, so the difference in time is due solely to the request being routed through the Web server. This difference is small, so it suggests that the Web server does not itself contribute much to the total execution time.

The difference between Test-4 and Test-5 is due to the ODL-Search software layer that handles the marshalling and unmarshalling of CGI parameters and the generation of XML responses from the raw list of identifiers returned by the IR engine. This difference is also small, indicating that the additional work done by the ODL layer does not contribute much to the total time of execution.

The difference between Test-5 and Test-6 is due to the spawning of a new process each time the IRDB component is used. This difference is substantial and indicates that process startup is a major component of the total execution time.

In general, the execution times for the IRDB component (as representative of ODL components in general) were much higher than the execution times for direct API calls. However, this difference in execution time is due largely to the spawning of new processes for each request. The ODL layer and the Web server contribute only a small amount to the total increase in execution time.

4.2 Nested Requests and Persistence

In order to avoid duplication of metadata entries, some of the ODL components (such as IRDB) do not store redundant copies - instead, every time a record is needed, it is fetched from the source archive by means of further internally-generated requests to the Web-based ODL interface. This procedure is hereafter referred to as a nested request.

An initial experiment compared nested requests that invoked Web-based services 10 times for each query processed with requests that avoided the use of Web-based services in favour of internal APIs. This experiment did not make use of any performance optimisation technology. The results confirmed that the response time is roughly proportional to the number of nested requests and that process startup time is the most significant contributor to the delays.

To deal with the process startup time, the SpeedyCGI package [1] was installed and components were configured to use it instead of regular CGI. The effect of this change was then tested and compared against the case where no optimisations are used. SpeedyCGI is a tool that speeds up access to Perl scripts without modification of the script or

the Web server. Instead of running Perl with each invokation of a script, the script is run by a relatively small SpeedCGI front-end program that connects to a memory-resident Perl back-end, creating the back-end process if necessary. Thus, the process startup time is determined by the execution speed of the front-end script rather than the Perl interpreter. The objectives of this study were to calculate the response times for single and nested requests, both with and without using SpeedyCGI, and to compare SpeedyCGI usage with the fastest approach thus far, that of directly utilising a programming API.

The IRDB search engine component was used for this test. The test was performed in the same experimental environment as for the Layering experiment. The test was to execute a search for a given query. Three queries were used: "computer science testing", "machine learning" and "experiments". The first run of each experiment was discarded and an average of the next 5 runs was taken in each case. For the first part of the experiment, all requests were submitted by executing wget, thus involving the Web server and the ODL interface in generation of the response. At most the first 10 results were requested in each case. Each query was executed 10 times by a script. Four runs were made for each query as follows:

1. By submitting ListIdentifiers (LI).
2. By submitting ListRecords (LR).
3. By submitting ListIdentifiers, where the components use SpeedyCGI (LIS).
4. By submitting ListRecords, where the components use SpeedyCGI (LRS).

Table 3 displays the comparisons from the first part of the experiment using SpeedyCGI and not, for both ListIdentifiers and ListRecords requests submitted to IRDB.

Table 3. Regular CGI vs. SpeedyCGI speed comparisons (seconds)

Query	LI	LR	LIS	LRS
"computer science testing"	1.67	16.50	0.44	2.22
"machine learning"	1.57	16.34	0.33	2.04
"experiments"	1.55	16.35	0.31	2.06

For the second part of the experiment, at most 1000 results were requested and each query was executed 100 times. The requests were submitted by executing wget, thus involving the Web server and the ODL interface in generation of the response. A single ListIdentifiers run was conducted for each query, and this was contrasted with the data obtained during the direct API measurements taken in the Layering experiment.

Table 4 displays the comparisons from the second part of the experiment using SpeedyCGI and comparing this to the previous measurements for the case with direct API use.

Results from the first part of the experiment indicate that there is a significant improvement in execution speeds for both ListIdentifiers (single requests) and ListRecords (nested requests) when SpeedyCGI is used. This is largely due to the elimination of the need to spawn new processes to handle each request to the Web server. The second set

Table 4. SpeedyCGI vs. direct API speed comparisons

Query	SpeedyCGI	API
"computer science testing"	40.27	39.70
"machine learning"	16.61	15.81
"experiments"	32.39	32.78

of results indicate that there is very little difference in execution times between using direct API calls and using a fully layered IRDB component when SpeedyCGI is used.

This is a significant result for the applicability of SOA in situations where performance is an issue. Generalising, these experiments confirm that there is little cause for concern, performance-wise, if Web technology is chosen wisely – for example, using persistent Web applications such as servlets for Java applications or SpeedyCGI for Perl applications. Other technologies exist for these and other languages and some of these have been evaluated in the ODL [18] and X-Switch [15] projects.

4.3 System Load

Under real-world conditions, response times can be drastically different as situations vary. The aim of this experiment was to assess the ability of a component to perform acceptably under high loads. To assess this, a server was artificially loaded and then the response times of typical requests were measured under different load conditions.

The Box component was used for this test because it has very little component logic and therefore provides lower bounds for execution speed that are indicative of the ODL component architecture and not the component logic. The component was installed in the same server environment used in the Layering experiment. A second identical machine was used to simulate client machines by running multiple processes, each of which submitted ListRecords requests to the server in a continuous loop. The server was primed with 100 dummy records for this purpose. The test was to submit GetRecord and PutRecord requests to the local server. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case. The experiment was then repeated using SpeedyCGI for the Box component.

Table 5 lists the average execution times for GetRecord and PutRecord operations under load conditions generated by 5, 10 and 50 simultaneous processes.

Table 5. Average execution times under different load conditions

Operation	5 clients	10 clients	50 clients
PutRecord	1.04	2.39	9.31
GetRecord	1.39	2.06	11.27

Table 6 lists the average execution times for GetRecord and PutRecord operations under load conditions generated by 5, 10 and 50 simultaneous processes, when the Box component uses SpeedyCGI.

Table 6. Average execution times when using SpeedyCGI

Operation	5 clients	10 clients	50 clients
PutRecord	0.691	0.702	2.146
GetRecord	0.449	0.316	1.801

There is variability in execution time because of the non-deterministic nature of client-server synchronisation and process startup. It is apparent, however, that the time taken to respond to a request increases as the load on the server increases. Using the persistent script mechanism of SpeedyCGI results in a reduction of the execution time as compared to the case without SpeedyCGI, but there is still an increase with increasing load, as expected.

In both cases, a high load on the server causes an increase in execution time for component interaction. The SpeedyCGI module, as representative of persistent script tools, helps to minimise this effect. Ultimately, however, ODL components are Web applications and the only way to get better performance for a heavily loaded Web server is to use more and/or faster servers. This suggests that a server farm or component farm based on cluster computing or grid technologies might be one possible solution for digital libraries that require a lot of processing power for computations in either the pre-processing (e.g., indexing), maintenance (e.g., reharvesting) or dissemination of data (e.g., online querying) phases.

4.4 User Interfaces

While inter-component communication speeds are important to system designers, it is the speed of the user interface that matters the most to users. To test this, requests were submitted to a mirror of CSTC (Computer Science Teaching Centre), based completely on ODL components, to determine its effectiveness. The objective was to submit requests to the CSTC interface, simulating typical user behavior, and then measure the response time.

The client machine was a 2Ghz Pentium 4 PC with 1GB of RAM running a pre-installed version of Red Hat Linux v7.3. The server used was a non-dedicated 600MHz Pentium 3 with 256MB RAM running Red Hat v6.2. The Web server was Apache v1.3.12 and data for all components was stored in a mySQL v3.23.39 database. All components used the SpeedyCGI tool to remain persistent in memory.

The first test involved simulating a browse operation. The second test involved simulating the viewing of metadata for a single resource. In both instances, the test was repeated 10 times per invocation of the test script. The test script was executed 5 times and the average of these was computed.

Table 7 displays the user interface execution times for the operations tested.

The browsing operation required 1 request that was submitted to the DBBrowse component, as well as 5 nested requests sent to the DBUnion component in order to fetch the metadata, resulting in a total of 6 requests. The display operation required 1 request for the metadata, 1 for the rating, 4 for the recommendation and 3 for the feedback mechanism - resulting in a total of 9 requests. The time taken is not simply

Table 7. Execution times for user interface actions

Action	t (Time taken for 10 requests)	t / 10
Browse first screen of items	9.28	0.93
Display metadata for first item	9.81	0.98

proportional to these request counts because different components contribute varying amounts to the total execution time. However, as indicated in Table 7, the total response time in both instances is less than 1 second for data transfer.

In general, the time taken to generate user interface pages is reasonably small for the new CSTC system, running on a production server. In combination with a higher load (as tested in the previous experiment), it can be expected that the response time will increase further and this is additional motivation for a generic SOA-based component grid/cluster where services/components can be replicated as the needs of the system change over time.

5 Conclusions

The Service Oriented Architecture is still a fairly new concept in DL systems, with most systems supporting one or two external interfaces, for example OAI-PMH. This work has investigated the applicability of SOA as a fundamental architecture within the system, an analysis of which has demonstrated its feasibility according to multiple criteria, while exposing issues that need to be considered in future designs. In summary, this works provides evidence in support of the following assertions:

- From a programmer's perspective, SOA is simple to understand, adopt and use.
- Components in an SOA can easily be integrated into or built upon by external systems.
- Performance penalties from additional layering can be managed.
- Performance issues do not have to permeate the architectural model - these can be handled as external optimisations.
- There are no inherent architectural restrictions that prevent the modelling of specific digital library systems.

6 Future Work

The most important aspect highlighted by past experiments was the need for better and simpler management of components, so that the complexity of deconstructing a monolithic system into service-oriented components did not fundamentally increase the complexity of overall system management. To this end, the ongoing "Flexible Digital Libraries" project is investigating how external interfaces can be defined for remote management of components, thus enabling automatic aggregration and configuration of components by installation managers and real-time component management systems. The first of such systems to be built, BLOX [16], allows a user to build a system

visually using instances of Web-accessible components residing on remote machines. Experiments conducted with BLOX by Eyambe [5] have demonstrated that users generally prefer high level programming of digital library systems by assembling building blocks visually as opposed to the traditional low-level programming API approach.

In addition, there is an ongoing project looking into how components designed with well-defined administrative interfaces can be packaged and redeployed, thus bridging the gap between components and monolithic systems from a system installation perspective. Ideally, a future digital library or online information system will be designed on a canvas and a package will be generated for distribution based on a concise specification of the system. Then distribution and installation will be as simple as possible for the ordinary user; but the administrator wishing to customise the system can still modify the specification of the system at a later date to upgrade or modify the components in use.

At the same time, some effort needs to go into how services are orchestrated and composed/aggregrated at a higher level to perform useful functions needed by users. The WS-Flow and WS-Choreography activities are useful starting points but more investigation is needed into their suitability as integration mechanisms between "front-end" and "back-end" systems.

User interface components are being developed in a related project, to address the common perception that information management components are usually only part of "back-end" systems. Initial experiments with the BLOX system have demonstrated that a user interface can be plugged into the system as easily as any other component.

Current directions for this research include extension of the core architecture to allow migration and replication of components to support "component farms" as a replacement for "server farms", adopting notions from the cluster and grid computing paradigms, where services are needs-based and location-independent. Thus, the component approach to building digital libraries will demonstrate scalability in addition to flexibility.

Eventually, it is hoped that SOA will form an integral part of an architecture for flexible online information management systems, with all the advantages of monolithic systems as well as component-based systems, and with the ability to meet the needs of both the resource-constrained trivially-small system and the scalability requirements of large-scale public knowledge bases - an architecture that can be easily and readily adopted by future generations of systems such as DSpace, EPrints and Greenstone.

References

1. SpeedyCGI, 2005. Website `http://daemoninc.com/speedycgi/`.
2. Donatella Castelli and Pasquale Pagano. OpenDLib: A Digital Library Service System. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 6th European Conference*, number 2458 in Lecture Notes in Computer Science, pages 292–308, Rome, Italy, September 2002.
3. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, 2001. Available `http://www.w3.org/TR/wsdl`.

4. Sergio Congia, Michael Gaylord, Bhavik Merchant, and Hussein Suleman. Applying SOAP to OAI-PMH. In R. Heery and L. Lyon, editors, *Research and Advanced Technology for Digital Libraries, Proceedings of the 8th European Conference*, volume 3232 of *Lecture Notes in Computer Science*, pages 411–420, Bath, UK, September 2004.
5. Linda K. Eyambe and Hussein Suleman. A Digital Library Component Assembly Environment. In G. Marsden, P. Kotzé, and A. Adesina-Ojo, editors, *Proceedings of SAICSIT 2004*, pages 15–22, Stellenbosch, South Africa, October 2004.
6. Edward A. Fox, Deborah Knox, Lillian Cassel, John A. N. Lee, Manuel Pérez-Qui nones, John Impagliazzo, and C. Lee Giles. CITIDEL: Computing and Information Technology Interactive Digital Educational Library, 2005. Website `http://www.citidel.org`.
7. M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielson. SOAP Version 1.2 Part 1: Messaging Framework and Part 2: Adjuncts. Technical report, W3C, June 2003. Available `http://www.w3.org/TR/2003/REC-soap12-part1-2003-0624/` and `http://www.w3.org/TR/2003/REC-soap12-part2-2003-0624/`.
8. M. Halbert. AmericanSouth.org, 2005. Website `http://www.americansouth.org`.
9. Ariba Inc., IBM, and Microsoft. UDDI Technical White Paper. Technical report, September 2000. Available `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.
10. C. Lagoze and J. R. Davis. Dienst - An Architecture for Distributed Document Libraries. *Communications of the ACM*, 38(4):47, 1995.
11. Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. The Open Archives Initiative Protocol for Metadata Harvesting Version 2.0. Technical report, June 2002. Available `http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm`.
12. Carl Lagoze, Walter Hoehn, David Millman, William Arms, Stoney Gan, Dianne Hillmann, Christopher Ingram, Dean Krafft, Richard Marisa, Jon Phipps, John Saylor, Carol Terrizzi, James Allan, Sergio Guzman-Lara, and Tom Kalt. Core Services in the Architecture of the National Science Digital Library (NSDL). In *Proceedings of Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 201–209, Portland, OR, USA, July 2002.
13. Xiaoming Liu, Kurt Maly, Mohammad Zubair, and Michael L. Nelson. Arc: an OAI service provider for cross-archive searching. In *Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 65–66, Roanoke, VA, USA, June 2001.
14. Ming Luo. Digital Libraries in a Box, 2005. Website `http://dlbox.nudl.org`.
15. Andrew Maunder and Reinhardt van Rooyen. Universal Web Server: The X-Switch System. Technical Report CS04-20-00, Department of Computer Science, University of Cape Town, 2004. Available `http://pubs.cs.uct.ac.za/archive/00000157/`.
16. David Moore, Stephen Emslie, and Hussein Suleman. BLOX: Visual Digital Library Building. Technical Report CS03-20-00, Department of Computer Science, University of Cape Town, 2003. Available `http://pubs.cs.uct.ac.za/archive/00000075/`.
17. Nava Mu noz and Sandra Edith. Federación de Bibliotecas Digitales utilizando Agentes Móviles (Digital Libraries Federation using Mobile Agents). Master's thesis, Universidad de las Américas, Puebla, Mexico, 2002.
18. H. Suleman. *Open Digital Libraries*. PhD thesis, Virginia Tech, Blacksburg, VA, USA, December 2002. Available `http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/`.
19. H. Suleman and E. A. Fox. Designing Protocols in Support of Digital Library Componentization. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 6th European Conference*, number 2458 in Lecture Notes in Computer Science, pages 568–582, Rome, Italy, September 2002.

20. H. Suleman and E. A. Fox. Towards Universal Accessibility of ETDs: Building the NDLTD Union Archive. In *Fifth International Symposium on Electronic Theses and Dissertations (ETD2002)*, Provo, Utah, USA, May 2002.
21. Hussein Suleman and Edward A. Fox. A Framework for Building Open Digital Libraries. *D-Lib Magazine*, 7(12), December 2001. Available `http://www.dlib.org/dlib/`december01/suleman/12suleman.html.
22. J. Wang. A Lightweight Protocol Between Visualization Tools and Digital Libraries. Master's thesis, Virginia Tech, Blacksburg, VA, USA, 2002.
23. I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge. Greenstone: A Comprehensive Open-Source Digital Library Software System. In *Proceedings of Fifth ACM Conference on Digital Libraries*, pages 113–121, San Antonio, Texas, USA, June 2000. ACM Press.
24. J. Yang. Web Service Componentization. *Communications of the ACM*, 46(10):35–40, October 2003.

A System Architecture as a Support to a Flexible Annotation Service

Maristella Agosti and Nicola Ferro

Department of Information Engineering – University of Padua,
Via Gradenigo, 6/b – 35131 Padova – Italy
{maristella.agosti, nicola.ferro}@unipd.it

Abstract. Digital Library Management Systems are systems that are able to manage collections of digital documents that form Digital Libraries and Digital Archives, and they are currently in a state of evolution. Today, most of the times they are simply places where information resources can be stored and made available, whereas for tomorrow they are becoming an integrated part of the way the user works. To cooperate towards reaching this new type of system, a digital library management system must become a tool that constitutes an active part of the intellectual production process.

Annotations are effective means in order to enable an effective interaction between users and digital library management systems, since they are a very well-established practice and are widely used. Annotations are not only a way of explaining and enriching an information resource with personal observations, but also a means of transmitting and sharing ideas in order to improve collaborative work practices. Furthermore, annotations represent a bridge between reading and writing, that facilitates the user's first approach when they begin dealing with an information resource. Thus, a service able to support annotation capabilities of collection of digital documents can be appealing to the user's needs.

This paper presents the main features of a flexible system capable of managing annotations in an automatic way in order to support users and their annotative practices. Indeed, a flexible architecture allows the design of a system with a widespread usage, so that users can benefit from its functionalities without limitations due to the architecture of a particular system. We named this system *Flexible Annotation Service Tool* and this paper is devoted to introduce most relevant design choices and characteristics of it.

1 Introduction

Nowadays, the notion of isolated information resources or applications is increasingly being replaced by a distributed and networked environment, where there is almost no distinction between local and remote information resources and applications. Indeed, a wide range of new technologies allow us to envision ubiquitous and pervasive access to information resources and applications. A wide range of wired and wireless technologies make it possible to offer almost ubiquitous connectivity; examples of such technologies are *Local Area Networks (LANs)*, *Wireless LANs (WLANs)*, *Asymmetric Digital Subscriber Line (ADSL)* and other broadband connections, *Third Generation Mobile System (3G)* networks as *Universal Mobile Telecommunication System*

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 147–166, 2005.

(UMTS) networks. Moreover, a variety of devices, that range from desktop computers to *Personal Digital Assistants (PDAs)*, mobile phones, and other handheld devices [1], and a series of emerging architectural paradigms, such as *Web Services (WS)*, *Peer-To-Peer (P2P)* and Grid architectures, are now available and allow us to design and develop services and systems that are more and more user-centered.

In particular, *Digital Librarys (DLs)*, as information resources, and *Digital Library Management Systems (DLMSs)*, that manage DLs, are currently in a state of evolution: today they are simply places where information resources can be stored and made available, whereas for tomorrow they will become an integrated part of the way the user works. For example, instead of simply downloading a paper and then working on a printed version, a user will be able to work directly with the paper by means of the tools provided by the DLMS and share their work with colleagues. This way, the user's intellectual work and the information resources provided by the DLMS can be merged together in order to constitute a single working context. Thus, the DL is no longer perceived as something external to the intellectual production process or as a mere consulting tool, but as an intrinsic and active part of the intellectual production process.

Annotations are effective means in order to enable this new paradigm of interaction between users and DLMSs, since they are a very well-established practice and widely used. Annotations are not only a way of explaining and enriching an information resource with personal observations, but also a means of transmitting and sharing ideas in order to improve collaborative work practices. Furthermore, annotations represent a bridge between reading and writing, that facilitates the user's first approach when they begin dealing with an information resource; thus, a DLMS offering annotation capabilities can be appealing to the user's needs. Finally, annotations allow users to naturally merge personal contents with the information resources provided by the DLMS, making it possible to embody the paradigm of interaction between users and DLs which has been envisaged above. We aim at designing a system capable of managing annotations in an automatic way in order to support users and their annotative practices.

2 Annotations

Over past years a lot of research work regarding annotations has been done [2]. All of this research work has led to different viewpoints about what an annotation is. These viewpoints are discussed in the following.

2.1 Metadata

Annotations can be considered as additional data which concern an existing content, that is annotations are metadata, because they clarify in some way the properties and the semantics of the annotated content.

[3,4] propose a data model for the composition and metadata management of documents in a distributed setting, such as a DLMS. They allow the creation of *composite documents*, that are made up of either composite documents or *atomic documents*, that can be any piece of material uniquely identifiable. A set of annotations is automatically

associated to each composite document starting from the annotations of its composing atomic documents, where [3,4] interpret annotations as terms taken from a controlled vocabulary or taxonomy to which all authors adhere.

The Annotea[1] project developed by the *World Wide Web Consortium (W3C)* [5] considers annotations as metadata and interprets them as the first step in creating an infrastructure that will handle and associate metadata with content towards the Semantic Web. Annotea uses *Resource Description Framework (RDF)* and *eXtensible Markup Language (XML)* for describing annotations as metadata and XPointer for locating the annotations in the annotated document. Annotea employs a client-server architecture based on *HyperText Transfer Protocol (HTTP)*, where annotations reside in dedicated servers and a specialized browser is capable of retrieving them upon request, when visiting a Web page.

Annotations are used also in the context of *DataBase Management Systems (DBMSs)* and, in particular, in the case of *curated databases* and *scientific databases*. SWISS-PROT[2] is a curated protein sequence database, which strives to provide a high level of annotation, such as the description of the function of a protein, its domains structure, and so on. In this case, the annotations are embedded in the database and merged with the annotated content. BIODAS[3] provides a *Distributed Annotation System (DAS)*, that is a Web–based server system for sharing lists of annotations across a certain segment of the genome. In this case, annotations are not mixed together with the content they annotate, but they are separated from it. In this context, [6] proposes an archiving technique in order to manage and archive different versions of such kind of databases, as time moves on. [6] exploits the hierarchical structure of scientific data in order to represent the content and the different versions of the database with a tree structure, and attaches annotations to the nodes of the tree, annotations that contain time–stamp and key information about the underlying data structure. Thus, these annotations are metadata about the database itself. These annotations are different from the annotations contained in the database, that are metadata about genome sequences.

[7,8] investigate the usage of annotations with respect to the *data provenance* problem, which is the description of the origins of a piece of data and the process by which it arrived in a database. Data provenance is a relevant issue in the field of curated and scientific databases, such as genome databases, because experts provide corrections and annotations to the original data, as time moves on. It is now clear that data provenance is essential to any user interested in the accuracy and timeliness of the data, especially for understanding the source of errors in data and for carrying annotations through database queries. [9] proposes and implements an extension to a relational DBMS and an extension to *Structured Query Language (SQL)*, called *propagate SQL (pSQL)*, which provides a clause for propagating annotations to tuples through queries. [9] intends annotations to be information about data such as provenance, comments, or other types of metadata; [9] envisages the following applications of annotations in DBMS: tracing the provenance and flow of data, reporting errors or remarks about a piece of data, and describing the quality or the security level of a piece of data.

[1] `http://www.w3.org/2001/Annotea/`
[2] `http://www.expasy.org/sprot/`
[3] `http://biodas.org/`

2.2 Contents

Annotations are additional contents which concern an existing content [2]; indeed, they increase existing content by providing an additional layer of content that elucidates and explains the existing one. This viewpoint about annotations entails an intrinsic dualism between annotation as content enrichment and annotation as stand-alone document [10]:

- *annotation as content enrichment*: in this view annotations are considered as mere additional content regarding an existing document and so they are not autonomous entities but in fact they rely an already existing information resource in order to justify their existence;
- *annotation as stand-alone document*: in this view annotations are considered as real documents and are autonomous entities that maintain some sort of connection with an existing document.

This twofold nature of the annotation is clear if we think about the process of studying a document: firstly, we can start annotating some interesting passages that require an in depth investigation, which is an annotation as content enrichment; then we can reconsider and collect our annotations and we can use them as a starting point for a new document, covering the points we would like to explain better which is an annotation as a stand-alone document. In this case the annotation process can be seen as an informal, unstructured elaboration that could lead to a rethinking of the annotated document and to the creation of a new one. Systems like COLLATE [11,12] or IPSA [13,14,15] support this task through annotations.

Different layers of annotations can coexist on the same document: a private layer of annotations accessible only by the annotations authors themselves, a collective layer of annotations, shared by a team of people, and finally a public layer of annotations, accessible to all the users of the digital library. In this way, user communities can benefit from different views of the information resources managed by the DLMS [16,17]. A DLMS can encourage cooperative work practices, enabling the sharing of documents and annotations, also with the aid of special devices, such as XLibris [18]. Finally, as suggested in [19,20], searching, reading and annotating a DL can be done together with other activities, for example working with colleagues. This may also occur in a mobile context, where merging content and wireless communication can foster ubiquitous access to DLMSs, improving well established cooperative practices of work and exploiting physical and digital resources. The wireless context and the small form factor of handheld devices challenge our technical horizons for information management and access and require specialized solutions in order to overcome the constraints imposed by such kinds of devices, as analysed in [1].

As a further example, *Multimedia Annotation of Digital Content Over the Web (MADCOW)* is based on a client-server architecture as Annotea is. Servers are repositories of annotations to which different client can connect, while the client is a plug-in for a standard Web browser [21,22]. MADCOW employs HTTP in order to annotate Web resources and allows both private and public annotations. Moreover, it allows different pre-established types of annotations, such as explanation, comment, question, solution, summary, and so on.

2.3 Hypertext

Annotations allow the creation of new relationships among existing contents, by means of links that connect annotations together and with existing content. In this sense we can consider that existing content and annotations constitute a hypertext, according to the definition of hypertext provided in [23]. This hypertext can be exploited not only for providing alternative navigation and browsing capabilities, but also for offering advanced search functionalities. Furthermore, [24] considers annotations as a natural way of creating and growing hypertexts that connect information resources in a DLMS by actively engaging users. Finally, the hypertext existing between information resources and annotations enables different annotation configurations, that are *threads of annotations*, i.e. an annotation made in response to another annotation, and *sets of annotation*, i.e. a bundle of annotations on the same passage of text [10,25].

2.4 Dialog Acts

Annotations are part of a discourse with an existing content. For example, [11,26] consider annotations as the document context, intended as the context of the collaborative discourse in which the document is placed. Also [27] agree, to some extent, with this viewpoint about annotations. Indeed, they interpret annotations as a means that allow a "two way exchange of ideas between the authors of the documents and the documents users".

3 Architectural Approach

Annotations have a wide range of usages in different *Information Management Systems (IMSs)*, ranging from DBMSs to DLMSs and corresponding to the different viewpoints about annotations, introduced in Section 2. Annotations are a key technology for actively involving users with an IMS and this technology should be available for each IMS employed by the user. Indeed, the user should benefit from a uniform way of interaction with annotation functionalities, without the need of changing their annotative practices only because a user works with different IMSs. Furthermore, annotations create an hypertext that allows users to merge their personal content with the information resources provided by diverse IMSs, according to the scenario envisaged in Section 1: this hypertext can span and cross the boundaries of a single IMS, if users need to interact with diverse IMSs. The possibility of having a hypertext that spans the boundaries of different IMSs is quite innovative because up to now such hypertext is usually confined within the boundaries of a single IMS. Moreover, IMSs do not usually offer hypertext management functionalities; for example, DLMSs do not normally have a hypertext connecting information resources with each other. Thus, annotations can be a way of associating a hypertext to a DL in order to enable an active and dynamic usage of information resources [25]. Finally, there are many new emerging architectural paradigms, such as P2P or WS architectures, that have to be taken into account.

Thus, our architectural approach is based on *flexibility*, because we need to adopt an architecture which is flexible enough to support both various architectural paradigms

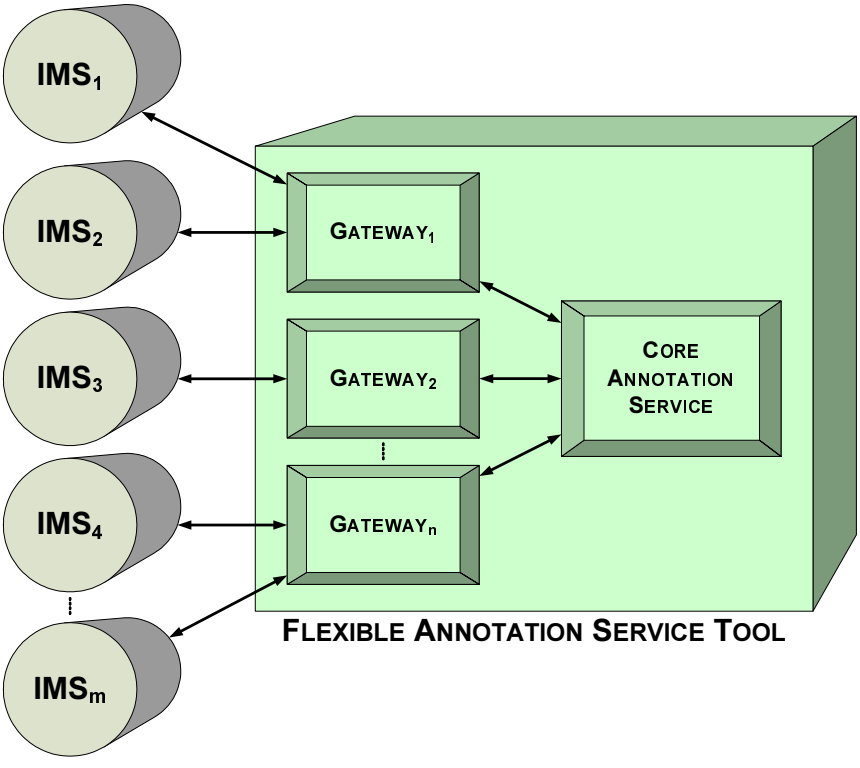


Fig. 1. Overview of the architecture of FAST with respect to different IMSs

and a wide range of different IMSs. Indeed, a flexible architecture allows the design of a system with a widespread usage, so that users can benefit from its functionalities without limitations due to the architecture of a particular IMS. Since our target system is flexible, we named it *Flexible Annotation Service Tool (FAST)*. In order to fulfil the requirements introduced above, our architectural approach is twofold:

1. to make FAST a stand-alone system, i.e. FAST is not part of any particular IMS;
2. to separate the core functionalities of the annotation service, from the functionalities needed to integrate it into different IMSs.

Figure 1 shows the general architecture of the FAST system and its integration with different IMSs: the *Core Annotation Service (CAS)* is able to interact with different gateways, that are specialised for integrating the CAS into different IMSs. From the standpoint of an IMS the FAST system acts like any other distributed service of the IMS, even if it is actually made up of two distinct modules, the gateway and the CAS; on the other hand, the FAST system can be made available for another IMS by creating a new gateway. Note that the additional layer introduced by the gateway allows the integration of the CAS also with legacy systems, that may benefit from the availability of annotation functionalities. The choice of making FAST a stand-alone system

is coherent with the approach adopted by different systems: for example, Annotea by the W3C, MADCOW, and BIODAS rely on stand-alone servers, that store and manage annotations separated from the annotated objects. On the other hand, the choice of separating the core functionalities of the annotation service, from the functionalities needed to integrate it into the different IMSs is quite new. In fact, you will not be able to find an architecture like this in the literature about annotation systems, to the best of our knowledge.

As a consequence of this architectural choice, it is worth pointing out that the FAST system knows everything about annotations, however it cannot do any assumption regarding the information resources provided by the IMS, being that it needs to cooperate with different IMSs.

This situation is very different from what is commonly found today. For example, both Annotea and MADCOW are stand-alone systems but they are targeted to work with Web pages. Indeed, they assume that the annotated object has a structured compliant with *HyperText Markup Language (HTML)*, as an example, and that they can use HTTP to transport annotations. On the contrary, FAST cannot assume that it is dealing with either HTML documents or the HTTP protocol, but it has to avoid any constraints concerning both the annotated information resource and the available protocols. The only assumption about information resources that FAST can make is that each information resource is uniquely identified by a *handle*, which is a name assigned to an information resource in order to identify and facilitate the referencing to it. This assumption is coherent with the assumption made by [3,4] who refer to and compose documents only by identifiers and annotate them with metadata from a taxonomy of terms.

Over the past years, various syntaxes, mechanisms, and systems have been developed in order to provide handles or identifiers for information resources. The mechanisms and the standards discussed in the following are all suitable to be used as handles, according to the assumption made above.

URI - URN - URL. The *Internet Engineering Task Force (IETF)*[4] defines: *Uniform Resource Identifier (URI)*, *Uniform Resource Name (URN)*, and *Uniform Resource Locator (URL)*. An URI [28] is a compact string of characters for identifying an abstract or physical resource. URIs are characterized by the following definitions:

- *uniform*: it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers;
- *resource*: a resource can be anything that has identity. Not all resources are network "retrievable"; e.g., human beings, corporations, and library books can be considered

[4] `http://www.ietf.org/`

resources as well. The resource is the conceptual mapping to an entity or set of entities. Thus, the resource does not necessarily have to correspond to the mapped entity at any given time, instead it is the conceptual mapping itself. In conclusion, a resource can remain constant even when its content—the entities to which it currently corresponds—changes over time, provided that the conceptual mapping is not changed in the process;
- *identifier*: an identifier is an object that can act as a reference to something that has an identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

The term URL refers to the subset of URIs that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term URN refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

DOI. The *International DOI Foundation (IDF)*[5] defines the *Digital Object Identifier (DOI)*, which is an *actionable identifier* for intellectual property on the Internet. Firstly, the IDF defines an identifier from different viewpoints:

(1) an identifier is *an unambiguous string or "label" that references an entity*. An example of such an identifier is the *International Standard Book Number (ISBN)*[6], which is a unique number assigned to a title or edition of a book or other monographic publication (serial publications excluded) published or produced by a specific publisher or producer;
(2) an identifier is *a numbering scheme*, such as a formal standard, an industrial convention, or an arbitrary internal system. This numbering scheme provides a consistent syntax for generating individual labels or identifiers, as stated in (1), that denote and distinguish separate members of a class of entities; we can still use the ISBN, as an example. The intention is establishing a one–to–one correspondence between the members of a set of labels (numbers), and the members of the set counted and labelled. An important point is that the resulting number is simply a label string, but it does not create a string that is "actionable" in a digital or physical environment without further steps being taken;
(3) an identifier is *an infrastructure specification*: a syntax by which any identifier as stated in (1) can be expressed in a suitable form for use with a specific infrastructure, without necessarily specifying a working mechanism; an example of such an identifier is the URI. This is sometimes known as creating an "actionable identifier" which means that in the context of that particular piece of infrastructure, the label can now be used to perform some action;
(4) an identifier is *a system for implementing labels (identifiers as stated in (1)) through a numbering scheme (identifiers as stated in (2)) in an infrastructure using a specification (identifiers as stated in (3)) and management policies*. This sense of "identifier" denotes a fully implemented identification mechanism that includes the ability

[5] `http://www.doi.org/`
[6] `http://www.isbn-international.org/`

to incorporate labels, conforms to an infrastructure specification, and adds to these practical tools for the implementation such as registration processes, structured interoperable metadata, and an administrative mechanism.

The DOI is a system which provides a mechanism to interoperably identify and exchange intellectual property in the digital environment. It is an identifier as stated in (4) above. One of the components is a syntax specification (identifier as stated in (2)). DOI conforms to a URI (identifier as stated in (3)) specification. It provides an extensible framework for managing intellectual content based on proven standards of digital object architecture and intellectual property management, and it is an open system based on non-proprietary standards.

OpenURL. The *National Information Standards Organization (NISO)* Committee AX[7] defines the OpenURL framework, which aims at standardizing the construction of "packages of information" and the methods by which they may be transported over networks. The intended recipients of these packages are networked service providers that deliver context-sensitive services. To enable such services, each package describes not only the resource for which services are needed, but also the network context of a reference to the resource in question. Thus, OpenURL is a standard syntax for transporting information (metadata and identifiers) about one or multiple resources within URLs, i.e. it provides a syntax for encoding metadata and identifiers, limited to the world of URLs.

PURL. The *Online Computer Library Center (OCLC)* defines the *Persistent URL (PURL)*[8], which is an URL from a functional standpoint. However, instead of pointing directly to the location of an Internet resource, a PURL points to an intermediate resolution service, that associates the PURL with the actual URL and returns that URL to the client as a standard HTTP redirect. The client can then complete the URL transaction in the normal fashion.

The *PURL-based Object Identifier (POI)*[9] is a simple specification for resource identifiers based on the PURL system, closely related to the use of the *Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)* defined by the *Open Archives Initiative (OAI)*[10]. The POI is a relatively persistent identifier for resources that are described by metadata "items" in OAI-compliant repositories. Where this is the case, POIs are not explicitly assigned to resources – a POI exists implicitly because an OAI "item" associated with the resource is made available in an OAI-compliant repository. However, POIs can be explicitly assigned to resources independently from the use of OAI repositories and the OAI-PMH, if desired.

Lexical Signatures [29] makes a proposal for identifying Web documents that is different from what has been discussed up to now. Indeed, the *Lexical Signatures (LSs)* aim at uniquely identifying a Web document by means of a signature extracted from its content and not by means of using some identifiers, as in the case of URLs.

[7] `http://www.niso.org/committees/committee_ax.html`
[8] `http://purl.oclc.org/`
[9] `http://www.ukoln.ac.uk/distributed-systems/poi/`
[10] `http://www.openarchives.org/`

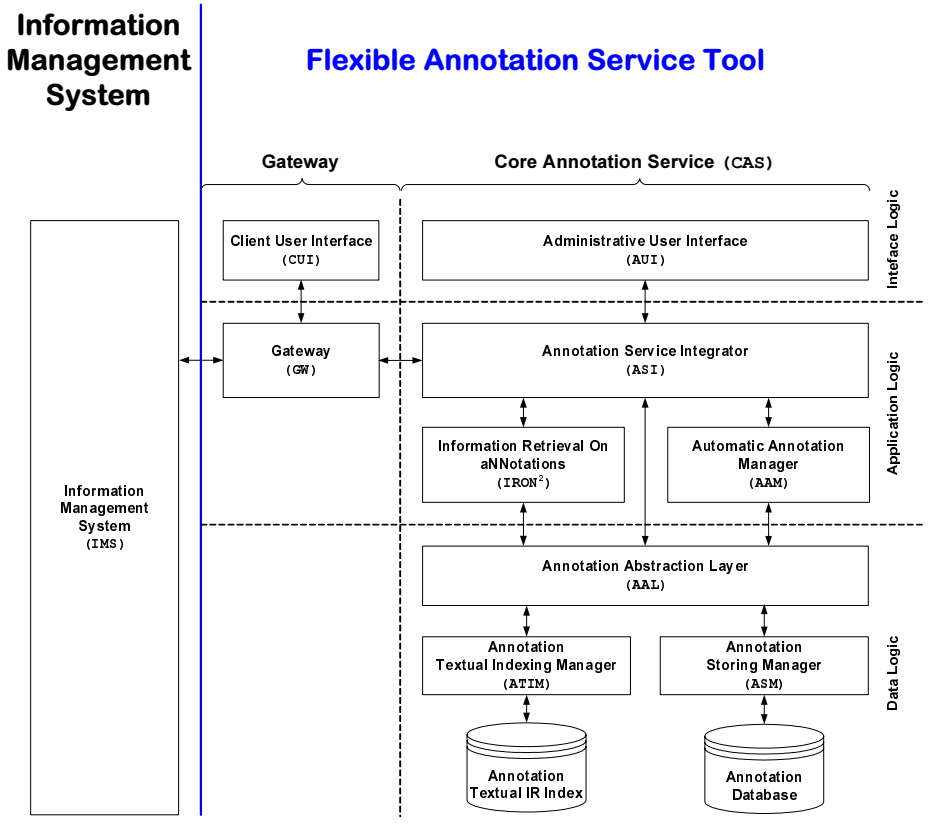


Fig. 2. Detailed architecture of the FAST system

LSs are a bunch of keywords extracted from a Web document that are used as query for a *Search Engine (SE)*. In this way, if a Web document cannot be found by means of its URL, then the LS of the document can be submitted to a SE in order to search and locate the document anyway. In conclusion, LSs represent an interesting alternative with respect to various kinds of identifiers and handles, due to the fact that LSs offer the possibility to almost uniquely identify a Web document by exploiting its own content.

4 FAST Conceptual Architecture

Figure 2 demonstrates the complete conceptual architecture of FAST, where FAST is depicted on the right, and the generic IMS is represented on the left. On the whole, the architecture is organized along two dimensions:

– *horizontal decomposition* (from left to right): consists of the IMS, the gateway and the CAS. It separates the core functionalities of FAST from the problem of integrating FAST into a specific IMS.

The horizontal decomposition allows us to accomplish the first two requirements of our architecture, since FAST is a stand-alone system that can be integrated with different IMSs by changing the gateway;

– *vertical decomposition* (from bottom to top): consists of three layers – the data, application and interface logic layers – and it is concerned with the organization structure of the CAS.
  This decomposition allows us to achieve a better modularity within FAST and to properly describe the behaviour of FAST by means of isolating specific functionalities at the proper layer. Moreover, this decomposition makes it possibile to clearly define the functioning of FAST by means of communication paths that connect the different components of FAST itself. In this way, the behaviour of the FAST system is designed in a modular and extensible way.

The conceptual architecture of FAST is designed at a high level of abstraction in terms of abstract *Application Program Interfaces (APIs)* using an *Object Oriented (OO)* approach. In this way, we can model the behaviour and the functioning of FAST without worrying about the actual implementation of each component. Different alternative implementations of each component could be provided, still keeping a coherent view of the whole architecture of the FAST system. We achieve this abstraction level by means of a set of interfaces, which define the behaviour of each component of FAST in abstract terms. Then, a set of abstract classes partially implement the interfaces in order to define the actual behaviour common to all of the implementations of each component. Finally, the actual implementation is left to the concrete classes, inherited from the abstract ones, that fit FAST into a given architecture, such as a WS or a P2P architecture. Furthermore, we apply the *abstract factory* design pattern [30], which uses a factory class that provides concrete implementations of a component, compliant with its interface, in order to guarantee a consistent way of managing the different implementations of each component. Java is the programming language in use for developing FAST. Java ensures us great portability across different hardware and software platforms, thus providing us with a further level of flexibility.

In the following sections we describe each component of FAST, according to figure 2, from bottom to top.

5 Data Logic Layer

5.1 Annotation Storing Manager

The *Annotation Storing Manager (ASM)* manages the actual storage of the annotations and provides a persistence layer for storing the objects which represent the annotation and which are used by the upper layers of the architecture.

The ASM relies on a *Relational DBMS (RDBMS)* in order to store annotations. The database schema is given by the mapping to the relational data model of the *Entity–Relationship (ER)* schema for modelling annotations, which has been proposed in [10]. Thus, the ASM provides a set of basic operations for storing, retrieving, updating, deleting and searching annotations in a SQL–like fashion. Furthermore, it takes care of mapping the objects which represent the annotations into their equivalent representation in

the relational model, according to the *Data Access Object (DAO)*[11] and the *Transfer Object (TO)*[11] design patterns.

The DAO implements the access mechanism required to work with the underlying data source, i.e. it offers access to the RDBMS using the *Java DataBase Connectivity (JDBC)* technology. The components that rely on the DAO are called *clients* and they use the interface exposed by the DAO, which completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients. Essentially, the DAO acts as an adapter between the clients and the data source. The DAO makes use of TOs as data carriers in order to return data to the client. The DAO may also receive data from the client in a TO in order to update the data in the underlying data source.

In conclusion, all of the other components of FAST deal only with objects representing annotations, which are the TOs of our system, without worrying about the details related to the persistence of such objects.

5.2 Annotation Textual Indexing Manager

The *Annotation Textual Indexing Manager (ATIM)* provides a set of basic operations for indexing and searching annotations for *Information Retrieval (IR)* purposes.

The ATIM is a full-text *Information Retrieval System (IRS)* and deals with the textual content of an annotation. It is based on the experience acquired in developing *Information Retrieval ON (IRON)*, the prototype IRS which has been used for participating in the *Cross-Language Evaluation Forum (CLEF)*[12] evaluation campaigns since 2002 [31,32,33,34]. CLEF is an international evaluation initiative aimed at providing an infrastructure for evaluating IRSs in a multilingual context.

Figure 3 shows the architecture of the last version of IRON, which has been used during the CLEF 2004 evaluation campaign [34]. Please note that in figure 3 the Logging component has been duplicated to make the figure more easily legible, but there actually is only one Logging component in the system.

IRON is a Java multi-threaded program, which provides textual IR functionalities and enables concurrent indexing and searching of document collections for both monolingual and bilingual tasks. IRON is made up of the following components:

- **Lexer:** implements an efficient lexer using JFlex 1.4[13], a lexer generator for Java. The current lexer is able to process any multilingual CLEF collection in a transparent way with respect to the document structure and to different character encodings, such as ISO 8859-1 or UNICODE[14].
- **IR engine:** is built on top of the Lucene 1.4 RC4[15] library, which is a high-performance text search engine library written entirely in Java. Lucene implements

[11] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/`
[12] `http://clef.isti.cnr.it/`
[13] `http://www.jflex.de/`
[14] The lexer has been designed and developed by G. M. Di Nunzio [33,34].
[15] `http://jakarta.apache.org/lucene/docs/index.html`

the vector space model, and a $(\mathrm{tf} \times \mathrm{idf})$–based weighting scheme [35]. Some parts of the Lucene library were completely rewritten, i.e. a set of parallel classes has been written without modifying the original source code of Lucene, so that IRON remain compatible with the official Jakarta distribution of Lucene. In particular, those parts of Lucene concerning the text processing, such as tokenization, elimination of stop words, stemming, and the query construction, have been modified. Furthermore Lucene has been adapted to the logging infrastructure of IRON;

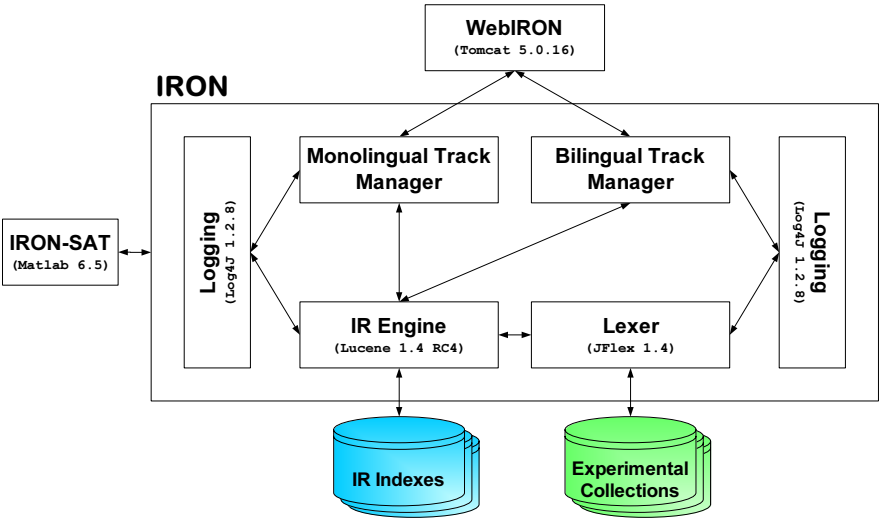


Fig. 3. Architecture of IRON

- **Monolingual Track Manager:** drives the underlying IR engine and provides high-level indexing and searching functionalities in order to carry out monolingual tasks. It provides a high-level API that allows us to easily plug together the different components of an IRS. This API can be further used to create a front-end application to IRON: for example we can develop a command-line application, a *Graphical User Interface (GUI)* for a stand-alone application, or a Web based *User Interface (UI)* to IRON;
- **Bilingual Track Manager:** drives the underlying IR engine and provides high-level indexing and searching functionalities in order to carry out the bilingual tasks. As the Monolingual Track Manager, also the Bilingual Track Manager provides a high-level API that can be used to develop different kinds of UIs for IRON;
- **Logging:** provides a full-fledged logging infrastructure, based on the Log4J 1.2.8[16] Java library. Each other component of IRON sends information about its status to the logging infrastructure, thus allowing us to track each step of the experiment.

[16] `http://logging.apache.org/log4j/docs/`

IRON is partnered with two other tools:

- **WebIRON:** is a Java servlet based Web interface. WebIRON is based on the Tomcat 5.0.16[17] Web server, making IRON a Web application. It provides a set of wizards which help the user to set all the parameters and choose the IR components, which are needed in order to conduct a run or, more generally, an IR experiment.
- ***IRON - Statistical Analysis Tool (IRON-SAT):*** is a Matlab program that interacts with IRON in order to carry out the statistical analysis of the experimental results. IRON-SAT parses the experimental data and stores the parsed information into a data structure suitable for the following processing. It is designed in a modular way, so that new statistical tests can be easily added to the existing code. The statistical analysis is performed using the Statistics Toolbox 4.0 provided by Matlab 6.5[18].

5.3 Annotation Abstraction Layer

The *Annotation Abstraction Layer (AAL)* abstracts the upper layers from the details of the actual storage and indexing of annotations, providing uniform access to the functionalities of the ASM and the ATIM.

The AAL provides the typical *Create–Read–Update–Delete (CRUD)* data management operations, coordinating the work of the ASM and the ATIM together. For example, when we create a new annotation, we need to put it into both the ASM and the ATIM.

Furthermore, the AAL provides search capabilities by properly forwarding the queries to the ASM or to the ATIM. Our modular architecture allows us to partner the ATIM, which is specialised for providing full text search capabilities, with other IRSs, which are specialised for indexing and searching other kinds of media. In any case, the addition of other specialised IRSs is transparent for the upper layers, due to the fact that the AAL provides the upper layers with an uniform access to those IRSs.

Note that both the ASM and the ATIM are focused on each single annotation in order to properly store and index it. On the other hand, both the ASM and the ATIM do not have a comprehensive view of the relationships that exist between documents and annotations. On the contrary, the AAL has a global knowledge of the annotations and their relationships by using the hypertext existing between documents and annotations. For example, if we delete an annotation that is part of a thread of annotations, what policy do we need to apply? Do we delete all the annotations that refer to the deleted one or do we try to reposition those annotations? The ASM and the ATIM alone would not be able to answer this question but, on the other hand, the AAL can drive the ASM and the ATIM to perform the correct operations by exploiting the hypertext between documents and annotations.

In conclusion, the AAL, the ASM and the ATIM constitute an IMS specialised in managing annotations, as a DBMS is specialised in managing structured data.

[17] `http://jakarta.apache.org/tomcat/index.html`
[18] `http://www.mathworks.com/`

6 Application Logic Layer

6.1 Automatic Annotation Manager

The *Automatic Annotation Manager (AAM)* automatically creates annotations for a given document. Automatic annotations can be created by using topic detection techniques in order to associate each annotation with its related topic, which constitutes the context of the annotation. In this way, a document can be re-organized and segmented into topics, whose dimension can range in many different sizes, and annotations can present a brief description of those topics.

6.2 Information Retrieval On aNNotations

Annotations introduce a new content layer aimed at elucidating the meaning of underlying documents, so that annotations can make hidden facets of the annotated documents more explicit. Thus, we can image to exploit annotations for retrieval purposes in order to satisfy better the user's information needs.

We need to develop a search strategy which is able to effectively take into account the multiple sources of evidence coming from both documents and annotations. Indeed, the combining of these multiple sources of evidence can be exploited in order to improve the performances of an information management system. We aim to retrieve more relevant documents and to rank them better than the case of a query without using annotations.

The *Unified Modeling Language (UML)* sequence diagram of figure 4 shows how searching for documents by exploiting annotations involves many components of FAST. Remember that we aim at combining the source of evidence which comes from annotations, managed by FAST, with the source of evidence which comes from documents, managed by the IMS. Thus, the search strategy requires the cooperation of both FAST and the IMS in order to acquire these two sources of evidence. Firstly, FAST receives a query from the end-user, which is dispatched from the user interface to *Information Retrieval On aNNotations (IRON^2)*. Secondly, the query is used to select all the relevant annotations, that is IRON^2 asks the *Annotation Service Integrator (ASI)* to find all the relevant annotations. Then, the hypertext between documents and annotations can be built and used to identify the documents that are related to the found annotations. Now we aim to combine the source of evidence which comes from the documents identified by the annotations with the one which comes from the documents managed by the IMS, as previously explained. Since the source of evidence concerning the documents is completely managed by the IMS, the FAST system has to query the IMS, which gives us back a list of relevant documents. Finally, once the FAST system has acquired this information from the IMS, it can combine this information with the source of evidence which comes from the documents identified by annotations in order to create a list of fused result documents that are presented to the users.

The reader, which is interested in knowing how FAST exploits annotations in order to search and retrieve relevant documents to answer to a user query, can refer to [36] where it is provided a formal framework which is used in facing the searching for documents from digital collections of annotated documents.

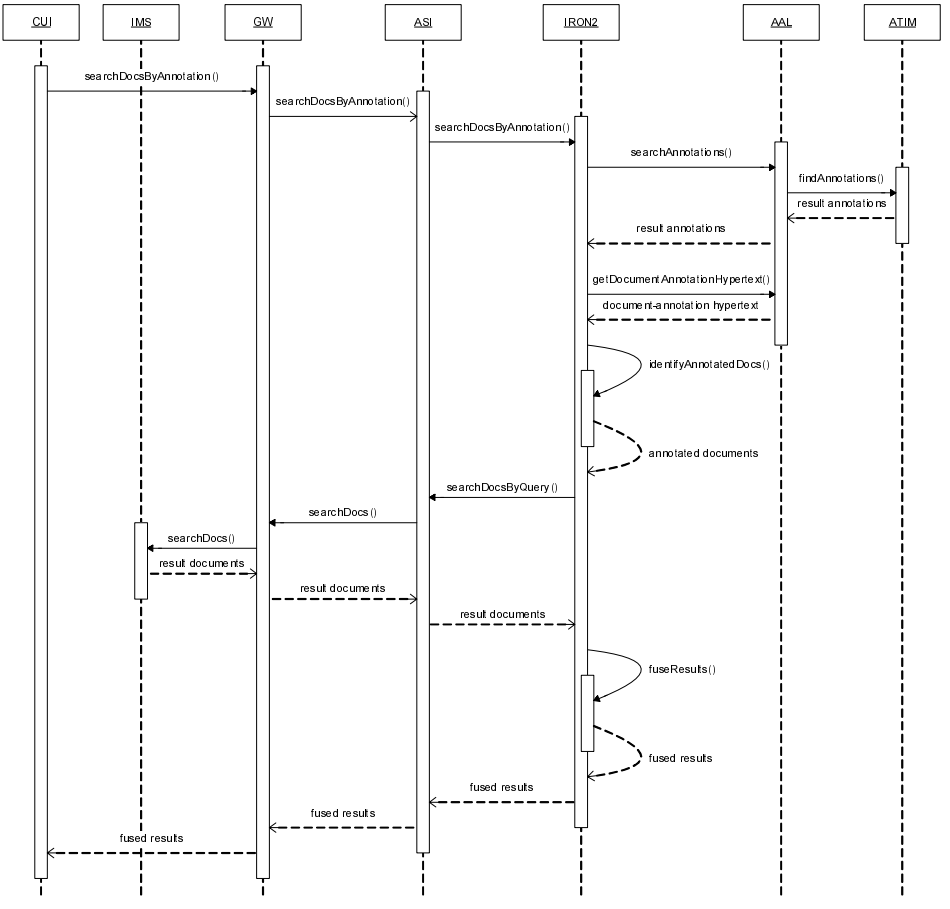


Fig. 4. Sequence diagram for searching documents by exploiting annotations

6.3 Annotation Service Integrator

The ASI integrates the underlying components and provides uniform access to them. It represents the entry point to the CAS for both the gateway and the user interface, dispatching their requests to underlying layers and then collecting the responses from the underlying layers.

The UML sequence diagram of figure 4 shows how the ASI plays a central role in coordinating the different components of FAST. In the example of figure 4, the ASI forwards the user query to IRON^2; it dispatches the request for relevant documents of IRON^2 to the *Gateway (GW)* in order to submit this query to the IMS; then, it passes the results provided by the IMS back to IRON^2; finally, it gives the fused result list produced by IRON^2 back to the GW in order to return this list to the user interface.

6.4 Gateway

As already discussed in Section 3, the GW provides functionalities of mediator between the CAS and the IMS. By changing the gateway, we can share FAST with different IMSs. In this way, we can provide a wide range of different architectural choices: firstly, the CAS could be connected to a specific IMS which uses proprietary protocols and data structures and, in this case, the gateway can implement them, as we did in the case of the OpenDLib[19] digital library [10]; secondly, we could employ WS to carry out the gateway, so that FAST is accessible in a more standardized way; finally, the gateway could be used to adapt FAST to a P2P network of IMSs.

7 Interface Logic Layer

7.1 Administrative User Interface

The *Administrative User Interface (AUI)* is a Web-based UI for the administration of FAST. It provides the different functionalities needed to configure and run FAST, such as the choice of the gateway to be used, the creation and management of the users granted by the system, and so on.

7.2 Client User Interface

The *Client User Interface (CUI)* provides end–users with an interface for creating, modifying, deleting and searching annotations.

The CUI is connected to, or even directly integrated into, the gateway, so that it represents a user interface tailored to the specific IMS for which the gateway is developed. In this way, the gateway forwards the requests from the CUI to the ASI, as it is shown in the example of figure 4.

8 Conclusions and Future Work

This paper discussed the conceptual architecture of the FAST system, which separates core functionalities from their integration in any particular IMS. In this way, FAST acts as a bridge between different IMSs and allows the hypertext to cross the boundaries of a single IMS, in order to exploit annotations as an active and effective collaboration tool for users.

We plan to enhance our conceptual architecture in order to support a network of P2P FAST systems. In this way, we will be able to implement FAST not only as a stand–alone system, that can be integrated into different IMSs, but also as a P2P network of FASTs that cooperate in order to provide advanced annotation functionalities to different IMSs.

[19] `http://www.opendlib.com/`

Acknowledgements

This work is partially funded by the ECD project, which is a joint program between the Italian National Research Council (CNR) and the Ministry of Education (MIUR), with regards to law 449/97-99. The work is also partially supported by the DELOS Network of Excellence on Digital Libraries, as part of the Information Society Technologies (IST) Program of the European Commission (Contract G038-507618).

References

1. Agosti, M., Ferro, N.: Chapter X: Managing the Interactions between Handheld Devices, Mobile Applications, and Users. In Lim, E.P., Siau, K., eds.: Advances in Mobile Commerce Technologies, Idea Group, Hershey, USA (2003) 204–233
2. Nagao, K.: Digital Content Annotation and Transcoding. Artech House, Norwood (MA), USA (2003)
3. Rigaux, P., Spyratos, N.: Metadata Inference for Document Retrieval in a Distributed Repository. In Maher, M.J., ed.: Proc. 9th Asian Computing Science Conference – Advances in Computer Science (ASIAN 2004) – Higher Decision Making. Dedicated to Jean-Louis Lassez on the Occasion of His 5th Cycle Birthday, Lecture Notes in Computer Science (LNCS) 3321, Springer, Heidelberg, Germany (2004) 418–436
4. Gueye, B., Rigaux, P., Spyratos, N.: Taxonomy-Based Annotation of XML Documents: Application to eLearning Resources. In Vouros, G.A., Panayiotopoulos, T., eds.: Proc. 3rd Helenic Conference on AI – Methods and Applications of Artificial Intelligence (SETN 2004), Lecture Notes in Computer Science (LNCS) 3025, Springer, Heidelberg, Germany (2004) 33–42
5. Kahan, J., Koivunen, M.R.: Annotea: an open RDF infrastructure for shared Web annotations. In Shen, V.Y., Saito, N., Lyu, M.R., Zurko, M.E., eds.: Proc. 10th International Conference on World Wide Web (WWW 2001), ACM Press, New York, USA (2001) 623–632
6. Buneman, P., Khanna, S., Tajima, K., Tan, W.C.: Archiving Scientific Data. ACM Transactions on Database Systems (TODS) **29** (2004) 2–42
7. Buneman, P., Khanna, S., Tan, W.C.: Why and Where: A Characterization of Data Provenance. In Van den Bussche, J., Vianu, V., eds.: Proc. 8th International Conference on Database Theory (ICDT 2001), Lecture Notes in Computer Science (LNCS) 1973, Springer, Heidelberg, Germany (2001) 316–330
8. Buneman, P., Khanna, S., Tan, W.C.: On Propagation of Deletions and Annotations Through Views. In Abiteboul, S., Kolaitis, P.G., Popa, L., eds.: Proc. 21st ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems (PODS 2002), ACM Press, New York, USA (2002) 150–158
9. Bhagwat, D., Chiticariu, L., Tan, W.C., Vijayvargiya, G.: An Annotation Management System for Relational Databases. In Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B., eds.: Proc. 30th International Conference on Very Large Data Bases (VLDB 2004), Morgan Kaufmann (2004) 900–911
10. Agosti, M., Ferro, N.: Annotations: Enriching a Digital Library. In Koch, T., Sølvberg, I.T., eds.: Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003), Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany (2003) 88–100

11. Frommholz, I., Brocks, H., Thiel, U., Neuhold, E., Iannone, L., Semeraro, G., Berardi, M., Ceci, M.: Document-Centered Collaboration for Scholars in the Humanities – The COLLATE System. In Koch, T., Sølvberg, I.T., eds.: Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003), Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany (2003) 434–445
12. Thiel, U., Brocks, H., Frommholz, I., Dirsch-Weigand, A., Keiper, J., Stein, A., Neuhold, E.J.: COLLATE – A collaboratory supporting research on historic European films. International Journal on Digital Libraries **4** (2004) 8–12
13. Agosti, M., Benfante, L., Orio, N.: IPSA: A Digital Archive of Herbals to Support Scientific Research. In Sembok, T.M.T., Zaman, H.B., Chen, H., Urs, S.R., Myaeng, S.H., eds.: Proc. 6th International Conference on Asian Digital Libraries. Digital Libraries – Digital Libraries: Technology and Management of Indigenous Knowledge (ICADL 2003), Lecture Notes in Computer Science (LNCS) 2911, Springer, Heidelberg, Germany (2003) 253–264
14. Agosti, M., Ferro, N., Orio, N.: Annotations as a Support to Research Users. In Catarci, T., Christodoulakis, S., Del Bimbo, A., eds.: Proc. 7th International Workshop of the EU Network of Excellence DELOS on Audio-Visual Content and Information Visualization in Digital Libraries (AVIVDiLib'05), Centromedia, Viareggio, Italy (2005) 117–120
15. Agosti, M., Ferro, N., Orio, N.: Annotating Illuminated Manuscripts: an Effective Tool for Research and Education. In Proc. 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2005), ACM Press, New York, USA (2005) (in print)
16. Marshall, C.C.: Annotation: from Paper Books to the Digital Library. In Allen, R.B., Rasmussen, E., eds.: Proc. 2nd ACM International Conference on Digital Libraries (DL 1997), ACM Press, New York, USA (1997) 233–240
17. Marshall, C.C., Brush, A.J.B.: Exploring the Relationship between Personal and Public Annotations. In Chen, H., Wactlar, H., Chen, C.C., Lim, E.P., Christel, M., eds.: Proc. 4th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2004), ACM Press, New York, USA (2004) 349–357
18. Schilit, B.N., Price, M.N., Golovchinsky, G.: Digital Library Information Appliances. In Witten, I., Akscyn, R., Shipman, F.M., eds.: Proc. 3rd ACM International Conference on Digital Libraries (DL 1998), ACM Press, New York, USA (1998) 217–226
19. Marshall, C.C., Golovchinsky, G., Price, M.N.: Digital Libraries and Mobility. Communications of the ACM **44** (2001) 55–56
20. Marshall, C.C., Ruotolo, C.: Reading-in-the-Small: A Study of Reading on Small Form Factor Devices. In Hersh, W., Marchionini, G., eds.: Proc. 2nd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2002), ACM Press, New York, USA (2002) 56–64
21. Bottoni, P., Civica, R., Levialdi, S., Orso, L., Panizzi, E., Trinchese, R.: MADCOW: a Multimedia Digital Annotation System. In Costabile, M.F., ed.: Proc. Working Conference on Advanced Visual Interfaces (AVI 2004), ACM Press, New York, USA (2004) 55–62
22. Bottoni, P., Civica, R., Levialdi, S., Orso, L., Panizzi, E., Trinchese, R.: Storing and Retrieving Multimedia Web Notes. In Bhalla, S., ed.: Proc. 4th International Workshop on Databases in Networked Information Systems (DNIS 2005), Lecture Notes in Computer Science (LNCS) 3433, Springer, Heidelberg, Germany (2005) 119–137
23. Agosti, M.: An Overview of Hypertext. In Agosti, M., Smeaton, A., eds.: Information Retrieval and Hypertext. Kluwer Academic Publishers, Norwell (MA), USA (1996) 27–47
24. Marshall, C.C.: Toward an Ecology of Hypertext Annotation. In Akscyn, R., ed.: Proc. 9th ACM Conference on Hypertext and Hypermedia (HT 1998): links, objects, time and space-structure in hypermedia systems, ACM Press, New York, USA (1998) 40–49
25. Agosti, M., Ferro, N., Frommholz, I., Thiel, U.: Annotations in Digital Libraries and Collaboratories – Facets, Models and Usage. In Heery, R., Lyon, L., eds.: Proc. 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2004), Lecture Notes in Computer Science (LNCS) 3232, Springer, Heidelberg, Germany (2004) 244–255

26. Frommholz, I., Thiel, U., Kamps, T.: Annotation-based Document Retrieval with Four-Valued Probabilistic Datalog. In Baeza-Yates, R., Maarek, Y., Roelleke, T., de Vries, A.P., eds.: Proc. 3rd XML and Information Retrieval Workshop and the 1st Workshop on the Integration of Information Retrieval and Databases (WIRD2004), http://homepages.cwi.nl/~arjen/wird04/wird04-proceedings.pdf [last visited 2004, November 22] (2004) 31–38
27. Fogli, D., Fresta, G., Mussio, P.: On Electronic Annotation and Its Implementation. In Costabile, M.F., ed.: Proc. Working Conference on Advanced Visual Interfaces (AVI 2004), ACM Press, New York, USA (2004) 98–102
28. Berners-Lee, T., Fielding, R., Irvine, U.C., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 (1998)
29. Park, S.T., Pennock, D., Giles, C.L., Krovetz, R.: Analysis of Lexical Signatures for Improving Information Persistence on the World Wide Web. ACM Transactions on Information Systems (TOIS) **22** (2004) 540–572
30. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object–Oriented Software. Addison-Wesley, Reading (MA), USA (1995)
31. Agosti, M., Bacchin, M., Ferro, N., Melucci, M.: Improving the Automatic Retrieval of Text Documents. In Peters, C., Braschler, M., Gonzalo, J., Kluck, M., eds.: Advances in Cross-Language Information Retrieval, Third Workshop of the Cross–Language Evaluation Forum (CLEF 2002) Revised Papers, Lecture Notes in Computer Science (LNCS) 2785, Springer, Heidelberg, Germany (2003) 279–290
32. Bacchin, M., Ferro, N., Melucci, M.: A Probabilistic Model for Stemmer Generation. Information Processing & Management **41** (2005) 121–137
33. Di Nunzio, G.M., Ferro, N., Melucci, M., Orio, N.: Experiments to Evaluate Probabilistic Models for Automatic Stemmer Generation and Query Word Translation. In Peters, C., Braschler, M., Gonzalo, J., Kluck, M., eds.: Comparative Evaluation of Multilingual Information Access Systems: Fourth Workshop of the Cross–Language Evaluation Forum (CLEF 2003) Revised Selected Papers, Lecture Notes in Computer Science (LNCS) 3237, Springer, Heidelberg, Germany (2004) 220–235
34. Di Nunzio, G.M., Ferro, N., Orio, N.: Experiments on Statistical Approaches to Compensate for Limited Linguistic Resources. In Peters, C., Clough, P., Gonzalo, J., Jones, G., Kluck, M., Magnini, B., eds.: Fifth Workshop of the Cross–Language Evaluation Forum (CLEF 2004) Revised Selected Papers, Lecture Notes in Computer Science (LNCS), Springer, Heidelberg, Germany (in print) (2005)
35. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, New York, USA (1983)
36. Agosti, M., Ferro, N.: Annotations as Context for Searching Documents. In Crestani, F., Ruthven, I., eds.: Proc. 5th International Conference on Conceptions of Library and Information Science - Context: nature, impact and role, Lecture Notes in Computer Science (LNCS) 3507, Springer, Heidelberg, Germany (2005) 155–170

A Service-Oriented Grid Infrastructure for Multimedia Management and Search

Michael Mlivoncic[1], Christoph Schuler[2], Can Türker[3], and Sören Balko[4]

[1] Swiss Federal Institute of Technology Zurich,
Institute of Information Systems, CH–8092 Zurich, Switzerland
mlivonci@inf.ethz.ch
[2] Ergon Informatik AG, Kleinstrasse 15, CH-8008 Zurich, Switzerland
christoph.schuler@ergon.ch
[3] Functional Genomics Center Zurich,
Winterthurerstrasse 190, Irchel, Y32 H06, CH–8057 Zurich, Switzerland
tuerker@acm.org
[4] University for Health Sciences, Medical Informatics and Technology (UMIT) Innsbruck,
Institute for Information Systems, Eduard Wallnöfer-Zentrum 1,
A-6060 Hall i. Tirol, Austria
Soeren.Balko@umit.at

Abstract. Nowadays, digital libraries are inherently dispersed over several peers of a steadily increasing network. Dedicated peers may provide specialized, computationally expensive services such as image similarity search. Usually, the peers of such a network are uncoordinated in the sense that their content and services are not linked together. Nevertheless, users expect to transparently access and modify all the (multimedia) content anytime from anywhere not only in an efficient and effective but also consistent way. To match these demands, future digital libraries require an infrastructure that combines various information technologies like databases, service-oriented architectures, peer-to-peer and grid computing. In this paper, we sketch such an infrastructure and illustrate how an example digital library application can work atop it. In detail, we show how similarity search is supported by this infrastructure, and discuss how query distribution and load balancing based on domain-specific knowledge can be exploited by the infrastructure to reduce query response times.

1 Introduction

Future digital libraries shall provide access to any multimedia content anytime and anywhere in a user-friendly, efficient, and effective way. One problem of nowadays digital libraries is that they are dispersed over a network in an uncoordinated fashion such that each peer of the network often works isolated without regarding other peers that manage related content. Another problem is to efficiently handle the steadily increasing amount of requests for multimedia content. Besides, the distributed content should be kept consistent and provided in a personalized way. Hence, an infrastructure is required for digital libraries that is a reliable, scalable, customizable, and integrated environment.

To build such an infrastructure, the best aspects of databases, service-orientation, peer-to-peer and grid computing must be combined. We refer to such an infrastructure

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 167–187, 2005.

as a *hyperdatabase* [12]. Within such an environment, databases and special servers provide basic services, such as efficient and reliable storage for various kinds of multimedia data like image, audio, and video. Service-orientation [13] helps to describe, customize, and deploy complex services, such as sophisticated search of images with respect to their content and annotations. The peer-to-peer paradigm [5] allows a loosely-coupled integration of digital library services and ad-hoc sharing of information, such as recommendations and annotations. Since certain services within a digital library are computationally intensive, e.g., the extraction of features from multimedia objects to support content-based similarity search, grid technology [10] supports the optimal utilization of the given computing resources.

In the recent years, we have developed OSIRIS (**O**pen **S**ervice **I**nfrastructure for **R**eliable and **I**ntegrated process **S**upport) [7], which is a prototype of a hyperdatabase infrastructure. As common standards like WSDL and SOAP, OSIRIS helps to access a wide range of service types and allows isolated service calls. In addition, OSIRIS support processes (compound services) as a means for combining existing services and executing them under certain (transactional) guarantees [19]. In a digital library, such services can be used, for instance, for the maintenance of dependencies among the various data repositories of the digital library. In this way, a sophisticated search may benefit from always up-to-date indexes.

In this paper, we describe how to organize such a self-contained application. The ISIS (**I**nteractive **SI**milarity **S**earch) application demonstrates an efficient management and organization of large multimedia collections atop OSIRIS. It features a wide range of functions that allow for metadata management as well as efficient and effective content-based similarity search on multimedia objects. The implementation of ISIS consequently follows the idea of service-orientation. Specifically, all digital library functionality is encapsulated by services, such as storage services for arbitrary types of media objects or feature extraction services for given media types. Such services are used to compose the entire ISIS application.

In principle, any service can be executed locally in a stand-alone manner, as with mobile devices, for example. However, due to several reasons, e.g., resource or license restrictions, some services might be locally unavailable. In such cases, the service execution depends on external services, which have to be bound and invoked on demand. On the other hand, in some cases the locally available services are sufficient to completely execute a compound service. For instance, even a content-based image similarity search does not require a feature extraction service provided the reference image is already part of the collection (i.e., is already indexed).

It is important to note that the entire service execution is transparent to the service designer. She therefore can fully focus on specifying and implementing the core service functionality itself. Service runtime environment tasks (startup, updates,...) and communication with the outside world are handled by the hyperdatabase infrastructure. As our experimental evaluations [16,15] show, this infrastructure does not only support dynamic changes of the execution environment but it also scales with the expected huge number of users, services, and grid peers of future digital libraries.

The infrastructure monitors the service execution as well as the routing of the corresponding services. Whenever necessary, it distributes and replicates certain services

on multiple grid peers in order to boost the performance. In this paper, we present an approach to reduce the response time of similarity queries by transparently distributing the query over the available peers of the infrastructure. The approach exploits the current peer loads as well as other peer-specific characteristics.

The rest of this paper is organized as follows: Section 2 gives an overview of ISIS together with the requirements for the underlying infrastructure. Section 3 describes such an infrastructure, OSIRIS, and discuss various important issues like service execution and service registration. Section 4 discusses the execution of similarity queries in detail. Finally, Section 5 concludes the paper.

2 ISIS – A Service-Oriented Application

ISIS (Interactive SImilarity Search) is a powerful service-oriented application for efficient management and organization of multimedia collections. The application features a wide range of functions that allow meta data management as well as efficient and effective content-based similarity search on multimedia objects. The realization of ISIS consequently follows the idea of service-orientation. All basic functionality is encapsulated by a set of services.

We distinguish five classes of such services:

Storage Services. These services provide storage for arbitrary types of media objects. A storage service at a certain peer may be dedicated to a certain content like video clips or images. Storage services can also act as web-caches for remote content in order to speed-up access to often used data. A storage service closely monitors its attached repositories for changes. If, for example, a new object is added to one of its repositories, it will issue a "new object" event. Likewise, a (local) deletion of an object will lead to an "object-deleted" event. These events can be handled by the digital library infrastructure, for example, to keep the consistency between object data and indexed information.

Metadata Services. Such services maintain keyword annotations, textual descriptions, as well as other object properties, such as the membership in several (sub)collections, or predicates like "copyrighted". As an media object might be available in different versions at different locations (e.g, as thumbnail of the image at thumbnail-storage, primary high-resolution image copy at remote storage location), metadata services also keep track of those locations and corresponding properties of the object versions at the various locations (e.g. resolution or thumbnail/primary-role). There is no fixed vocabulary for describing such properties. Hence, ISIS can store arbitrary information about the objects. For example, one might want to store textual information gathered from the surrounding web pages together with an image or the artist and the song title together with a piece of music. Besides such "per-object" information, metadata services also maintain general knowledge about the object types themselves. This includes existing feature descriptors for an object type, the availability of feature extractors and indexing containers within the digital library infrastructure that are able to manage a given feature descriptor.

Feature Extraction Services. Content-based retrieval depends on features that describe the raw content of media objects in a certain feature domain. For example,

the pixel information of an image can be described in the color or texture feature domain [11,18,6], while a piece of music can be described in terms of beat and pitch [20]. A feature is therefore a kind of descriptive fingerprint for an object. In content-based retrieval, object similarity is expressed as similarity of their descriptors in a given feature domain. For a given media type, there could be several feature extraction services computing various kinds of feature descriptors. Also, there is more than one way to describe a concept like color. For example, we could use a histogram with 64, 256 or whatsoever bins. We could also use color moments. There are many meaningful descriptors and variants around. One might want to use several of them — even in combination. In many cases, the extraction of features is a computationally very expensive task. Therefore, feature extraction can profit from grid infrastructures.

Indexing and Search Services. Indexes allow for efficient search of objects. Different indexes have to handle data from different domains: Besides often high dimensional feature descriptors, object predicates and numerical values as well as keyword annotations of the objects should be indexed efficiently. Searching over an arbitrary *combination* of those attributes efficiently is a non-trivial task. Efficient search strategies on this level are however beyond the scope of this paper. For further information please refer to [22,23,3]. At this point, we only state that concept and design of such a service should be carefully chosen in a way that it allows for massive scalability through dynamic partitioning of a query onto a set of several search services. Those services will be spread all over the grid infrastructure, similar to the extraction services.

Presentation Services. These services provide frontend functionality to browse and query the multimedia collection and also to initiate some maintenance and administrative tasks. In ISIS, this task is shared among services for the interactive part (frontend), the layout part (XML-to-HTML rendering) and supporting services (session management and template repository).

The entire ISIS application consists of a set of compound services over these basic services. Once the application logic is divided into such basic services, they can be distributed and replicated without changing the description (implementation) of any (compound) service. Figure 1 shows the insertion of a multimedia object as an example for a compound service. The given service is triggered by the "new object" event of a storage service mentioned above when a new media object physically enters the repository.

The first activity of the compound service is to store the object, i.e., the location and available meta information about the object is stored by the metadata service. Depend-

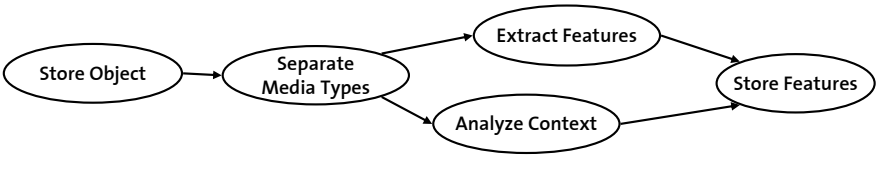


Fig. 1. Compound Service Insert Multimedia Object

ing on the media type, further information is extracted. In case of a web document, the object will not only contain an image, but also some text surrounding this image on the page. By analyzing the HTML source, it is possible to gather some layout information and applying some heuristics in order to determine textual descriptions that might be relevant for the image. This text can be indexed later on. Independent of the analysis of the context of the image and its surrounding, the extract features activity uses the raw pixel information of the image to extract several descriptors for color and texture. Note that it is transparent to the user, whether this activity is a single service or a compound service which is composed of single feature extraction services. The distinction between a single and a compound service is important for the infrastructure. By explicitly knowing about the semantics of the various activities of a compound service, the infrastructure is able to further parallelize the extraction to achieve better performance. The store features activity hands all gathered object descriptors and metadata information over to the metadata service, which will in turn care for the indexing and replication of each data item in a suitable way.

Infrastructure Requirements

ISIS performs some complex and sometimes computational expensive tasks. As ISIS may change over time, e.g., when new feature descriptors are becoming available, we need an infrastructure that provides us with flexibility in order to define and modify the logic of the digital library applications, i.e., the corresponding compound services, as necessary. In order to focus on the core digital library tasks, the infrastructure should support a transparent way of communication among services. For example, while designing the extraction service, we do not want to *explicitly* deal with workload distribution, it should be sufficient to "solicit" that a certain task, e.g., a feature extraction, should be executed on any one of the potential service providers. Thus, the infrastructure must also support service discovery. Service providers should be able to declare what kind of functionality they are offering and service users should be able to find them.

As mentioned before, it can be useful, if application logic specifies compound services over the basic services (cf. the feature extraction example). In such cases, feature extraction as well as search can profit even from a higher degree of parallelism. As multimedia content tends to be storage intensive — especially with large-scale general purpose digital libraries — one might also want to distribute the storage services and searching facilities over the peers of the grid. Besides this complexity, we still demand that the overall digital library should be reliable, i.e., (temporary) unavailability of single service *instances* shall not jeopardize the operation of the overall system. Services once invoked (like the insertion of an object) should lead to a guaranteed execution of the defined activities and thus ensure the consistency of the data within the metadata and indexing services. Mentioning consistency, we also want to ensure highest possible "freshness", i.e., changes should be propagated "immediately" instead of monthly updates like in Google and other major web search engines. On object deletion, references to that object should be removed immediately from all indexes. Also, changes of feature data should immediately be propagated to *any* indexing service related to that data.

3 The OSIRIS Hyperdatabase Infrastructure for ISIS

In the following, we sketch how a hyperdatabase infrastructure provides us with all the required functionality as elaborated so far. For further details on the infrastructure, we refer to [17,16].

3.1 Overview of the Infrastructure

A hyperdatabase supports applications following the ideas of a service-oriented architecture. Using a peer-to-peer network, service requests are transparently routed along the connected peers. Following the concept of service-oriented architecture, a service can be *atomic* or *compound*. Compound services are composed of existing services. The composition is realized using the notion of a transactional process [14]. By defining a data and control flow for processes, the involved service calls must appear in the specified application specific invocation order. Transactional processes allows for providing execution guarantees similar to transactions in classical databases. While databases focus on basic data querying and manipulation operations, a hyperdatabase orchestrates service calls. These service calls are executed according the description of the compound service.

In order to provide this functionality, the hyperdatabase infrastructure consists of a software layer installed on each peer of the grid. In Figure 2, these are the dark gray layers. A hyperdatabase layer integrates the peer into the overall hyperdatabase network. By this integration, local available services can be registered and used by all peers in the grid. Moreover, the local hyperdatabase layer is now enabled to transparently call remote services. In the figure, the light gray shaded grid symbolizes the Grid. Indeed, every peer can transparently connect and communicate with any other peer of the network.

Figure 3 shows the architecture of the hyperdatabase layer, which we will discuss in more detail later. Beside communication, the layer also supports management of compound services and load balancing. This functionality depends on grid common knowledge accessible via the replication manager available at every peer.

OSIRIS [17] as an implementation of a hyperdatabase is realized as a service-oriented application itself. Besides the core hyperdatabase layer, additional system functionality is needed to organize the grid and to provide service transparency and the execution of compound services. OSIRIS implements this functionality also in terms of services. As depicted in Figure 2, OSIRIS provides a bunch of core services that manage the global system knowledge:

- Register Service Description. This service holds interface information about all available service types in the system. Beside the interface definition, compound services feature a process definition that specify the exact composition of the corresponding existing services.
- Register Service Instance. In a dynamic system, the current system configuration must be available to all peers in the system. This global service provides ensures that all peers receive the needed information.

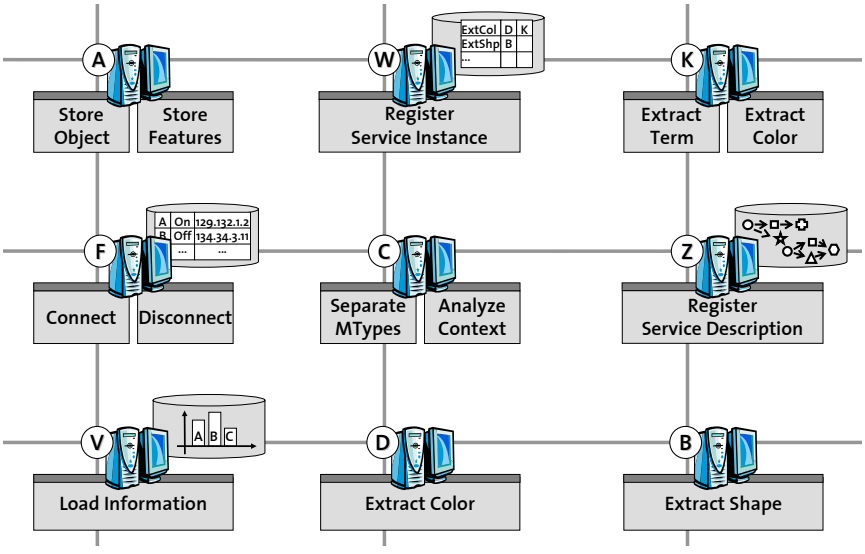


Fig. 2. The OSIRIS Hyperdatabase Infrastructure

- Connection Service. OSIRIS allows peers to be in disconnected mode. In this mode, a connected proxy represents the disconnected peer and collects the corresponding information.
- Load Information. The load service helps to optimize the workload balancing within the system. For this reason, it gathers current load information from peers. The feedback cycle within a large system such as OSIRIS is typically rather lazy. Therefore a current CPU load cannot be a measure for an effective workload balancing. However, the current length of the incoming work queue might be a good measure to equally distribute the load. For this reason, each peer evaluates an expected execution time t_Q for each service type.

Figure 2 also shows that there is no distinction between core services and atomic application-specific services such as Extract Color or Extract Term.

3.2 Peer-to-Peer Service Execution

As mentioned before, OSIRIS distinguishes two classes of services: atomic versus compound services. The execution of an atomic service consists of one call to a function provided by a service in the grid. A call to a compound service however initiates a peer-to-peer execution of a process. According to the description of a compound service, the contained services, i.e., the process activities, are executed sequentially. Although the execution semantics of the two service classes are different, the invocations of both services are not distinguishable to the caller. In other words, the caller does not care about whether the service is atomic or compound. The same holds for activities of a compound service, which can be compound services themselves.

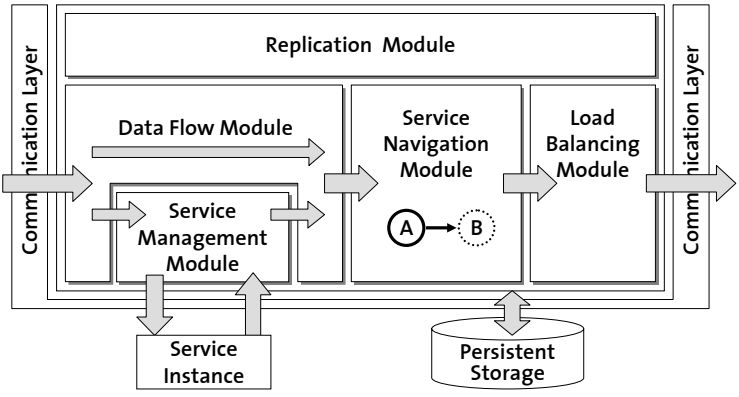


Fig. 3. The OSIRIS Hyperdatabase Layer

OSIRIS provides a transparent way of calling services. A service request that is sent to any peer of the network is transparently routed by the local hyperdatabase layer to an available instance of this service type. For that, OSIRIS implements a distributed service bus that provides routing and load balancing over all existing service instances.

Albeit the infrastructure hides the difference between an atomic and a compound service at the call interface, the execution within the infrastructure layer respects and exploits the differences between atomic and compound services. While the first has to be routed to a service instance, the latter must be driven by the infrastructure itself since its activities can spread over multiple peers of the grid. Therefore, each peer has to provide a minimal service manager that supports a peer-to-peer service execution. This includes the ability to call local services, to navigate to the subsequent activity of the compound service, and to handle execution failures. A compound service consists of a set of ordered services. These services has to be executed according the order defined by the service description. This information is available at the local replication manager. After instantiating a new compound service at any peer in the grid, it is migrated to the hyperdatabase layer of a peer that provides an instance for the first activity.

Figure 3 shows the flow of the service execution of one single step in a compound service. After the service has migrated to the hyperdatabase layer, it enters the *data flow module*. A part of the service data is needed to prepare the call of the service. The *service management module* executes a function at the service instance. After the execution of the service the resulting data is incorporated into the context of the compound service. The next task is to determine the service that has to be executed subsequently. Based on the locally replicated part of the service description, the *service navigation module* decides which service type to call next. The *load balancing module* finally routes the service instance to an available service instance considering system workload. This routing requires grid configuration information. Therefore, the *replication module* has also to provide grid configuration information. During the execution of a compound service, the service instance migrates from one peer to the next in a truly peer-to-peer fashion. After executing the last activity of the compound service, the response is sent back to the caller, as in the case of an atomic service.

Example 1. Assume a multimedia object is inserted into the ISIS digital library by calling our example compound service Insert Multimedia Object. OSIRIS initiates a service instance for this particular service call. After initialization, the instance is routed to the provider of a service for the first activity. In this way, the service instance reaches peer A in our example in Figure 2. In the context of the compound service, then a local call to the service Store Object is performed. After this call, the service instance migrates to peer C in order to execute the service Separate Media Types. This migration is done directly using a peer-to-peer connection between A and C. The next two service calls can be performed parallel. This requires that the service instance splits and navigates separately along the definition paths. The first part of the instance stays at peer C in order to execute the service Analyze Context, while the second part is migrated to peer K or D. Considering the current workload of the peers D and K, the service instance migrates to peer K. Finally, the service Store Features has to be executed on peer A. Since the original service instance was split and distributed on the network, OSIRIS has to synchronize and join these parts before migrating to peer A. After finishing the local call, the compound service is completely executed and the response is sent to the caller.

3.3 Registration of Service Descriptions

In the previous discussion, we have already distinguished between service types and service instances without explicitly mentioning this. In service-oriented architectures, this distinction enables a transparent routing during service execution. In fact, a service type is called. The infrastructure matches a currently available service instances to perform the execution of the requested service type. To enable this behavior — in the literature referred as *enterprise service bus* [4] — all available service types have to be registered in the system. While calls to atomic services just have to be routed through the system, compound services have to be executed by the infrastructure itself as described in the previous subsection. For that, the description of compound services has to be published to the infrastructure. In OSIRIS, this publication is handled by the service Register Service Description.

Whenever a service description is published using this service, it is analyzed and prepared for replication along the peers. Keep in mind that service execution is done in a peer-to-peer fashion relying on locally replicated information. To provide exactly the information that is required to perform a completely peer-to-peer execution, the service description has to be enriched and divided into parts containing all information concerning the execution of one contained service. These parts of service description is then replicated to the peers hosting an instance of the corresponding service type. This strategy allows peer-to-peer execution to have most information already available at the local replication module.

Example 2. Assume that the description of the compound service Insert Multimedia Object is inserted into the system. The service Register Service Description splits the definition into five parts — one part for each activity of the compound service. The part concerning the service Extract Features contains all information about the parameter handling the call of that service. This information is needed by the *service management*

module of the hyperdatabase layer at the peers D, K and B, respectively. In addition, the service navigation module needs to know about the subsequent services. All this information is replicated to the peers D and K immediately after inserting the service description. The peer-to-peer service execution will route every instance of the service Insert Multimedia Object to one of these peers. Consequently, the corresponding part of the service description should be replicated there.

3.4 Registration of Service Instances

A grid infrastructure has to deal with continues changes of the overall system configuration. Service instances may join and leave the system quite frequently. Therefore the infrastructure has to keep track of the current configuration. OSIRIS provides completely transparent service calls by dynamically replicating information about the available service instances to the corresponding grid peers. This replication is based on a publish-subscribe mechanism, which is described in [17]. While the call of a service can occur at any peer, no prior replication can be performed in order to speed up this routing. However, a temporary replication is reasonable since the probability that this information will be needed in the near future is rather high. Beside this temporary replication, information needed during execution of compound services can be predicted by analyzing the service descriptions. In addition, the information about the subsequent services are needed at a particular peer.

Example 3. In the configuration depicted in Figure 3, the peer A holds the replicated information about the current instances of service Separate Media Types since within the compound service Insert Multimedia Object this service must be invoked after the service Store Object, which is provided at peer A. This way a lot of information is replicated along the grid in order to perform and optimize peer-to-peer execution of services. Assume a new instance of the service Separate Media Types is started at peer B and registered via the service Register Service Instance. As a consequence, OSIRIS triggers the replication of additional configuration information to peer B — mainly the part Separate Media Types of the compound service Insert Multimedia Object and information on available instances of Analyze Context. Extract Features is a compound service and thus has no location associated with. Since the (parallel) activities of this compound service are Extract Color and Extract Shape, the location information about these two services are replicated to peer B.

3.5 Stand-Alone Service Execution

The peer-to-peer execution of compound services relies on the transparent routing of service calls. OSIRIS can provide this transparent routing only within the same OSIRIS cell. A cell corresponds to a separate grid environment. Within such a cell, each service of the registered peers is provided to all peers of that cell. As a special case, a complete cell can be installed on one peer. This installation includes OSIRIS core services as well as all services of an application such as ISIS. This peer can also be a mobile device. Albeit local services can be used to realize a stand-alone application, some service types like Extract Features cannot be performed efficiently by the mobile peer. In

order to execute such a service, the peer should join a larger cell to perform this service on a peer having more resources. We also allow peers to join a cell without registering its available services to the cell. In this case, the peer acts as a service user. This kinds of a join can be performed by using a proxy service, which provides a bridge between the two cells. As depicted in Figure 4, two instances of the proxy service can be linked in order to establish a bridge. A proxy service provides a service stub and forwards all service calls to the other cell. OSIRIS routes the call to a real service instance.

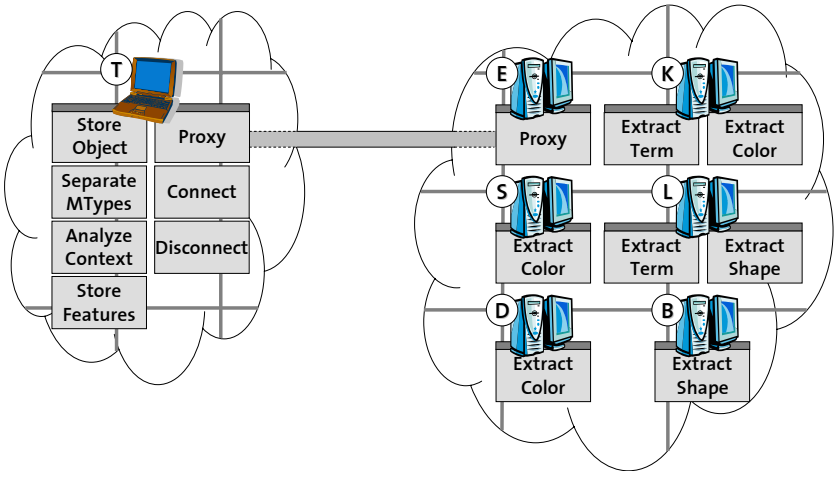


Fig. 4. Proxy Services for Stand-alone Service User

Note that this forwarding of service calls works also for compound services. However, the peer-to-peer service execution cannot use the bridge to migrate. If the mobile device calls a compound service on the remote cell the proxy service will forward the service call and the complete service will be executed at the remote cell. If a local compound service contains an activity for which only a remote service instance is available, the local proxy service provides a stub for that service. In the context of the local grid, the proxy service executes the service. Therefore, the compound service instance will stay at the local device while executing the service at the proxy service. In this way, the concept of a proxy allows for exploiting services of a remote grid cell.

4 Similarity Search on the Grid

Management of multimedia content is highly resource intensive. Indexing digital objects usually requires computationally intensive feature extraction steps. This task is performed whenever new objects are encountered. Thus, it is not time critical and can utilize spare resources. In contrast, content-based retrieval has to be performed in a timely manner as users are interacting with the system. Query evaluation over large media collections is rather time consuming.

Both tasks, indexing as well as retrieval, may benefit from grid architectures in terms of overall throughput by deploying a service on additional peers. However, a *single* query does not benefit from deploying additional search services since the evaluation time is independent of the number of available peers. In order to speed up the evaluation of a single query, several peers have to jointly evaluate the query. To increase this intra-query parallelism, domain-specific knowledge can be used to spread one search task on several service instances. Clearly, intra-query parallelism is closely dependent on the underlying search technology. Therefore, in the following, we first briefly review the principal approach of our search service and then discuss how to optimally distribute the workload over the grid.

4.1 Similarity Search Using VA-File

Content-based multimedia retrieval frequently relies on numeric features extracted from an object's raw content like color histograms, wavelet coefficients, Fourier descriptors, and others. Those feature values are assembled to some high-dimensional feature vector. The domain of these features vectors is called feature space. Similarity between objects is expressed regarding their representatives in the feature space. Usually, proximity is the basic measure of content-based retrieval, i.e., the "closer" the feature vectors are, the more similar the objects are. Thus, similarity search relies on nearest neighbor search which returns the "closest" data point to the user-provided query point with respect to the given metrics to compute the distance between the feature vectors.

Numerous indexing techniques have been proposed to accelerate nearest neighbor search. Not all of them are suitable for nearest neighbor search in high dimensional space (cf. [22]). Our work relies on the VA-file [22] which is an approximation-based technique. Unlike tree-like indexes that employ some sort of clustering, approximation techniques exclusively address each point on its own. That is, a compact signature approximates its exact location in vector space. The VA-file employs a plain quantization scheme that fully partitions each dimension into a number of disjoint intervals. Each interval receives a unique bit code of b bits, permitting to encode an overall number of 2^b intervals. A cross-dimensional bit concatenation forms the point signature, which is stored in a flat file. The choice of b determines the granularity of those signatures.

Query evaluation is conducted in a *filter-and-refine*-algorithm that comprises two subsequent stages. First stage (filtering) scans the signature file in sequential accesses. Point locations are approximated by its signature that permits to compute lower and upper bounds to the precise distance between the query point and the data point. Using these bounds one can determine the objects that cannot be the nearest neighbor. Thus, an approximation may safely be discarded if its lower bound exceeds the smallest upper bound computed so far. This adaptive filtering leaves only a few objects as candidates. Second stage (refinement) visits those candidates in ascending order of their lower-bound-distances. Precisely, the corresponding data points are retrieved in random accesses to compute their actual exact distance to the query point. This algorithm can easily be adapted to evaluate k-Nearest-Neighbor (kNN) queries.

The granularity parameter b has direct impact on the quality of the bounds (i.e., approximation error). A small b leads to a compact signature file but also a higher

number of candidates due to a larger approximation error. Therefore, it is often useful to maintain several index versions concurrently.

4.2 Grid Search Issues

Grid infrastructures allow for transparent workload distribution and system optimization. Ideally, the workload is well distributed over all available search services. The overall execution time of a single query cannot be reduced by this concept. However, the specific application domain allows for further optimization. Nearest-Neighbor search based on the VA-file approach allows for dynamic, horizontal partitioning of indexes [21]. A query can be expressed as a compound service having several paths that are executed in parallel. This enables the infrastructure to further optimize single queries. It introduces intra-query parallelism by distributing parts of the query over a number of grid peers. Figure 5 shows the principle idea of the search parallelization.

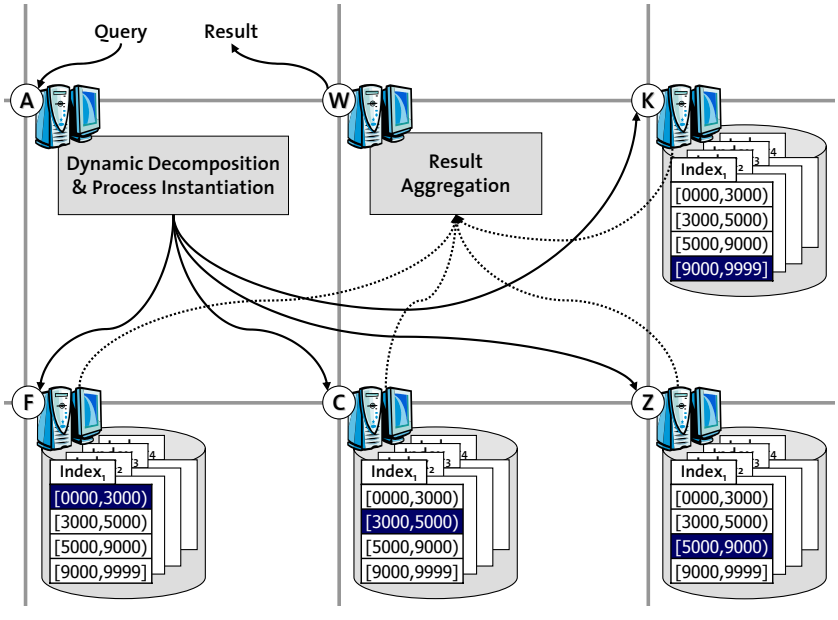


Fig. 5. Similarity Search on the Grid

From the view of the digital library infrastructure, it is transparent whether a query request is evaluated by a single search service in isolation or by several search services in a distributed manner. The search service first receiving a query request is responsible for decomposing the task into several subtasks. These subtasks are assigned to certain grid peers, each of them responsible for a specific share of the overall multimedia collection (horizontal partitioning). In our example in Figure 5, the search services at the peers C, F, K, and Z hold a full replica of a VA-file. Each of those service instances

evaluates the query for its share. The results of these local queries are then combined to compute the global result. That is, the global *k* most similar objects are computed from the *k* most similar objects of each partition.

Note that this aggregation is a rather trivial operation, similar to merging pre-sorted lists. In a vertical partitioning scenario, this would become much more complex: Here, each search service is responsible for a vertical partition of the index, i.e., only for a subset of the features. The evaluation of each partition delivers the *k* most similar objects from the *whole* collection w.r.t. the criteria of this partition. The followup aggregation step corresponds to a classic combiner scenario [8]. This is especially more expensive and requires additional coordination between the various search services for all the partitions [3].

Clearly, search decomposition depends on several parameters, e.g., load and characteristics of the available peers as well as size and range of each index partition. Based on such parameters, the search can be adapted for each query to match the current system configuration as well a dynamic system state changes. In addition, workload balancing can be improved by using knowledge of the application domain. A smart load-balancing strategy can decide which partition of the overall query can be assigned to which peer. To realize this, a special query planner service must be able to access low level system information such as current load and available resources. The query planner service can optimize a query in different dimensions:

- Which peers shall handle a partition of a certain size?
- What index version should be used (for a partition) to optimize the index granularity?

The following subsection shows an analytical way how to solve this optimization problem.

4.3 Load-Based Parallel Query Processing

In order to distribute the workload evenly over all available grid peers, we have to take into account the current state (utilization) of the peers. Since the consistent assessment of the exact state of the peers is not possible, we estimate the currently available resources using a simplified model. Using this information, the query planner can optimize the distribution of the query evaluation by considering (1) workload partitioning on a fixed number of peers and (2) parameterization of the employed indexes on those peers. The overall goal is to minimize query processing times.

The grid infrastructure provides information about all available service instances including the current workload of the corresponding peers. Let m be the number of these peers. Then, t_{Q_i} expresses the latency before query processing can commence on peer i. That is, t_{Q_i} reflects the estimated time consumption of queued processes derived from statistically averaged process execution times. Thus, it reflects the fact that the peers queue processes as the number of concurrent processes is restricted.

As we dynamically distribute the workload on a per query basis, remember that each peer has to hold a complete replica of the index files, including different granularity levels. This guarantees not only full optimization flexibility but also robustness in case

of system failures like peer black-outs. We assign *differently* sized index partitions to the involved peers. The index partition size relates to the peer's varying load factors t_{Q_i} that implicitly result from a peer's process load and hardware configuration. For each partition, we choose the index granularity b independently.

Our approach is a two-stage greedy-style performance optimization. First, we logically partition the index files according to the individual peer load factors. Secondly, we compute the optimal local index configuration of each peer in an analytic approach.

Unsymmetric Index Partitioning. An index file comprising N entries shall be logically partitioned into m shares where N_i is the size of the i-th share, i.e. $N = \sum_{i=1}^{m} N_i$. A badly performing (heavily occupied) grid peer shall receive smaller index shares than a well performing (highly available) peer, $N_i \sim t_{Q_i}^{-1}$. The formula

$$N_i = \frac{N}{t_{Q_i} \cdot \sum_{j=1}^{m} t_{Q_j}^{-1}}$$

gracefully incorporates this plain relation. In that sense, a small load factor t_{Q_i} expresses a high availability of the i-th peer which will receive a large index chunk, and vice versa. Subsequently, we discuss a locally optimal index configuration.

Local Index Configuration. As known from databases, index and scan operations suffer from heavy disk usage. In that sense, I/O is proven to be the bottleneck for similarity query processing as well, leaving CPU times out of consideration. We rest our parameterization approach upon disk parameters like average head seek times (seek_i), rotational latencies ($\mathsf{latency}_i$), and block transfer times ($\mathsf{transfer}_i$). The block size ($\mathsf{blocksize}_i$) is assumed to be fixed. Thus, the number of bits per dimension b_i remains as the only subject for parameterization[1]. Distributed, index-supported query processing must employ four stages:

1. global index partitioning (*decomposition*),
2. local filtering scan through the index shares,
3. local candidate inspections in random lookups, and
4. globally joining the local results to the overall query outcome (*aggregation*)

Second and third steps are subsumed to a plain k-Nearest-Neighbor (kNN) search [22]. Precisely, k is the number of results to be computed over the complete index comprising all chunks. As all results may potentially originate from the same peer, each peer must conduct a kNN search whose results are merged, later on.

As mention before, kNN search on top of a VA-file is based on a filter-and-refine algorithm executing two subsequent stages. Both stages will consume a certain amount of I/O cost [2]. The first stage iterates over the index chunk by sequentially accessing all N_i approximations that are processed into a number of remaining candidates. Each approximation requires a storage space of $d \cdot b_i$ bits where d is the dimensionality of the employed feature. Disk reads are conducted in a block-wise manner. After one initial

[1] Notice, that the i in subscript relates those parameters (N_i, t_{Q_i}, seek_i, $\mathsf{latency}_i$, $\mathsf{transfer}_i$, and $\mathsf{blocksize}_i$) to the i-th peer.

disk seek to set up the index scan, a number of $\lceil N_i \cdot d \cdot b_i / \mathsf{blocksize}_i \rceil$ sequential block accesses occurs. Altogether, this stage spends an estimated processing time of:

$$\mathsf{filtering}_i = \mathsf{seek}_i + \mathsf{latency}_i + \left\lceil \frac{N_i \cdot d \cdot b_i}{\mathsf{blocksize}_i\ [\mathrm{bit}]} \right\rceil \cdot \mathsf{transfer}_i$$

The second stage visits the candidates in ascending order of their *lower-bound*-distances. Actually, refinement visits at least those features whose candidate approximation encodes a *lower bound* distance that falls short of the kNN-distance. Experimental evaluations confirm this assumption to be a fairly accurate estimate. In fact, refinement rarely visits few more features than expressed by this relation between *lower-bound*- and kNN-distance.

In Figure 6, both stages are illustrated by means of a 2-NN search scenario in the plain. It comprises nine VA-style-approximations $N_1, \ldots, N_9$, depicted as rectangles in vector space. The black circle (•) represents the query point q. Those approximations, whose *lower-bound*-distance falls short of the second-smallest *upper-bound*-distance[2] may potentially host one of the two result points. Those "candidate" approximations are depicted as light and dark grey boxes. The outer circle is the second-smallest *upper bound* distance (between q and A_7). The inner circle represents the actual 2-NN-distance, which is initially unknown. Nonetheless, it can be employed to find a lower estimate for the number of those candidates, that actually have to be visited (dark grey boxes). The white circles (○) represent the encoded points, which must be retrieved in random accesses upon inspection.

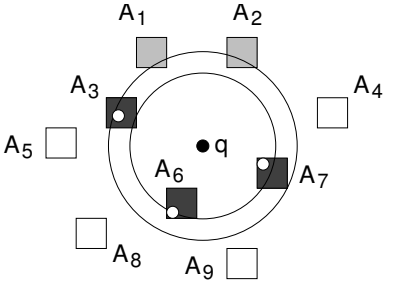


Fig. 6. 2NN Search in $\mathbb{R}^2$

Roughly spoken, we need to find estimates for (1) the kNN-distance and (2) the proportion of approximations whose lower bound falls short of the kNN-distance. The kNN-distance is the k-th smallest exact distance between a query point and a (data) point. Both estimates can be provided my means of formal stochastics, which rests upon two basic assumptions. First, we exclusively employ L_2-norm-based distances. Other metrics can be mapped into Euclidean space by means of the *Fastmap* [9] transformation. Secondly, feature values are assumed to be uniformly distributed in $[0, 1]$.

[2] The *upper-bound*-distance is the largest *possible* distance between the query point and the approximated point.

Alternative distance distributions must be statistically sampled from the feature data. However, the assumption of uniformity suffices our needs within this index configuration approach. In [1], we elaborate on stochastic distributions of L_2-distances in high-dimensional space. Precisely, we provide several distribution functions characterizing L_2-distances between uniformly distributed points, where $\Phi(x)$ is the Gaussian distribution function, $N(0, 1)$:

$$F_{\|p-q\|_2}(x) = \mathrm{P}(\|p - q\|_2 \leq x) = \Phi\left(\frac{x^2 - d/6}{\sqrt{7d/180}}\right)$$

and *lower-bound*-distances between a query point and a VA-approximation shape (hyperrectangle positioned at discrete locations):

$$F_{\text{cube}}(x) = \mathrm{P}(\textit{lower-bound}\text{-distance} \leq x) = \Phi\left(\frac{x^2 - \mu}{\sigma}\right)$$

with $\mu = \frac{d}{6}(w_i - 1)^2$, $\sigma = \sqrt{\frac{d}{180} \cdot (-11w_i^4 + 20w_i^3 - 16w_i + 7)}$, and $w_i = 2^{-b_i}$. On top of $F_{\|p-q\|_2}(x)$, we can compute the expected kNN-distance as follows:

$$\mathsf{kNN}_i = F^{-1}_{\|p-q\|_2}\left(\frac{k}{N_i}\right) = \sqrt{\sqrt{\frac{7d}{180}} \cdot \Phi\left(\frac{k}{N_i}\right) + \frac{d}{6}}$$

On average, k feature vectors (data points) will lie closer to the query than kNN_i. Figure 7 depicts the analytically and experimentally determined 5-NN-distance for $N_i = 10000$ and $d = 30, \ldots, 100$. The experimental validation proves our analytic estimation to be feasible. For a rising dimensionality, both graphs converge towards one another.

The proportion of (candidate) approximations whose *lower bound* distance lies below kNN_i can easily be expressed by $F_{\text{cube}}(\mathsf{kNN}_i)$. Each of the corresponding data points must be looked up in a random disk access which results in an overall effort of:

$$\mathsf{refinement}_i = N_i \cdot F_{\text{cube}}(\mathsf{kNN}_i) \cdot (\mathsf{seek}_i + \mathsf{latency}_i + \mathsf{transfer}_i)$$

In merging both cost terms into a global cost formula, we obtain:

$$\mathsf{cost}_i = \mathsf{filtering}_i + \mathsf{refinement}_i \rightarrow \min$$

That is, $\mathsf{cost}_i \rightarrow \min$ represents a nonlinear optimization problem. We illustrate the optimization by means of a specific hardware example which is given by two grid peers with widely diverging load factors of $t_{Q_1} = 10$ and $t_{Q_2} = 1000$. The first peer is a highly available, well-performing server equipped with fast storage (Seagate Cheetah: $\mathsf{seek}_1 = 3.5\,\text{ms}$, $\mathsf{latency}_1 = 2\,\text{ms}$, $\mathsf{transfer}_1 = 0.08\,\text{ms}$ per 8 kByte block). The second peer is an older laptop computer ($t_{Q_2} = 1000$) with dedicated mobile storage components (Hitachi Travelstar: $\mathsf{seek}_2 = 15\,\text{ms}$, $\mathsf{latency}_2 = 7.1\,\text{ms}$, $\mathsf{transfer}_2 = 0.18\,\text{ms}$ per 8 kByte block). Notice, that block sizes are assumed to be identical, i.e. $\mathsf{blocksize}_1 = \mathsf{blocksize}_2 = 8\,\text{kByte}$.

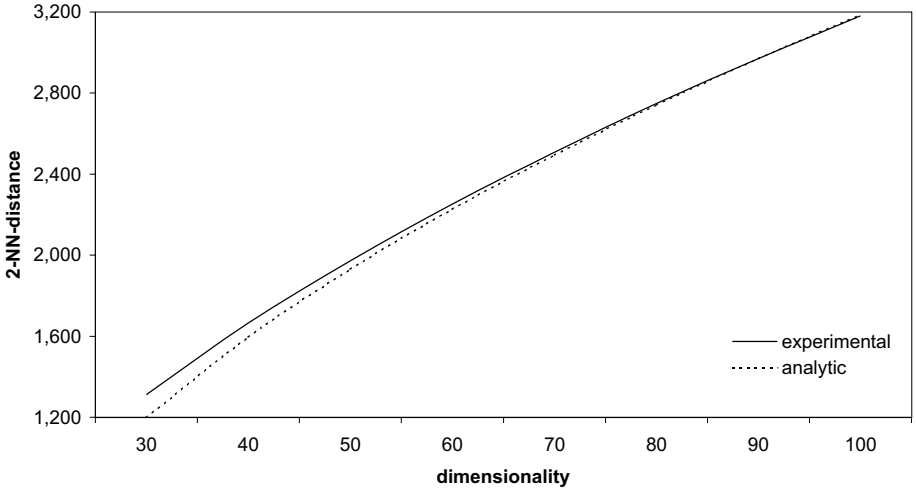


Fig. 7. Experimental and Analytic 5-Nearest-Neighbor Distance

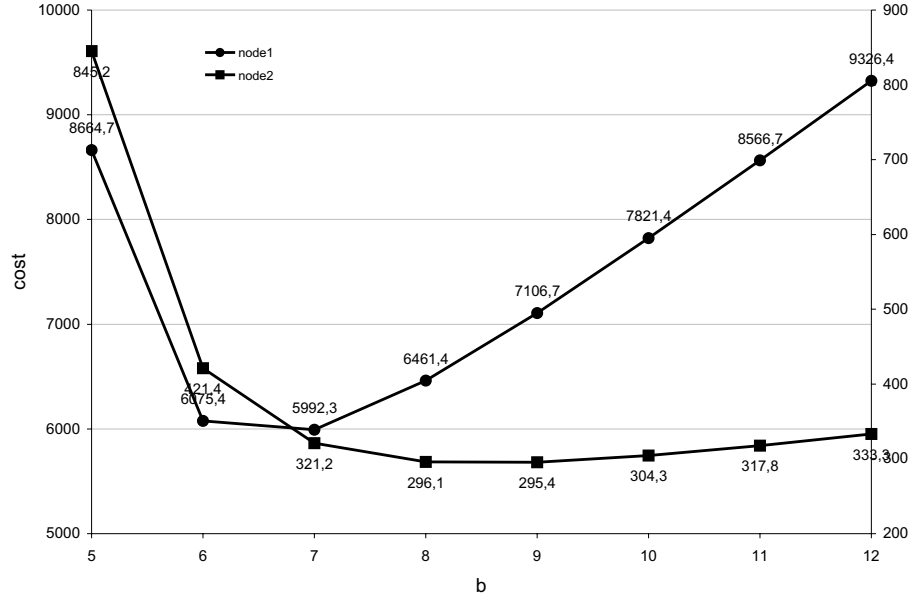


Fig. 8. Retrieval Cost for Different Index Configurations on Both Peers

The overall index file comprises $N = 5 \cdot 10^6$ entries, where the accumulated dimensionality is $d = 128$. Query processing shall be distributed onto both peers (i.e. $m = 2$). First, we logically partition the index file resulting in two shares of

$$N_1 = 4950495 \quad \text{and} \quad N_2 = 49505$$

entries. Next, we determine the estimated query cost for various bit ratios b on both peers. Figure 8 depicts the retrieval cost of both peers under different index configurations. We have scaled the bit ratio in $b = 5, \ldots, 12$ and computed the estimated time consumption (cost_1, cost_2). Notice, that cost_1 refers to the left y-axis whereas cost_2 refers to the right y-axis. On the first peer, the minimal cost of $\mathsf{cost}_1 = 5992.3$ ms accumulates for $b_1 = 7$. On the second peer, minimal cost of $\mathsf{cost}_2 = 295.4$ ms is achieved at $b_2 = 9$.

In this way, this plain scenario delivers further insights into our approach. First, (1) the optimal index configuration may vary on differently equipped peers. Peers should, therefore, maintain different index configurations simultaneously and pick one for query processing at runtime. Secondly, (2) different load factors implicitly translate into different query processing times. Notice, that this effect is fully intentional, as larger load factors delay the beginning of query processing. Smaller processing times may partially compensate this effect.

We are currently underway to merge this greedy-style optimization approach into a *one-step* multi-dimensional, non-linear optimization problem. That is, we feed in all load factors, disk parameters, communication cost, and partitioning/merging effort into an integrated formalization targeted at (1) the appropriate number and location of participating peers, (2) the optimal index chunks and (3) the index parameters that yield minimal cost. Clearly, this enhanced approach will add further complexity from the implementation point of view. For instance, a large number of parameters will be involved, if lots of grid peers participate. Solving multi-dimensional optimization problems is non-trivial and requires appropriate heuristics to converge quickly. Further research has to be conducted in order to obey tight time constraints in query optimization.

5 Conclusions

This paper showed how service-orientation helps to describe and implement complex digital library applications like ISIS as compositions of services. A hyperdatabase infrastructure like OSIRIS, which combines service-orientation with peer-to-peer and grid computing, is then able to exploit the knowledge about the services and their composition to execute them in an optimal fashion, even under changing workloads and system configurations. Following the idea of grid computing, the execution of service calls respects parameters like the workload of the peers to dynamically select the best fitting service instance. Besides, OSIRIS is able to replicate service instances on demand on any peer of the grid, and thus to optimize the utilization of the given resources. The peer-to-peer style of service navigation avoids that the navigation becomes a bottleneck of the overall system, and thus provides the basis for a scalable infrastructure. The current implementation of ISIS atop OSIRIS demonstrates this very nicely.

References

1. S. Balko. *Foundations, development, and Evaluation of an efficient Approximation Technique for Nearest-Neighbor-Queries in High-Dimensional Vektor Spaces*. Dissertationen zu Datenbanken und Informationssystemen (DISDBIS 86). Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2004. Doctoral Thesis, (In German).

2. S. Balko, I. Schmitt, and G. Saake. The Active Vertice Method: A Performant Filtering Approach to High-Dimensional Indexing. *Data & Knowledge Engineering*, 51:369–397, 2004.
3. K. Böhm, M. Mlivoncic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *VLDB 2001, Proc. of the 27th Int. Conf. on Very Large Data Bases, September 11–14, 2001, Roma, Italy*, pages 211–220, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
4. D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.
5. M. G. Curley. Peer-to-Peer Computing Enabled Collaboration. In *Proc. of the Int. Conf. on Computational Science, ICCS 2002*, pages 646–654, 2002.
6. A. Dimai. Spatial Encoding Using Differences of Global Features. In I. K. Sethi and R. Jain, editors, *Storage and Retrieval for Image and Video Databases V, February 8–14, 1997, San Jose, CA, USA*, volume 3022, pages 352–360. SPIE Proceedings Series, 1997.
7. ETH Database Research Group. OSIRIS: an Open Services Infrastructure for Reliable and Integrated process Support. `http://www.osiris.ethz.ch`.
8. R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.
9. Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, 22.-25. Mai 1995*, pages 163–174. ACM Press, 1995.
10. I. Foster, C. Kesselmann, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Global Grid Forum 2002*, 2002. `http://www.gridforum.org/ogsi-wg`.
11. W. Niblack, R. Barber, W. Equitz, M. Flickner, E. H. Glasman, D. Petkovic, P. Yanker amd C. Faloutsos, and G. Taubin. The QBIC Project: Querying Images by Content Using Color, Texture, and Shape. In W. Niblack, editor, *Storage and Retrieval for Image and Video Databases, 31 January–5 February 1993, San Jose, CA, USA*, volume 1908, pages 173–187. SPIE Proceedings Series, 1993.
12. H.-J. Schek, H. Schuldt, C. Schuler, and R. Weber. Infrastructure for Information Spaces. In *Advances in Databases and Information Systems, Proc. of the 6th East-European Symposium, ADBIS'2002, Bratislava, Slovakia, September 2002*, volume 2435 of *Lecture Notes in Computer Science*, pages 23–36, Berlin, 2002. Springer-Verlag.
13. M. Schmid, F. Leymann, and D. Roller. Web Services and Business Process Management. *IBM Systems Journal*, 41(2):198–211, 2002.
14. H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and Isolation in Transactional Processes. *ACM Transactions on Database Systems*, 27(1):63–116, March 2002.
15. C. Schuler. *Distributed Peer-to-Peer Process Management — The Realisation of a Hyperdatabase*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, ETH-Zentrum, CH-8092 Zurich, Switzerland, 2005. (In German).
16. C. Schuler, C. Türker, H.-J. Schek, R. Weber, and H. Schuldt. Scalable Peer-to-Peer Process Management. *International Journal on Business Process Integration and Management*, 2005. *To appear*.
17. C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek. Peer-to-Peer Process Execution with OSIRIS. In *Service-Oriented Computing — ICSOC 2003, First International Conference, Trento, Italy, December 15–18, 2003, Proceedings*, volume 2910 of *Lecture Notes in Computer Science*, pages 483–498, Berlin, 2003. Springer-Verlag.

18. M. Stricker and M. Orengo. Similarity of Color Images. In W. Niblack and R. C. Jain, editors, *Storage and Retrieval for Image and Video Databases III, San Jose, California, February 9–10, 1995*, volume 2420, pages 381–392. SPIE Proceedings Series, 1995.
19. C. Türker, K. Haller, C. Schuler, and H.-J. Schek. How can we support Grid Transactions? Towards Peer-to-Peer Transaction Processing. In M. Stonebraker, G. Weikum, and D. DeWitt, editors, *Proceedings of the Second Conference on Innovative Data Systems Research, CIDR 2005, January 4-7, 2005, Asilomar, CA, USA*, pages 174–185, 2005.
20. G. Tzanetakis and P. Cook. Audio Information Retrieval (Air) Tools. In *Proc. Int. Symposium for Audio Information Retrieval, ISMIR 2000, 23-25 October 2000, Plymouth, Massachusetts, USA*, 2000.
21. R. Weber, K. Böhm, and H.-J. Schek. Interactive-Time Similarity Search for Large Image Collections Using Parallel VA-Files. In *Research and Advanced Technology for Digital Libraries, Proc. of the 4th European Conf., ECDL 2000, Lisbon, Portugal, September 18–20, 2000*, volume 1923 of *Lecture Notes in Computer Science*, pages 83–92, Berlin, 2000. Springer-Verlag.
22. R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *Proc. of the 24th Int. Conf. on Very Large Data Bases, VLDB'98, New York City, August 24–27, 1998*, pages 194–205, San Francisco, CA, 1998. Morgan Kaufmann Publishers.
23. R. Weber, H.-J. Schek, J. Bollinger, and T. Gross. Architecture of a Networked Image Search and Retrieval System. In *Proc. of the 8th ACM CIKM Int. Conf. on Information and Knowledge Management, November 2–6, 1999, Kansas City, Missouri, USA*, pages 430–441, New York, 1999. ACM Press.

StreamOnTheFly: A Network for Radio Content Dissemination

László Kovács[1], András Micsik[1], Máté Pataki[1], and Robert Stachel[2]

[1] MTA SZTAKI, DSD,
Lágymányosi utca 11, H-1111 Budapest, Hungary
{laszlo.kovacs, micsik, mate.pataki}@sztaki.hu
[2] Team Teichenberg,
Operngasse 22, A-1050 Vienna, Austria
robert@teichenberg.at

Abstract. A distributed digital library has been designed and implemented for the support of community radios. This framework, developed by the StreamOnTheFly IST project of the EU, provides a common background for preparation, archival, exchange and reuse of radio programs and supports radio personalization. The architecture is based on a decentralized network of software components using automatic metadata replication in a peer-to-peer manner. This approach combines the principles and practice of OAI (Open Archives Initiative) with the peer-to-peer networking paradigm, and extends usual content dissemination with the aggregation of use statistics and user feedback. The network also promotes social self-organization of the community and a new common metadata schema and content exchange format.

1 Introduction

Nowadays many community radio stations (non-profit, independent stations) exist, but there is little cooperation among them. One of the causes for this is the lack of technical support for easy exchange of radio programs. A significant part of the programs produced at community radio stations are of high quality and worth to be broadcast more than once, and listened to by a larger number of people in a wider geographical range.

The StreamOnTheFly IST project [17] was set out to find and demonstrate new ways for archival, management and personalization of audio content. In the project Public Voice Lab (Austria) and MTA SZTAKI DSD (Department of Distributed Systems at the Computer and Automation Research Institute of the Hungarian Academy of Sciences) were development partners and Team Teichenberg (Austria) provided connections to community radio stations and associations.

The project team envisioned and implemented a network of digital archives and helper tools for radio stations. The general aim of this network is to foster alternative, self-organized and independent media by new ways of content distribution, international community based presentation platforms and many added values for local production and organizational work. In the following an overview on the world of radio is

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 188–206, 2005.

given, the concept and the current status of the StreamOnTheFly network are presented, and the architectural design of the network is explained in details.

2 Community Radios in Europe

During the past 30 years community radio stations developed in most countries of Europe, enriching the existing media world with a "third sector", next to public broadcasting as the first and commercial media as the second. Community radios are non-profit-oriented, are open to the general public and have a local or regional range. They provide access to radio production facilities for organizations, groups and individuals who aim to make their own radio shows. There are over 1500 non-profit stations in the whole of Europe, an estimated third of them is provided with a full and permanent 24 hour broadcasting license on their own frequency.

The global umbrella organization AMARC defines community radio as *"which is for, by and about the community, whose ownership and management is representative of the community, which pursues a social development agenda, and which is non-profit."* The success of a community radio cannot be assessed by audience ratings and revenues. The important factor is the effect a broadcast series may have on the community targeted. A radio show in a language spoken by a small minority will be highly appreciated by this group of people, although they would not be considered to be a market big enough to be targeted by other media.

There are a number of projects existing that aim to foster production of programs for an international audience that can be aired all over Europe or even wider. With AMARC's "Radio voix sans frontieres" (Radio without frontiers), more than 30 stations in Europe share a full day of live broadcasting once a year on the UN anti-racism day. Many exchange projects are oriented around themes of global interest, like migration, sustainable development or Human Rights.

What keeps program exchange from becoming a day-to-day routine, is the technical and structural thresholds. The existing program exchange platforms on the Internet are not connected to each other and do not follow a common metadata scheme and user interface, which makes it harder to search and offer content on an international scale, as every program needs to be uploaded and indexed in different standards for every country. The user might not even be able to understand the language a sharing platform's website is written in.

Building on the experiences made at Orange 94.0 in Vienna and in other community radio stations in Austria, we researched the requirements of producers and editors for easy access to online archiving and content sharing. A common metadata scheme and the ability to search in all databases at once are main critera, the other is to provide convenient access to the repositories. Participating in the exchange of audio content should be as easy as sending an email or using a search engine, with program recording and uploading being done automatically.

Only an easy-to-use system that encourages more people to enter their day-to-day programs into an international platform will create a critical mass of content that can be searched and re-used in local production. Such a platform would certainly help to strengthen the position of community radio and foster international cooperation in Europe.

3 Overview of the StreamOnTheFly Network

The project aimed to create a network of radio content that develops in an organic and dynamic way, in decentralized structures that build synergies on an international, community-based level. The content is offered by local producers or stations to the network, but it's the international communities who choose which content is presented and how it is presented. International publishing on the net is not a form of archiving or recycling anymore, but a motivation to communicate with a whole new group of people.

The StreamOnTheFly network consists of the following basic elements: nodes, radio stations and portals (Fig. 1). These components are introduced in more details in the following subsections. Nodes together can be seen as a distributed digital library specialized for audio content. Stations send content to the nodes for archival, and portals present thematic selections of the content of nodes. The StreamOnTheFly network tries to serve the whole lifecycle of radio programs, starting with the creation and ending with various (re)uses and discussions of the programs.

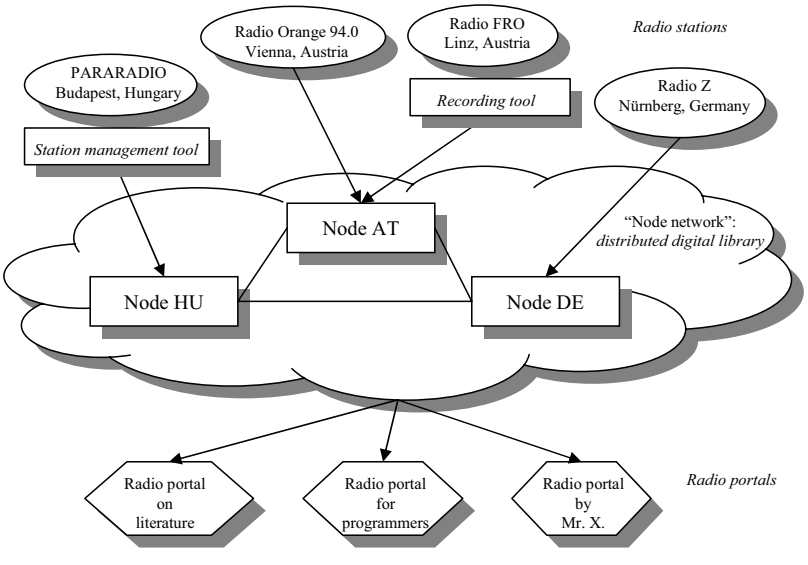


Fig. 1. Example for a StreamOnTheFly network

For example, Orange 94.0, a community radio in Vienna, may send some of its broadcast material on the node running in Austria. These programs are archived at the node and become accessible for the whole StreamOnTheFly network for a given period of time. A German radio may find a program on the Austrian node that they want to rebroadcast locally. They find the contact for this program, in this case Radio Orange and ask for their permission for rebroadcast. If there is a portal in France for the Turkish people, its portal editor may include some of the Turkish programs of Radio Orange in

the recommendation list. Users may also go directly to a node, query/browse the whole content of the network, and listen to selected programs.

3.1 Nodes

Nodes implement the basic digital library infrastructure for the network. Each node hosts a set of radio stations. For each station the node archives radio programs with a rich set of metadata and other associated content. For example, the content may be present in several audio formats, and photos may be attached to the audio content.

Users get transparent access to the content of the whole network at any node. They can browse through radio stations, topic trees or search in the archive. Continuing our previous work [14], an advanced query facility is integrated which enables users to construct and save complex queries. Queries are built step by step by composing atomic expressions (e.g. broadcasted before year 2004) and joining them with logical AND/OR. In atomic expressions the query target can be any field of the metadata scheme. The possible query syntax offered for the user changes according to the type (a string, a date, a number or an enumeration) of selected metadata element in the expression. Query expressions can be saved with a name in the users' profiles, and later reexecuted easily.

Interesting items found in search results or during browsing can be collected into a playlist. Playlists can also be used as a kind of personalized radio. A user may listen to all or some of the programs in her playlist successively. Several audio formats are supported for listening (MP3, Ogg/Vorbis) in different bit rates or quality.

The node gathers and forwards information about radio programs back to their creators. This involves the collection of various statistics (e.g. audio file downloads), the collection of comments, references and rating information.

Editors have a comfortable "console screen" where they get an overview about all of their programs, they can publish new programs and manage existing ones. Access privileges can be controlled on various objects, for example insert, delete and modify privileges can be defined for stations, series or programs.

3.2 Stations

A station management tool which connects radio stations to the network (Fig. 2) has been developed by the project. This tool provides an easy way to feed a node with radio programs. Furthermore, it helps creting the schedule of the station, and leads the creators by the hand before and after the broadcast of a radio program. Authors may add metadata to their programs and publish them on a node.

The station management tool provides editors and other users of a radio with proper access privileges. Programs and series can be scheduled for broadcast by the creators or editors, and station managers control the weekly broadcast plan. This tool provides most of the basic software support required for a radio.

A recording tool can be used when no pre-recorded audio is available for a radio program (e.g. live shows). This software records the broadcasted audio of a station, and automatically cuts the stream into individual radio shows.

The desktop upload tool is under development, it will enable creators to assemble metadata and audio files into a program package and send it directly to the node from their personal computer.

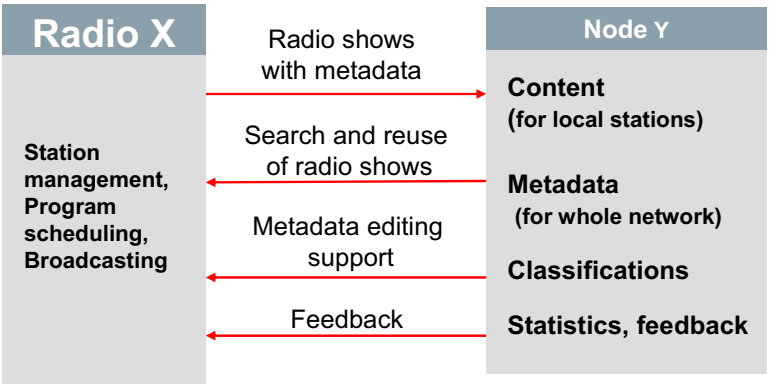


Fig. 2. Roles and cooperation of stations and nodes

3.3 Portals

In the StreamOnTheFly project a radio portal offers a selection of radio programs in a custom design (Fig. 3). Portals may serve communities or geographical regions by selecting relevant content from the whole network. Additionally, a portal can offer reviews, opinions or pictures about the selected programs. Visitors may also rate, comment and discuss programs. Comments and ratings made on portals are forwarded to the nodes, and the comments are delivered to the creators of the program via e-mail (Fig. 4).

The StreamOnTheFly portal is a specialized CMS, thus the main activities are content and design management. Design is controlled by a special HTML based WYSIWYG editor, which can be used from any web browser, without installing any extra programs, components or plugins. The user arranges a table as the skeleton of the portal page, where each cell is filled either with design elements or with content. The editor uses rows as basic screen layout elements, where any number of cells can be inserted in each row. Each cell has its own type; so far the following five have been implemented: Text/HTML, Picture, Spacer, Query, Program list. When the user clicks on a cell in design mode, current cell settings appear in a popup window and can be changed according to the desired layout. This solution requires minimal Javascript capabilities available in all new and most of the old web browsers. Using this method, users without HTML knowledge can also build elegant portals, while trained web editors may insert HTML pieces into the cells and use CSS as well.

The two cell types for presentation of content are Query and Program list. The first one is a saved query expression, which is normally constructed using the advanced query interface of the node and then it is sent from the node to the portal. With this method one can make a self updating portal page, where the newest programs are automatically added to the page. The second option is to use a Program list, which is a static collection of selected programs. The number of query and program list boxes on a page is not limited, so the editor can place chosen content on pages rather freely. Program

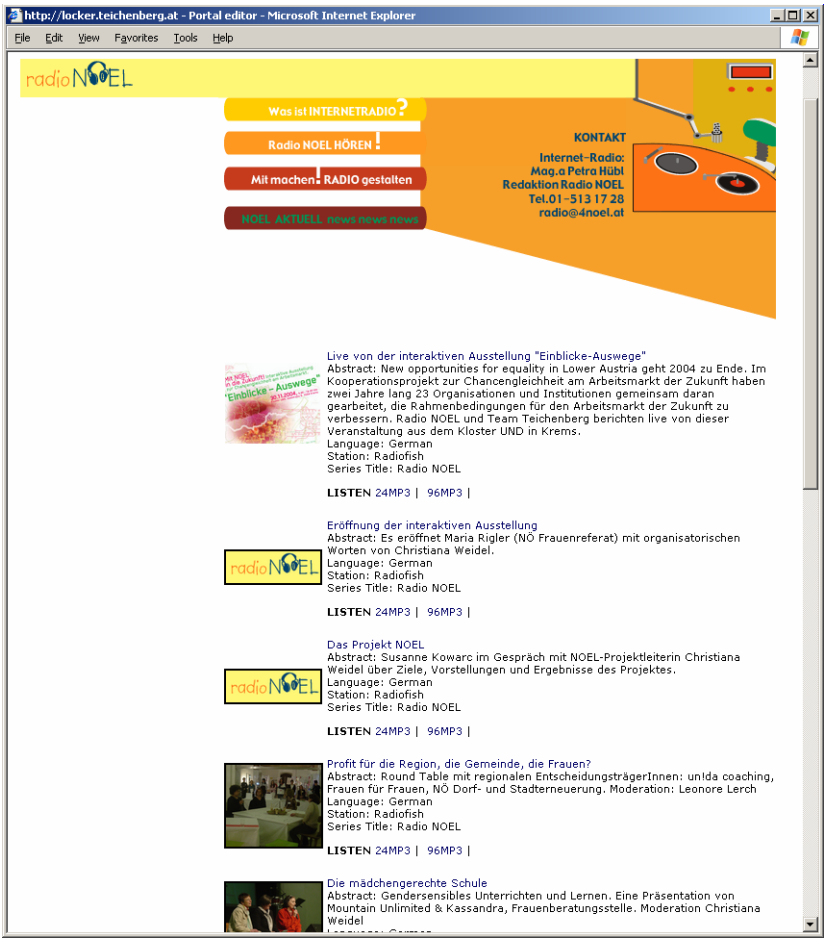


Fig. 3. One of the portals in the StreamOnTheFly network

references used in program lists, similarly to queries, can be sent from the node to the portal.

The portal editor may create a short introductory text for a program (often called a teaser in jargon), which is visible in program lists. A longer review with attached images or documents can also be given for a program. Clicking on a program in a list brings up the program page, where the review and all the other details about the program both from the node and the portal are displayed, including metadata, listen/download option, ratings, statistics and comments by users.

3.4 Content and Metadata Format

At the time of the design phase, the project team found many emerging activities for attaching metadata to multimedia content. These solutions provide schemas for metadata

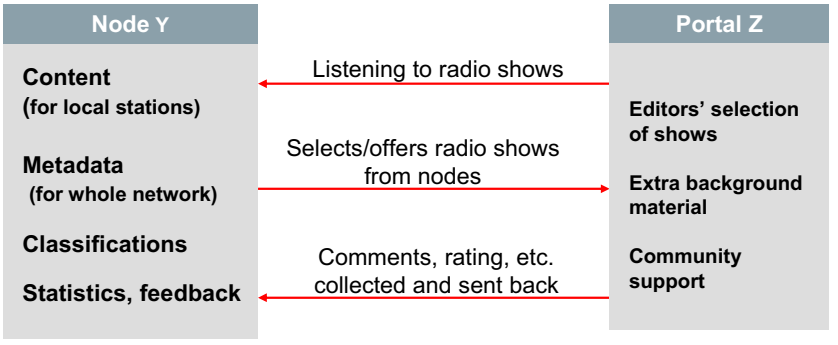


Fig. 4. Roles and cooperation of portals and nodes

and methods to attach or embed metadata into the content (e.g. SMPTE, AAF, MPEG-7 [19,6,12]). Schemas often contain an object hierarchy for metadata description and controlled vocabularies for the values of object properties. We found that these activities were overlapping and they all targeted large radio and television stations with a highly complex format and structure. Community radios expressed on various forums that the use of these metadata formats is neither feasible nor practical for them and they would welcome a simpler and more accessible recommendation for metadata usage.

The European Broadcasting Union (EBU) prepared a recommendation on a core metadata set for radio archives [9] based on Dublin Core [1]. This document defines how to use the Dublin Core elements in the area of radio archives. The StreamOnTheFly project participated in the launch of an initiative called Shared Online Media Archive (SOMA). The SOMA Group (consisting of AMARC, OneWorld, CMA and Panos) published the SOMA Metadata Element Set [3], which builds upon the EBU Core Metadata Set.

The SOMA metadata set became part of the new exchange format called XBMF (Exchange Broadcast Binary and Metadata Format) defined by the project. XBMF is designed for transfer and archival of radio programs coupled with metadata. This is a simple container format, in which metadata (in XML format), content files and other associated files can be packaged together. XBMF may contain the main content in several audio formats and also any other files related to the content (e.g. photos, documents, logos). XBMF can easily be applied for video as well.

Technically, an XBMF file is a compressed archive of a fixed directory structure, where naming conventions and placement of files identify the data they convey. The implementation of software for handling XBMF files is straightforward, and can be based partially on existing open software. The content of an XBMF archive can be examined on every computer platform with standard utilities.

The metadata part of XBMF can be seen as an application profile of Dublin Core for radios. This profile is constructed from Dublin Core elements and the StreamOnTheFly element set according to the rules implemented for metadata schema registries in the CORES project [13].

4 Architectural Aspects of StreamOnTheFly

The most important requirements to form the base of our implementation decisions were:

- no organized hierarchy in the operation of the network can be expected,
- most of the services in the network should be decentralized and without a single point of access,
- user access to radio shows should be fast and effective (focusing on search facilities),
- the amount of network communication for each node should be controllable.

These properties are easily deducible from the usual characteristics of community radios. These radios act very independently, and they operate on a rather low budget. There is little chance to establish rigid rules or hierarchies of cooperation among them. Based on the above facts, a working solution in case of a very loose cooperation was sought. The criterion of controllable network load originates in the financial status of these radios and radio associations. They need to plan the cost of their Internet connections and hardware equipment well in advance and from a low budget. Despite all these restrictions, searching and browsing should be global and effective, otherwise users will not change to this new platform.

The requirements clearly drove us towards a solution where the paradigms of peer-to-peer communication and component-based architecture are applied.

The architecture can be seen as the evolution of the OAI approach [15] using peer-to-peer communication. The principle of open archives proved to be useful and it is becoming widespread for connecting repositories worldwide. Basically the principle is that valuable content remains under the control of the creator or publisher, while less valuable metadata is made available through a well-defined protocol.

In the StreamOnTheFly network metadata is diffused using peer-to-peer communication in a push manner (Fig. 5). Content metadata for the whole network is available at all nodes of the network, which creates the possibility of fast searching and advanced query interfaces based on metadata. The user interface functionality (including searching, browsing, listening) is also available on all nodes of the network. In this situation the traditional data provider, harvester and service provider components are all present at each node of the network.

The push method used for metadata replication was considered better than the pull method used in OAI, as fast distribution and notification of new radio programs is better achieved this way. The fact that a radio show could be quickly and surely removed from the network, also voted for push-style communication. More details on metadata replication can be found in section 4.2.

Replication of content was considered as not feasible due to the relatively big size of broadcast quality audio material (the storage requirement for one hour audio material was calculated as 65MB) compared to the storage capabilities of node operators. The size of metadata for a single radio program is usually between 2kB and 25kB which presents no problem for replication on all nodes.

Nodes in the StreamOnTheFly network have to meet certain capabilities: they need a permanent, wide bandwidth, reliable connection to the Internet so that they can handle

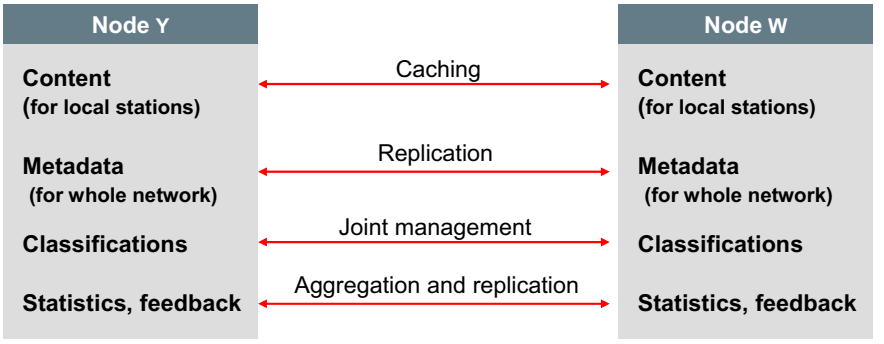


Fig. 5. Cooperation of nodes

many simultaneous audio streams and replication traffic. They also need higher processing power and more storage space than average workstations. Professional maintenance of the software (e.g. periodical backups) is also beneficial in this case. Typically, StreamOnTheFly nodes are provided on a higher granularity than per radio. Nodes offer their services to radios as a national, regional or ad hoc cooperation with or without commercial or other purposes.

4.1 Managing Network Topology

The project is planned to establish some dozens of nodes in Europe. Dynamically changing and dynamically optimized topologies are dispensable with this small number of nodes. On the other hand, a decentralized authentication of peer-to-peer connections is a basic requirement. The project decided on using a static network topology where these static connections provide the way for authentication and a control for expansion as well. Nodes may exist in themselves, without being part of any StreamOnTheFly network. Usually, this is the way how new nodes operate at first times. When a node wants to join a StreamOnTheFly network, personal agreements are made with the operators of the prospective neighbours of that node, and new peer-to-peer communication channels are configured connecting the node to the rest of the network. This method has one more advantage in this situation: personal contacts are established among the operators of nodes, which may enforce some kind of cooperation in the network. Each node has choices for the level of participation in the network: for example, a single connection to the network is less reliable but needs less network traffic, while several connections to various parts in the topology provides more fault-tolerance and content will be faster updated.

If the communication between nodes is configured to be one way, a node gets all content from the network, yet its content remains local. On the contrary, another node might choose to live with its own content exclusively, still it is able to disseminate its content to the whole network. This extreme situation has already been required by one of our partners.

In the example topology of Figure 6 nodes A, C and D form one core and node group G forms another one. These groups are connected with a single connection. Breaking

this connection results in two separate StreamOnTheFly networks. Node E uses the network for the propagation of its own content, thus makes it available at any other node in the network. It does not receive replicated data, so its user interface shows only local content. Node B uses the network to get more content. All network content is available at node B, yet its own content remains hidden for other nodes.

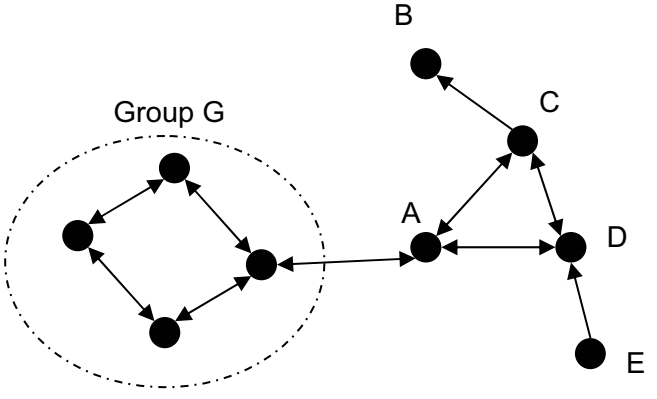


Fig. 6. Example for network topology

Standard operations with the network topology are examined in the following. When a new node joins the network by establishing a new connection, all replicable contents are exchanged through that connection until both ends of that connection have equal metadata database. Then new content from the new node begins to spread over the network. This method works also in the case when two node groups get connected.

Leaving the network is equivalent to the elimination of all connections to other nodes. Upon discontinuing the connections, the metadata of the network could remain available on the node that left the network, but this may lead to data inconsistency over time. Normally, after the elimination of the connection, all the nodes in the local registry are checked if they are reachable. This can be calculated using the table of active connections which is also replicated at all nodes. For unreachable nodes a deletion message is propagated through the network, which results in a cascading deletion of all data belonging to those nodes. This operation also works when two node groups get disconnected.

Finally, optimization for better connectedness can be done manually by configuring more connections between nodes.

It can be seen that node operators govern the network. They decide on accepting new radio stations, on establishing and eliminating network connections. Those unsatisfied with the administration of a node may change to another node.

In case of an editor or a radio station abuses the network, the hierarchical permission system of nodes makes it possible for station managers or node operators to remove unwanted content within their station or node, respectively. What happens when a node abuses the network? Banning a node from the network has to be based on a consensus

of node operators, so that all connections to that node are ceased. The misbehaving node should not be able to negotiate new connections with any other nodes in the network. Personal agreements and exchange of opinions have an important role in the life of a StreamOnTheFly network.

Manual maintenance of peer-to-peer links in the network balances the evident drawbacks of less efficient communication topologies with social management and control, which is in some sense forced by the implementation of the network. Elaboration of this aspect is continued in section 4.5.

4.2 Replication Between Nodes

The software of StreamOnTheFly nodes is built upon an object repository, containing three main types of objects. The first type is local object which does not contain information of global interest, and therefore is not replicated in the network. The other two types of objects are replicated over the network, the difference is that there are simpler objects containing basic data, and complex objects which aggregate simple objects and refer to other complex objects.

Examples for local objects include user data, as it was decided not to replicate the user identities and their preferences in the network. The reasons for not doing so were partly fear from having the network flood with data of one-time users and partly to entitle the nodes to implement their own usage policies. User authentication is implemented as open API and sample wrappers are provided to connect various user management solutions to the node. One example wrapper provides basic user management in itself, while other wrapper delegates user management to the Typo3 portal engine.

Examples for replicated simple objects include a single audio file of a radio show, or a metadata field and value describing the radio show. Complex replicated objects are for example radio shows and series. The object storage is implemented using two parallel storage methods, one in PostgreSQL database and one in plain file system.

Each replicated object has a globally unique identifier in the network. This identifier is composed of the node identifier where the object was created, an object type designator and a local identifier part. Additionally a current home node identifier is known for each object, this is the only node (the owner) which is allowed to modify the content of that object. An object stamp which is incremented at every modification of the object is also stored. Replication is implemented based on these administrative data. When an object is modified it is placed on the replication queue with the list of the current neighbour nodes as addressees. Each neighbour node is contacted periodically and the objects on the replication queue are sent to there. When a node receives new or modified objects from one of its neighbours, it places these objects also on the replication queue, and forwards them to all other neighbours, except the one who sent it. Modifications may arrive to a node from more than one of its neighbours. Based on the stamps it can be decided which object has a newer state than the other, so this kind of redundancy in replication cannot cause problems in data integrity, rather it makes replication safer and helps to detect network problems.

Deleting an object on the node triggers the creation of a new object which contains the fact of deletion. This so called deletion object is replicated over the network, and whichever node it reaches, the replica of the given object is deleted there as well.

All network communication between nodes is based on XML-RPC calls, a very simple way of remote procedure call with XML data as input and output. XML-RPC is an ancestor of SOAP, and migration to SOAP is very easy. This migration is planned in the near future as SOAP is now widely accepted.

4.3 Extensions to Data Replication

The StreamOnTheFly network uses controlled vocabularies (or simple thesauri with other words) for classification purposes. These vocabularies can be used to define the genre of the show (e.g. interview, comedy, news), the roles of creators and contributors (e.g. speaker, producer, composer) and the topic of the show (in this case the controlled vocabulary is in the form of a subject tree). These subject trees have to be identical on all nodes, and translated to all languages used. The replication mechanism was extended to deal with changes in controlled vocabularies as well. In this way, there is no central authority controlling the vocabularies, rather these can be enhanced and translated in a distributed way.

The StreamOnTheFly network collects usage events about the stored content. This is another important addition compared to traditional open archives. While a passive repository disseminates data only from the creator to the user, in our case creators get feedback on how their content was used.

For the implementation of this scenario we selected the collection of statistics, ratings and comments as examples. Statistics are collected on number of page impressions, listens and downloads of each radio show. Users may rate any show simply on a 1-5 scale. Users may give comments or participate in discussions about radio shows on portals. All the information is collected, processed and presented to creators. It is also presented to normal users, but in a limited way.

Calculation of rating information and statistics from usage data is most effective when usage data are concentrated on a single node. Gossip algorithms [18] are also known for calculating the mean (or other expression) of values distributed in a peer-to-peer network, but in this case the large number of aggregate values that needs to be computed in the whole network discourages the use of such algorithms.

It was decided to collect all usage information about a radio show on its home node (where the audio content is stored). The direct messaging protocol was introduced as an extension to the replication mechanism. While normal replication (called data sync) goes through the neighbour nodes, direct messages go straight to the addressed node. Elementary rating and usage data are wrapped into direct messages and sent to the home node for each object. A node receiving direct messages stores incoming elementary data as local objects and puts the subject radio show on its update queue. An internal update process periodically visits objects in the update queue and recalculates their aggregate statistics and rating data. As these recalculated data are stored in replicated objects, they travel on the usual way to all other peers of the network.

This solution ensures that usage data is collected quickly on the home node of the object. Update periodicity can be tuned to the computational capacity of the node. It can happen that overall statistics about a radio show on a remote node in the network is some days older, but the home node, where usually the creator of the show is a user, always contains the most up-to-date usage information.

StreamOnTheFly portals use the content of the whole network and they need access to the open APIs of nodes to obtain the content and metadata. Following earlier design decisions about security and controllable network traffic each portal is connected to a single node. This makes it easier for administrators to protect node servers or portal engines, and a software felt insecure is hardly adapted in this part of the content industry. A portal issues queries and downloads metadata from its so called parent node it is connected to. Complemented with caching, this should not present bigger load for a node than a couple of normal users. Portals are important for collecting consumer feedback. Collected data are sent periodically to the parent nodes, where these usage data are routed towards the home nodes of programs referred in the data as direct messages. The collected information states when

- a portal editor adds or removes a show on the portal,
- a show is visited/downloaded/listened by a user,
- a show is rated by a user,
- a show is commented by a user,
- the portal is visited or updated.

Streaming is provided by the parent node of the portal. Audio files are downloaded directly from the node. This is in harmony with previous design principles and ease the maintenance of the portal engine.

Nodes may choose to cache the audio content. A radio show which is selected to be listened is downloaded from its home node, stored in the cache and streamed to the user in parallel. Subsequent requests are served from the cache until the content gets outdated. Object replication signals the modification of the audio file, and this also triggers the deletion of that content from the cache. Caching is a fair solution for the community of StreamOnTheFly, as instead of being proportional to the number of popular radio shows the streaming load of a node will be proportional to the local users of node. Therefore, smaller nodes are not discouraged from publishing popular content.

4.4 Adding New Content to the Network

Station management software is the last piece in the StreamOnTheFly software suite. Its main purpose is to provide easy ways for uploading new content to nodes, as part of daily work of radio people. It also offers a comfortable way to insert metadata for the radio shows. In order to provide topic or genre for the show, it needs the up-to-date controlled vocabularies from the StreamOnTheFly network. These are periodically downloaded through open APIs of the node.

The station management software is also considered as a demonstrator for the use of station-node communication channels. As such, we wanted to apply very simple communication patterns. XBMF was designed to contain all information stored at a node. It was natural to implement content upload as upload of XBMF files. Nodes have a special directory for incoming new or modified content as XBMF. It depends on the node administrator how the XBMF files actually get into this directory. The sample software solution uses rsync for this purpose, a reliable and secure file transfer utility. Incoming XBMF files are periodically processed by the node software, the radio show is created/updated and audio files are automatically converted to recommended formats.

Some conversion of the metadata is also needed during this process. The station software assigns identifiers to objects within its scope and sends these identifiers along with metadata to the node. The node records these identifiers, but creates its own globally unique identifier for the object. The mapping between show identifiers at the station and identifiers at the node helps to detect if an older version of the radio show has to be modified.

Technically, the StreamOnTheFly network is growing into a heterogeneous component-based network where several methods coexist for the injection and reuse of content (Fig. 7).

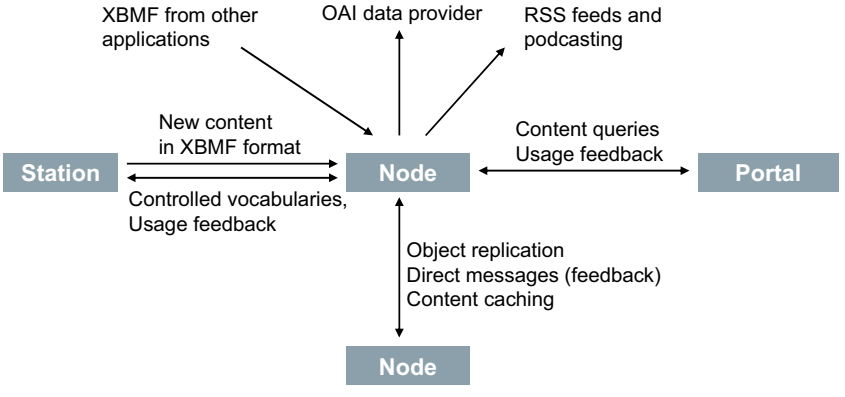


Fig. 7. Overview of communication within a StreamOnTheFly network

RSS feeds of the node not only disseminate the latest content from radios, series or people, but can also provide all important information for a single radio show including metadata, listening URLs and statistics. This latter option is a simple but efficient way of content export into other portal engines. Nodes can also act as OAI-PMH data providers, which is another way of opening up StreamOnTheFly archives.

4.5 Social Considerations

The architecture of the StreamOnTheFly network is based on the principles of self-organization. In system theory, the field of self-organization seeks general rules about the growth and evolution of systemic structure, the forms it might take and finally, methods that predict the future organization that will result from changes made to the underlying components [2]. Self-organization in community life (such as within the community of radio stations in Europe) means that the (social) rules governing the activities within the community evolve in time in order to adapt the community to the new challenges appearing within the sector. Self-organization in social systems can be identified as a continuous loop of emerging new structures and the reflection/response of these new structures in personal actions [11].

In this social meaning of the word, StreamOnTheFly has several properties that enhance the function of self-organization. The network consists of several modifiable and replacable components, and several gateways for communication with the rest of the world. Data moves in and out of the network very easily, providing practical ways for symbiosis with other information or content services.

Distributed data repositories, based on the principles of self-organization, should reflect the community (social) structure and adapt to its evolution. Self-organizing networks encourage users to contribute information. The StreamOnTheFly network helps editors to contribute radio shows. The network can thus be considered as a virtual platform for collaborative, community-based radio content production, distribution, and evaluation. Of equal importance, self-organizing networks aggregate and qualify the most relevant information in (semi-) automated fashion. The results are twofold: users can find quality information with confidence, while administrators and developers need less work to manually sift through large amounts of information. The StreamOnTheFly network provides services for content rating (and other statistical information about the actual use of audio content) and backward propagation of this information to the authors. Self-organizing networks create a community-like environment, and attempt to emphasize and visualize relationships between users thus increase user participation [5]. StreamOnTheFly network users are clustered according to their roles (e.g. station manager, editor, series editor, metadata classification scheme editor, general listener) and system is structured according to their feature requirements. The lack of central control within StreamOnTheFly network reflects the actual rather chaotic community structure of European community radios and the easy way of joining and leaving the network mimics the dynamics of movements in this sector.

The StreamOnTheFly system is a special example that is also based on the general principles of computational self-reflection [8] but the self-reflected system architecture (in which the self description of the actual system is an active part of the system itself) is substituted by a live human community and its social structure. An active (social) community (the community radios in Europe) is thus in the loop, the system is structured according to their (social) relationships. The community mimed system (service) structure emphasizes the openness, thus the system is rather an open infrastructure than a closed and static network, in which users can change and adapt the system itself to their own patterns of usage. Adaptation is possible on both syntactic and semantic level via the portal and the metadata related service components. Dynamics within the community can also be easily reflected. New StreamOnTheFly nodes can join or leave according to the actual changes in the community. A new classification scheme or evolution of the current classification scheme can immediately be embedded in an automatic way.

Summarizing the needs for a self-organizing collaborative media-sharing network, the following principles were used:

- no central role, server or governance, rather provision of an open, evolving infrastructure,
- easy to join and leave, but with the possibility to limit or exclude partners,
- free and unified access and content publishing,
- support for personalization on several levels (e.g. content, user interface, community),
- support for information (e.g. awareness, rating) propagation back to the authors.

5 Use Cases

The StreamOnTheFly network has been in operation since October 2003. The framework was built by using open source and free software components and the code base was published as open source on SourceForge. Since that time StreamOnTheFly has proved its flexibility and extensibility in several cases. On the other hand, the take-up of new technology by the radio community is rather slow. The core of the network (excluding experimental and stand-alone nodes) is accessible in 4 languages, and contains more than 500 hours of audio content from 20 radio stations[1].

The first real use case of the system is at the Technical Museum in Vienna. Medien.welten is the permanent exhibition on the development of media at the museum. To illustrate the convergence of radio and computers, StreamOnTheFly is used to record the complete Ö1 Programme, segment and categorize it automatically and publish it as a 30 day archive in the exhibition. Due to copyright issues this node is not available outside the museum.

The Volkswirtschaftliche Gesellschaft uses StreamOnTheFly technology to run a webradio station[2]. The project started late summer 2004, and will include up to 10 involved schools in the lower Austrian region.

For the third time, Christian Berger's team had broadcasted live from the Frankfurt Book Fair. In 2004, the whole webcast and archive was realized with StreamOnTheFly technology. The Frankfurt Book Fair is Europe's largest come-together of the european literature scene. Literadio is a joint project of German and Austrian community radios, and covers lectures, round tables, speeches and interviews in a daily webcast from 10am to 17pm live from the studio on Literadio's stand. The shows were stored at a node, and special integration with other portals was done using the RSS feed of the node. Another similar project based on StreamOnTheFly was the live radio broadcast at the 2002 IST conference in Copenhagen.

RSS 2.0 gave rise to a new phenomenon called podcasting. With a special software one can download not only the news like in traditional RSS, but also audio files for radio shows. Users select series they like and download them directly to their iPod or other MP3 listening device. After that they can listen their favourite programs anytime anywhere. With a small software update StreamOnTheFly is now capable of serving content for podcasters.

Another interesting approach to use StreamOnTheFly was developed to proof-of-concept stage at the Vorarlberg Universtity of Applied Sciences, where the node was used as a backend for a music publication site. In this case, the user interface was completely developed in Typo3, a php-based Content Management System. The interaction with the node was designed completely by using the XML-RPC API and the RSS distribution methods, thus validating the approach used in the design.

Radioswap.net uses this platform for content collection for Belgian community radio stations. At the moment their node cannot join the network because of legal and copyright issues. They are also working on a desktop tool facilitating easy upload of radio shows to StreamOnTheFly nodes.

[1] http://radio.sztaki.hu
[2] http://www.schulradio.at

Trials were implemented also to use StreamOnTheFly as a commercial audio service, an e-learning platform, and to add support for handling video content. The project officially finished in June 2004, but voluntary work keeps going on implementing various extensions.

6 Related Work

The StreamOnTheFly network builds upon the successful principles of the OAI-PMH protocol, although it applies its own special protocol for communication because of additional features. StreamOnTheFly replicates metadata in a peer-to-peer network in a push manner. As a result, the service provider functionality is uniformly available at all nodes in the network. Furthermore, the communication is extended with collection of feedback and statistics about the access and use of content.

There are only a few references to extend OAI-PMH into the peer-to-peer direction. OAI-P2P is a suggestion to organize a peer-to-peer network from nodes which are both OAI data providers and service providers [4]. This approach is based on Edutella, a network for storage, query and exchange of metadata [20]. In these networks queries are automatically routed to nodes which are able to provide answers, and the query is executed in the usual peer-to-peer way, parallelly on several nodes. In StreamOnTheFly, such solution was unacceptable because of two reasons: distributed searching may cause incomplete results and thus information loss (if some of the nodes are temporarily unavailable), and the response time of the query may be fluctuating. Distributed search is also a setback when experimenting with novel search methods and interfaces, as it was the case in our project as well.

Lagoze and Van de Sompel are considering peer-to-peer concepts in the OAI world [16]. They realize that OAI harvesting chains have increasing complexity, and thus they are thinking of peer-to-peer networks behaving as OAI data providers as an evolutional next step. Ding and Solvberg suggest a framework for harvesting metadata with heterogeneous schemes in a super-peer based P2P network [7].

Cooperation in archival and program exchange is an old demand of the radio community, and there are some services supporting this, for example the Open Meta Archive[3] and the Cultural Broadcasting Archive[4]. These services are all centralized and lacking the full support of the radio "supply chain" from production to listening.

Personalisation of radio or television program is getting more and more interest nowadays. An art project and experiment called Frequency Clock[5] provided a broadcasted TV channel where the program schedule could be modified during broadcast through a web server. Last.fm is a personalized online radio station where users create their profiles by selecting songs to listen, and then they get recommendations of new songs. The idea of flycasting [10] is to adapt the playlist according to the musical tastes of the people currently listening to an online radio station. StreamOnTheFly creates a unified platform for these ideas of personalisation, including rating, collaborative filtering, personalized radio program, personalized queries and portals.

[3] http://oma.sourceforge.net
[4] http://cba.fro.at
[5] http://www.frequencyclock.net

7 Conclusions

StreamOnTheFly is an operating example for a distributed network of heterogeneous digital library services. The core of the network combines peer-to-peer networking and open archive principles, while the rest of the services are realized as separate network components communicating through open APIs.

The network fulfills the initial goal to establish a common format and software support for the archival, exchange and reuse of radio programs. It also proceeds towards experimentation with latest networking paradigms and social self-organization. Our experience shows that the software components are easily adapted by various user groups and new emerging phenomena, such as RSS feeds or podcasting, are simple to integrate with the network.

Instead of automatically optimized network topologies, this network chooses the manual maintenance of peer-to-peer connections, which solves the problem of node authentication and fosters social self-organization of the network. We presented various tasks, problems and their solutions in such a network. The number of trials and applications supports the design and implementation of StreamOnTheFly. In the future, we plan to examine and further support emerging synergies catalysed by this service platform.

With the upcoming multimedia services via broadband and G3, audio archives get an additional boost. As seen in the US, streaming audio in general became an important aspect of daily business [21]. It is time to support this new technical potential with recent service infrastructure.

Acknowledgements. Authors would like to thank to the participants of the project for their support and help in the design and development of StreamOnTheFly software: Roland Alton-Scheidl, Thomas Hassan and Alexey Kulikov from Public Voice Lab, Jaromil, an independent audio and streaming expert, Thomas Thurner, Wolfgang Fuchs and Roland Jankowski from Team Teichenberg, Tamás Kézdi and Gergő Kiss from MTA SZTAKI DSD. This work was partially supported by the EU under contract IST-2001-32226.

References

1. Dublin core metadata initiative. `http://dublincore.org/`.
2. Self-organizing systems (sos) frequently asked questions. `http://www.calresco.org/sos/sosfaq.htm`.
3. Shared online media archive (soma) metadata element set version 1.0, September 2002. `http://soma-dev.sourceforge.net/`.
4. Benjamin Ahlborn, Wolfgang Nejdl, and Wolf Siberski. Oai-p2p: A peer-to-peer network for open archives. In *31st International Conference on Parallel Processing Workshops (ICPP 2002 Workshops)*, pages 462–468. IEEE Computer Society, 2002.
5. Hisham Alam. Self-organizing sites. `http://www.newarchitectmag.com/archives/2002/01/alam/`.
6. AAF Association. Advanced authoring format (aaf) object specification v1.1, 2005. `http://www.aafassociation.org/`.

7. Hao Ding and Ingeborg Solvberg. Metadata harvesting framework in p2p-based digital libraries. In *International Conference on Dublin Core and Metadata Applications (DC-2004), 11-14 Oct 2004, Shanghai, China*, Oct 2004.
8. Paul Dourish. Developing a reflective model of collaborative systems. *ACM Transactions on Computer-Human Interaction*, 2(1):40–63, 1995.
9. European Broadcasting Union (EBU). Core metadata set for radio archives (tech3293), 2001. http://www.ebu.ch/CMSimages/en/tec_doc_t3293_tcm6-10494.pdf.
10. James C. French and David B. Hauver. Flycasting: On the fly broadcasting. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.
11. Christian Fuchs. Science as a self-organizing meta-information system, 2004. http://ssrn.com/abstract=504244.
12. MPEG Group. Mpeg-7 (multimedia content description interface). http://www.chiariglione.org/mpeg/index.htm.
13. Rachel Heery, Pete Johnston, Csaba Fülöp, and András Micsik. Metadata schema registries in the partially semantic web: the cores experience. In *Dublin Core Conference, 2003, Seattle, Washington USA*, Oct 2003.
14. László Kovács, András Micsik, Balázs Pataki, and István Zsámboki. Aqua (advanced query user interface architecture). In José Luis Borbinha and Thomas Baker, editors, *Research and Advanced Technology for Digital Libraries, 4th European Conference, ECDL 2000*, volume 1923 of *Lecture Notes in Computer Science*, pages 372–375. Springer, 2000.
15. Carl Lagoze and Herbert Van de Sompel. The open archives initiative: building a low-barrier interoperability framework. In *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2001, Roanoke, Virginia, USA*, pages 54–62. ACM, 2001.
16. Carl Lagoze and Herbert Van de Sompel. The oai and oai-pmh: where to go from here?, February 2004. Presentation at the CERN Workshop Series on Innovations in Scholarly Communication: Implementing the benefits of OAI (OAI3). http://agenda.cern.ch/fullAgenda.php?ida=a035925.
17. András Micsik, Thomas Hassan, and László Kovács. Distributed archive and web services for radio stations. In *12th International World Wide Web Conference, 20-24 May 2003, Budapest, Hungary*, May 2003.
18. Alberto Montresor, Márk Jelasity, and Özalp Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *2004 International Conference on Dependable Systems and Networks (DSN 2004)*, pages 19–28. IEEE Computer Society, 2004.
19. Oliver Morgan. Metadata systems architecture. In *International Broadcasting Convention (IBC 2002)*, September 2002. http://www.broadcastpapers.com/ibc2002/ibc2002.htm.
20. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: a p2p networking infrastructure based on rdf. In *11th International World Wide Web Conference*, pages 604–615, 2002.
21. Arbitron/Edison Media Research. The value of internet broadcast advertising. *Internet and Multimedia*, 12, 2004.

Supporting Information Access in Next Generation Digital Library Architectures*

Predrag Knežević[1], Bhaskar Mehta[1], Claudia Niederée[1], Thomas Risse[1], Ulrich Thiel[1], and Ingo Frommholz[2]

[1] Fraunhofer IPSI, Integrated Publication and Information Systems Institute, Dolivostrasse 15, 64293 Darmstadt, Germany
{knezevic, mehta, niederee, risse, thiel}@ipsi.fhg.de
[2] University of Duisburg-Essen, Institute of Informatics and Interactive Systems, D-47048 Duisburg, Germany
ingo.frommholz@uni-due.de

Abstract. Current developments on Service-oriented Architectures, Peer-to-Peer and Grid computing promise more open and flexible architectures for digital libraries. They will open the Digital Library (DL) technology to a wider clientele, allow faster adaptability and enable the usage of federative models on content and service provision. These technologies raise new challenges for the realization of DL functionalities, which are rooted in the increased heterogeneity of content, services and metadata, in the higher degree of distribution and dynamics, as well as in the omission of a central control instance. This paper discusses these opportunities and challenges for three central types of DL functionality revolving around information access: metadata management, retrieval functionality, and personalization services.

1 Introduction

Currently, there is a considerable amount of R&D activity in developing viable strategies to use innovative technologies and paradigms like Peer-to-Peer Networking, Grid, and Service-oriented Architectures in digital libraries (see e.g. the European Integrated Projects BRICKS [8] and DILIGENT [12]). The promise is that these efforts will lead to more open and flexible digital library architectures that:

- open up digital library (DL) technology to a wider clientele by enabling more cost-effective and better tailored digital libraries,
- allow faster adaptability to developments in DL services and IT technologies, and
- enable usage of dynamic federative models of content and service provision involving a wide range of distributed content and service providers.

The use of Service-oriented Architectures, Grid infrastructures, and the Peer-to-Peer approach for content and service provision has implications for the realization of

* This work is partly funded by the European Commission under BRICKS (IST 507457), COLLATE (IST-1999-20882), DILIGENT (IST 004260) and VIKEF (IST 507173)

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 207–222, 2005.

enhanced DL functionality. These implications are mainly rooted in increased heterogeneity of content, services and metadata, in the higher degree of distribution and dynamics, as well as in the omission of a central control instance. On one hand, these are opportunities for better and more multifarious DL services; on the other hand, these are new challenges to ensuring long-term, reliable, and quality-ensured DL service provision that also exploits the technology promises. This paper discusses these opportunities and challenges for three central types of DL functionality revolving around information access: metadata management, retrieval functionality, and personalization services.

The rest of this paper is structured as follows: Section 2 presents the key ideas of next generation DL architectures based on exemplary RTD projects. Section 3 discusses how these new ideas influence information access in the areas of metadata management, information retrieval, and personalization support. Related work in these areas is considered in section 4. The paper concludes with a summary of the paper's key issues.

2 Next Generation Digital Library Architectures

Current plans for next generation DL architectures are aiming for a transition from the DL as an integrated, centrally controlled system to a dynamic configurable federation of DL services and information collections. This transition is inspired by new technology trends and developments. This includes technologies like Web services and the Grid as well as the success of new paradigms like Peer-to-Peer Networking and Service-oriented Architectures. The transition is also driven by the needs of the "DL market":

- better and adaptive tailoring of the content and service offer of a DL to the needs of the respective community as well as to the current service and content offer;
- more systematic exploitation of existing resources like information collections, metadata collections, services, and computational resources;
- opening up of DL technology to a wider clientele by enabling more cost-effective digital libraries.

To make these ideas more tangible we discuss three RTD projects in the field and discuss the relationship to upcoming e-Science activities.

2.1 Virtual Digital Libraries in a Grid-Based DL Infrastructure

DILIGENT[1] is an Integrated Project within the IST 6th Framework Programme. It's objective is "to create an advanced test-bed that will allow members of dynamic virtual e-Science organizations to access shared knowledge and to collaborate in a secure, coordinated, dynamic and cost-effective way."

The DILIGENT testbed will enable the dynamic creation and management of Virtual Digital Libraries (VDLs) on top of a shared Grid-enabled DL infrastructure, the DILIGENT infrastructure. VDLs are DLs tailored to the support of specific e-Science communities and work groups. For creating a VDL, DL services, content collections, metadata collections are considered as Grid resources and are selected, configured, and

[1] DILIGENT - A DIgital Library Infrastructure on Grid ENabled Technology.

integrated into processes using the services of the DILIGENT infrastructure. This infrastructure builds upon an advanced underlying Grid infrastructure as it is currently evolving e.g. in the EGEE project[2].

Such a Grid infrastructure will already provide parts of the functionality required for DILIGENT. This includes the dynamic allocation of resources, support for cross-organizational resource sharing, and a basic security infrastructure. For effectively supporting DLs, additional services like support for redundant storage and automatic data distribution, metadata broker, metadata and content management, advanced resource brokers, approaches for ensuring content security in distributed environments and the management of content and community workflows are required in addition to services that support the creation and management of VDLs. A further project challenge are systematic method to make the treasure of existing DL services and collections utilizable as Grid resources in the DILIGENT infrastructure.

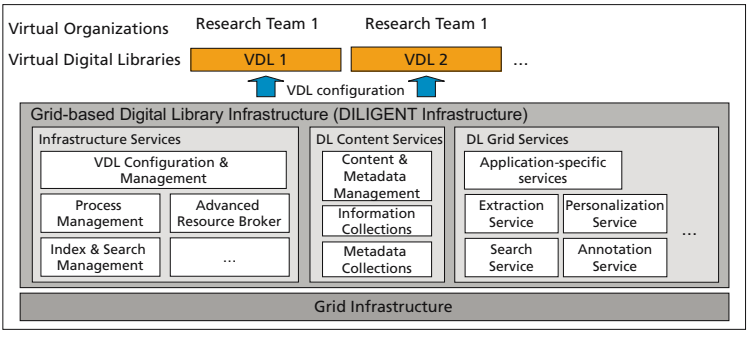


Fig. 1. DILIGENT Virtual Digital Library Infrastructure

Figure 1 shows an architecture overview of the DILIGENT infrastructure. Based on the Grid infrastructure it contains three types of services. The *Infrastructure Services* are a group of services that are specific for operating the infrastructure. This group contains the services for the configuration and management of VDLs. The *DL Grid Services* are a set of specific library services. On one hand, existing DL services are wrapped as Grid Services or adapted to the new Grid-based environments. On the other hand, new services are implemented that specifically take into account the operation environment, e.g. services for distributed search. Besides generic services like annotation and search services, this group also contains application specific services. Finally, the *DL Content Services* include services for content and metadata management as well as existing information and metadata collections wrapped as Grid services. This DILIGENT infrastructure is used to create VDLs in support of virtual organizations.

The DILIGENT project will result in a Grid-enabled DL testbed that will be validated by two complementary real-life application scenarios: one from the Cultural Heritage domain and one from the environmental e-Science domain.

[2] http://public.eu-egee.org

2.2 Service-Oriented and Decentralized DL Infrastructure

The aim of the BRICKS[3] project [8] is to design, develop and maintain an open user and service-oriented infrastructure to share knowledge and resources in the Cultural Heritage domain. The target audience is very broad and heterogeneous and involves cultural heritage and educational institutions, research community, industry, and citizens. Typical usage scenarios are integrated queries among several knowledge resource, e.g. to discover all Italian artefacts from renaissance in the European museums. Another example is to follow the life cycle of historic documents, whose manual copies are distributed all over Europe. These examples are specific application, which are running on top of the BRICKS infrastructure.

The BRICKS infrastructure uses the Internet as a backbone and has to fulfill the following requirements:

- Expandability, which means the ability to acquire new services, new content, or new users, without any interruption of service.
- Scalability, which means the ability to maintain excellence in service quality, as the volumes of requests, of content and of users increase.
- Availability, which means the ability to operate in a reliable way over the longest possible time interval.
- Graduality of Engagement, which means the ability to offer a wide spectrum of solutions to the content and service providers that want to become members of BRICKS.
- Interoperability, which means the ability to make available services to and exploit services from other digital libraries.

In addition, the user community has the economic requirement to be low-cost. This means (1) that an institution should be able to become a BRICKS member with minimal investments, and (2) that the maintenance costs of the infrastructure, and in consequence the running costs of each BRICKS member, are minimized.

Interested institution should not invest much additional money in its already existing infrastructure to become a member of BRICKS. In the ideal case the institution should only get the BRICKS software distribution, which will be available for free, install it, connect to the Internet and become a BRICKS member. This will already gives the possibility to search for content and access some services. For sure, additional work is necessary to integrate and adapt existing content and services to provide them in BRICKS.

Also, the BRICKS membership will be flexible, such that parties can join or leave the system at any point in time without administrative overheads. To minimize the maintenance cost of the infrastructure any central organization, which maintains e.g. the service directory, should be avoided.

With respect to access functionality, BRICKS provides appropriate task-based functionality for indexing/annotation and collaborative activities e.g. for preparing a joint multimedia publication. An automatic annotation service will enable users to request background information, even if items have not been annotated by other users yet. By

[3] BRICKS - Building Resources for Integrated Cultural Knowledge Services.

selecting appropriate items, such as definitions of concepts, survey articles or maps of relevant geographical areas, the service exploits the currently focused items and the user's goals expressed in the user profile. In addition, the linking information, which is generated dynamically, must be integrated into the documents. The design of the access functionality is influenced by our experiences in the 5th Framework project COLLATE.

2.3 COLLATE: A Web-Based Environment for Document-Centered Collaboration

Designed as a content- and context-based knowledge working environment for distributed user groups, the COLLATE system supports both individual work and collaboration of domain experts with material in the data repository. The example application focuses on historic film documentation, but the developed tools are designed to be generic and as such adaptable to other content domains and application types. This is achieved by model-based modules.

The system supports collaborative activities such as creating a joint publication or assembling and creating material for a (virtual) exhibition, contributing unpublished parts of work in the form of extended annotations and commentaries. Automatic indexing of textual and pictorial parts of a document can be invoked. Automatic layout analysis for scanned documents can be used to link an annotation of individual segments. As a multi-functional means of in-depth analysis, annotations can be made individually but also collaboratively, for example in the form of annotation of annotations, collaborative evaluation, and comparison of documents. Through interrelated annotations users can enter into a discourse on the interpretation of documents and document passages.

The COLLATE collaboratory is a multi-functional software package integrating a large variety of functionalities that are provided by cooperating software modules residing on different servers. It can be regarded as a prototypical implementation of a decentralized, Service-oriented DL architecture which serves as a testbed for the collaborative use of documents and collections in the Humanities. The collaborative creation of annotation contexts for documents offers new opportunities for improving the access functionality, as we will illustrate later on.

2.4 Next Generation DL Architectures and e-Science

Scientific practice is increasingly reliant on data-intensive research and international collaboration enabled by computer networks. The technology deployed in such scenarios allows for high bandwidth communication networks, and by linking computers in "Grids" places considerably more powerful computing resources is at their disposal than a single institution could afford. If we view e-Science as being primarily motivated up to now by notions of resource sharing for computationally intensive processes (e.g. simulations, visualization, data mining) a need is emerging for new approaches, brought up by ever more complex procedures, which, on the one hand, assume the reuse of workflows, data and information and, on the other hand, should be able to support collaboration in virtual teams. Future concepts of e-Science will be less focused on data and computing resources, but will include services on the knowledge and organizational

levels as well. Embedding future DL architectures in an emerging e-Science infrastructure will meet these requirements by providing access to information and knowledge sources, and appropriate collaboration support on top of the Grid-based infrastructure.

3 Information Access in Next Generation DL Architectures

A decentralized, service-oriented architecture poses new challenges to the technologies employed for information access. DLs based on such an architecture should, for example, not only provide access and retrieval functionality for the documents residing on the local peer, but should also consider other peers which might host relevant document w.r.t. a query. In the following, we will outline possible approaches for enhanced services for information access. Such services will utilize the functions of a decentralized metadata management ensuring the availability of all documents (and their parts) while reducing overhead costs. Retrieval functions can be improved by taking into account the annotational contexts of documents emerging for the collaborative process of interpreting and discussing items of interests by a group of users. In addition, individual users‘ contexts can be used to personalize the access services.

3.1 Decentralized Metadata Management

DLs usually like to keep content under control in their local repositories. On the contrary, metadata should be available for all parties, stored in some central place accessible for everybody. Decentralized architectures by definitions avoid having central points, for the following reasons: they are candidate single point of failure and performance bottleneck. Therefore, metadata must be spread in the community. A naïve approach for metadata searching would be to distribute queries to all members, but it is obvious that the solution is unscalable. Hence, efficient metadata access and querying are very important challenges within the new decentralized settings.

Our proposal to these challenges is a decentralized Peer-to-Peer datastore that will be used for managing XML-encoded metadata. It balances resource usage within the community, has high data availability (i.e. data are accessible even if creator disappears from the system, e.g. system fault, network partitioning, or going offline), is updateable (i.e. stored data can be modified during the system lifetime), and supports a powerful query language (e.g XPath/XQuery).

XML documents are split into finer pieces that are spread within the community. The documents are created and modified by the community members, and can be accessed from any peer in a uniform way, e.g. a peer does not have to know anything about the data allocation. Uniform access and balanced storage usage are achieved by using a DHT (Distributed Hash Table) Overlay [26] and having unique IDs for different document parts.

Figure 2 shows the proposed database architecture. All layers exist on every peer in the system. The datastore is accessed through the P2P-DOM component or by using the query engine. The query engine could be supported by an optional index manager that maintains indices.

P2P-DOM is the core system layer. It exports a large portion of the DOM interface to the upper layers, and maintains a part of a XML tree in a local store (e.g. files,

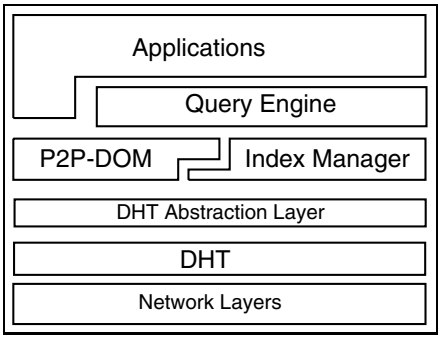


Fig. 2. Decentralized XML Storage Architecture

database). P2P-DOM serves local requests (adding, updating and removing of DOM-tree nodes) and requests coming from other peers through a Distributed Hash Table (DHT) [35,1,37] overlay, and tries to keep the decentralized database in a consistent state. In order to make the DHT layer pluggable, it is wrapped in a tiny layer that unifies APIs of particular implementations, so the upper layer does not need to be modified. A more detailed discussion about the proposed approach, challenges and open issues can be found in [21].

In the rest of the subsection, we are giving more details how the proposed datastore could be used for managing service metadata, which are an additional type of DL metadata introduced by Service-oriented Architectures.

Service metadata describe service functionalities, interfaces and other properties. These meta-information are usually encoded by using WSDL (Web Service Description Language [39]) and published to an UDDI (Universal Description, Discovery and Integration [30]) service directory. Service discovery queries are usually more complex than simple name matching, i.e. they contain qualified, range and/or boolean predicates.

In order to realize a decentralized service directory with advanced query mechanisms, the community of service providers will create and maintain in the decentralized P2P data store a pool of the service descriptions. Every service will be able to modify its description during the lifetime and to search for needed services. Query execution will be spread at many peers, the query originator will only get the final result back.

At the same time, due to uniform data access, new community members can start using the service directory immediately after joining the system without additional setup and administration. A member decision to leave the community will not make any influence for the rest of the system, because data are replicated. Even if network partitioning happens, the service directory would provide access to service metadata available in the partition allowing some parties to continue with work without interruption.

For details about the use of the decentralized datastore in other scenarios see [34].

3.2 Decentralized Context-Based Information Retrieval

DLs based on a decentralised architecture should not only provide access and retrieval functionality for the documents residing on the local peer, but should also consider other

peers which might host relevant document w.r.t. a query. It is clear that for a scenario like described above appropriate search functionality has to be defined. In the following, we will outline possible approaches for enhanced retrieval services.

Services. In order to be able to abstract from the underlying infrastructure, retrieval functionality should be implemented as a service with a predefined API and behavior. This has the advantage that other peers are able to query the local repository, which is an important feature for enabling P2PIR. An example Web Service specification for search and retrieval is SRW[4]. It considers content-based retrieval functionality, but lacks context-based features as proposed above. When performing retrieval based on the annotation context (see below), such context information should be contained in the result set in order to elucidate why an item was retrieved. So a common API for queries, results and indexing requests has to be identified which is capable of taking advanced queries and context information into account.

Annotation Context. According to the considerations in [3], annotations can be discussed from different viewpoints. From a syntactic point of view, annotations are metadata, since they are connected to the objects they annotate. From their semantics, annotations can either contain content about content (like we find it, for instance, in reviews, recensions, and judgements about documents) or additional content manifesting itself in, e.g., elaborations or augmentations of the content at hand, but also in links that connect documents and annotations with other objects. Interpretations, like we find them in the cultural domain, are an example of annotations conveying both content about content and additional content. Regardless of their semantics, annotations are dialogue acts following certain pragmatics. These pragmatics describe the intention behind a user's statement. This means that annotations consist of certain communicative acts [36], which can be, e.g., assertives, directives, and commissives[5]. In both digital libraries and collaboratories, annotations can play a beneficial role w.r.t. certain aspects of such systems. They support authoring and editing, access and retrieval, effective use, interaction, and sharing of data and resources.

Annotations can be either manually or automatically created. Manual annotations range from personal to shared to public ones. They can include personal notes, e.g., for comprehension, and whole discussions about documents [13,2]. Annotations are building blocks for collaboration. In a distributed, decentralized environment, especially shared and public annotations pose a challenge to the underlying services. Users can create shared and public annotations residing on their peers, but this data has to be spread to other peers as well.

By automatic annotations, we mean the automatic creation and maintenance of annotations consisting of links to and summaries of documents on other peers which are similar to documents residing on the local peer. Such annotations constitute a context in which documents on a peer are embedded. Dynamic links raise the degree to which the users' work is efficiently supported by the digital library [38]. They provide the opportunity to create comprehensive answers to submitted queries, an idea which is also

[4] `http://www.loc.gov/z3950/agency/zing/srw/`

[5] Even creating a link between objects is a communicative act, since one makes an assertion about the relationship of these objects, or at least that there is such a relationship at all.

stated in [17]. For each document, agents could be triggered to periodically update the information at hand, similar to the internal linking methods like similarity search, enrichment and query generation proposed in [38]. P2PIR methods can possibly be applied for this. The underlying assumption is that a user stores potential interesting documents on her peer and is interested in similar publications. Automatic annotations can be created w.r.t. several aspects. For instance, topical similar documents can be sought after. Another interesting kind of automatic annotation can be extracted from the surroundings of a citation. If a document residing on another peer cites a document on the local peer, the surroundings of this citation usually contain some comments about the cited document (similar as reported in [5]). Since only annotations to documents residing on the peer are created, storage costs can be kept low. Regular updates performed by agents keep the user informed.

Annotations, either manual or automatic ones, constitute a certain kind of *document context*. Annotation-based retrieval methods [13] can employ the annotation context without the need to actually access other peers. Since annotations, being manually or automatically created, contain additional information about the document, we assert that annotation-based retrieval functions boost retrieval effectiveness. Future work will show if this assumption holds. Using annotations for information retrieval in a decentralized environment has the advantage that annotations are locally available, but reflect information lying on other peers. In this way, annotations create new access structures which help addressing problems arising when performing information retrieval on an underlying P2P infrastructure.

3.3 Cross-Service Personalization

Personalization approaches in DLs dynamically adapt the community-oriented service and content offerings of a DL to the preferences and requirements of individuals [28]. They enable more targeted information access by collecting information about users and by using these user models (also called user profiles) in information mediation.

Personalization typically comes as an integral part of a larger system. User profiles are collected based on a good knowledge about the meaning of user behavior and personalization activities are tailored to the functionality of the respective system. Within a next-generation distributed DL environment, which is rather a dynamic federation of library services than a uniform system, there are at least two ways to introduce personalization. In the simple case, each service component separately takes care of its personalization independently collecting information about users. A more fruitful approach, however, is to achieve personalization across the boundaries of individual services, i.e., cross-system or, more precisely, cross-service personalization. In this case, personalization relies on a more comprehensive picture of the user collected from his interaction with different library services.

Cross-service Personalization Challenges. Cross-service personalization raises the following challenges:

- How to bring together the information collected about a user and his interactions with the different services in a consistent way?

- How to make make up-to-date, aggregated user information about the user available to the different services, i.e. how to manage, update, and disseminate user models to make them accessible to the different services?
- How to support (at least partial) interpretation of the user model in a heterogeneous, and dynamically changing DL service environment?

This requires a shared underlying understanding of the user model. Furthermore, it raises issues of privacy and security, since personal data is moved around in a distributed system. It has to be taken into account that the privacy concerns of the user might be increased by the fact that the information collected from the interaction with different services is combined. This adds an additional challenge for cross-system and cross-service personalization, namely to give the user some control over the information that is collected about him.

Approaches to Cross-Service Personalization. We identified two principle approaches which differ from each other in their architecture:

Adaptor approach: The adaptor approach relies on the ideas of wrapper architectures. A kind of wrapper is used to translate information access operations into personalized operations based on the information collected in the context passport.
The advantage of this approach is that personalization can also be applied to services that themselves do not support personalization. The disadvantage is that every service will need its own wrapper. Unless there is a high degree of standardization in service interfaces, creating wrappers for every individual services may not be practical and does not scale well in dynamic service environments.

Connector approach: In contrast to the adaptor approach, the connector approach relies on the personalization capabilities of the individual services. It enables the bi-directional exchange of data collected about the user between the context passport and the personalization component of the respective service. The context passport is synchronized with individual user models/profiles maintained by services. The advantage here is that personalization of one service can benefit from the personalization efforts of another.

A flexible and extensible user model that can capture various characteristics of the user and his/her context is in the core of both approaches. An example of such a model, that is rather a metamodel for describing different user models is the UUCM model described in [29].

As an operationalization of such a model we developed the idea of a *context passport*. A context passport accompanies the user and is ”presented” to services to enable personalized support. The context passport [29] is positioned as a temporal memory for information about the user. It covers an extensible set of facets modeling different user model dimensions, including cognitive pattern, task, relationship, and environment dimension. The selection of the dimensions is based on user models in existing personalization approaches and on context modeling approaches. The context passport acts as an aggregated service-independent user profile with services receiving personalization data from the context passport. Services also transfer information to the context passport based on relevant user interaction which add up-to-date information to the user's context. Here it is important that the information is exchanged on an aggregation level that

is meaningful outside the individual service. The context passport is maintained by an active user agent which communicates with the services via a specific protocol.

A flexible protocol is required for this communication between context passport and the service-specific personalization component. Such a protocol has to support the negotiation of the user model information to be exchanged and the bidirectional exchange of user information. In more detail such a protocol operates in three phases a) negotiation phase, b) personalization phase, and c) synchronization phase. In the negotiation phase, the Context Passport and the service agree on information to be exchanged. The main goal to be achieved is a common understanding on the type of information that the other partner can understand and use. In our approach the UUCM provides the common vocabulary for negotiating about the available user information (dimensions, facets about the user, etc.) In order to perform an automatic negotiation about what activities can be supported, there needs to be an agreement on a machine understandable common vocabulary or ontology of the respective domain (e.g. Travel). After an agreement has been reached on the activity to be performed and the available user information, the Context Passport needs to extract information relevant to this activity (context selection). This is communicated to the system in the personalization phase. After the personalized activity has been performed, the respective service has a slightly changed understanding of the user. In the synchronization phase the service informs the context passport about these changes keeping the Context Passport uptodate.

There is thus a requirement from bidirectional information exchange so that other services may benefit from up-to-date information about the user. An early implementation of the Context Passport has been done in the WWW scenario, which supports web systems for Cross-system Personalization. This implementation supports an early version of the CSCP enabling synchronization of user profiles between two test web systems. Implementation details are available in [24]. Future implementations will support task based reasoning and relationship based recommendations.

4 Related Work

Metadata Management. Decentralized and peer-to-peer systems can be considered as a further generalization of distributed systems. Therefore, decentralized data management has much in common with distributed databases, which are already well explored [32,6]. However, some important differences exist. Distributed databases are made to work in stable, well connected environments (e.g. LANs) with the global system overview, where every crashed node is eventually replaced by a new proper one. Also, they need some sort of administration and maintenance.

On the contrary, the P2P systems are deployed mostly on the highly unreliable Internet. Some links can be down, network bandwidths are not guaranteed. The P2P systems allow disconnection of any peer at any time, without a need for replacement, and none of the peers is aware of the complete system architecture. Therefore, the system must self-organize in order to survive such situations.

Many distributed databases like Teradata, Tandems NonStopSQL, Informix Online Xps, Oracle Parallel Server and IBM DB2 Parallel Edition [9] are available on the mar-

ket. The first successful distributed filesystem was Network File System (NFS) succeeded by Andrew File System (AFS), Coda and xFS, etc.

Current popular P2P file-sharing systems (e.g. KaZaA, Gnutella, eDonkey, Past [26]) might be a good starting point for enabling decentralized data management. However, these systems have some important drawbacks: file-level granularity and write-once access, i.e. files are non-updateable after storing. Storing a new version requires a new filename. Usually, a file contains many objects. As a consequence, retrieving a specific object would require getting the whole file first. If a object must be updated, then a whole new file version must be created and stored. In current systems it is not possible to search for a particular object inside the files. The query results contain the whole files, not only requested objects. Advanced searching mechanism like qualified, range or boolean predicates search is not supported. Usually, metadata have rich and complex structure and queries on them are more than simple keyword match. Also, metadata should be updateable. Thus, the presented P2P systems are not suitable for decentralized metadata management.

There are some attempts [18] to extend Gnutella protocols to support other types of queries. It would be quite possible to create a Gnutella implementation that understands some variant of SQL, XPath or XQuery. However, such networks would have problems with system load, scalability and data consistency, e.g. only locally stored data could be updated and mechanisms for updating other replicas do not exist.

Information Retrieval. Typical Peer-to-peer information retrieval (P2PIR) methods are working decentralized, as proposed by the P2P paradigm [26]. No server is involved as it would be in a hybrid or client-server architecture. Common P2PIR approaches let the requesting peer contact other peers in the network for the desired documents. In the worst case, the query is broadcast to the whole network resulting in lots of communication overhead. Another approach would be to store all index information on every peer and search for relevant documents locally. Peers would request the required information during the initial introduction phase, and updates would be spread from time to time. However, this approach is not feasible since the expected storage costs would be quite high. Intermediate approaches which try to balance communication and storage costs work with peer content representations like the clustering approach discussed in [27]. Such a peer content representation does not need the amount of data a snapshot of the whole distributed index would need, but conveys enough information to estimate the probability that a documents relevant to the query can be found on a certain peer.

Some annotation systems [31] provide simple full-text search mechanisms on annotations. The Yawas system [11] offers some means to use annotations for document search, e.g. by enabling users to search for a specific document type considering annotations. Golovchinsky *et al.* [16] use annotations as markings given by users who judge certain parts of a document as being important when emphasizing them. Their approach gained better results than classic relevance feedback, as experiments showed. Agosti *et al.* [3] discuss facets of annotations and propose an annotation-based retrieval function based on probabilistic inference. Frommholz *et al.* [14] present a nested annotation retrieval approach based on probabilistic logics. This approach does not only consider syntax and semantic of annotations, but makes use of (explicitly given) discourse structure relations among them. The idea of automatic annotations is motivated

by the internal linking methods described in [38] by Thiel *et al*. Related to this is the overview given by Agosti and Melucci in [4], where they discuss how to use information retrieval techniques to automatically create hypertexts.

Personalization Support. The most popular personalization approaches in digital libraries or more general in information and content management systems are recommender systems and methods that can be summarized under the term personalized information access. Recommender systems (see e.g. [7]) give individual recommendations for information objects following an information push approach, whereas personalized information access (personalized newspapers, etc.) is realized as part of the information pull process, e.g. by filtering retrieval results or refining the queries themselves.

Personalization methods are based on modeling user characteristics, mainly cognitive pattern like user interests, skills and preferences [23]. More advanced user models also take into account user tasks [20] based on the assumption that the goals of users influence their needs. Such extended models are also referred to as user context models [15]. A flexible user context model that is able to capture an extensible set of user model facets as it is required for cross-service personalization can be found in [29]. Information for the user models (also called user profiles) are collected explicitly or implicitly [33], typically by tracking user behavior. These user profiles are used for personalized filtering in information dissemination (push) as well as in information access (pull) services. An important application area is personalized information retrieval. The information about the user is used for query rewriting [19], for the filtering of query results [10] as well as for a personalized ranking of query results [25].

5 Conclusions and Future Work

In this paper, we discussed opportunities and challenges for information access support resulting from the transition of more traditional, centrally controlled DL architectures to DLs as dynamic federations of content collections and DL services. The discussion focused on metadata management, information retrieval, and personalization support. In addition to discussing the central challenges, an advanced approach has been discussed for each of the three aspects: For metadata management a decentralized P2P data store solves the problem of systematic and efficient decentralized metadata management. Applications of annotations and annotation-based retrieval in the P2P context is considered as a way to improved information retrieval support in a decentralized environment. Finally, cross-service personalization is discussed as an adequate way to handle personalization in a dynamic service-oriented environment.

The list of the considered information access issues discussed is not meant to be exhaustive. Further challenges raise within next-generation DL architectures like effective metadata brokering and advanced methods for ensuring content security and quality. The envisaged support for information access needs to combine the approaches mentioned above in a balanced way to ensure that users will benefit from decentralized architectures, while at the same time, maintaining the high level of organization and reachability that users of DL systems are used to. Such issues are addressed in the BRICKS and the DIIGENT project in which our institute is involved together with partners from other European countries.

References

1. Karl Aberer. P-Grid: A self-organizing access structure for P2Pinformation systems. *Lecture Notes in Computer Science*, 2172, 2001.
2. M. Agosti and N. Ferro. Annotations: Enriching a Digital Library. *In: Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*. Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany,pages 88–100, 2003.
3. Maristella Agosti, Nicola Ferro, Ingo Frommholz, and Ulrich Thiel. Annotations in digital libraries and collaboratories – facets, models and usage. In: *Proc. 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 244–255, 2004.
4. Maristella Agosti and Massimo Melucci. Information retrieval techniques for the automatic construction of hypertext. In: A. Kent and C. M. Hall, editors, *Encyclopedia of Library and Information Science*, volume 66, pages 129–172. Marcel Dekker, New York, USA, 2000.
5. Giuseppe Attardi, Antonio Gullí, and Fabrizio Sebastiani. Automatic Web page categorization by link and context analysis. In: Chris Hutchison and Gaetano Lanzarone, editors, *Proceedings of THAI-99, 1st European Symposium on Telematics, Hypermedia and Artificial Intelligence*, pages 105–119, Varese, IT, 1999.
6. Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurency Control and Recovery in Database Systems*. Addison-Wesley, 1997.
7. Vincent Bouthors and Olivier Dedieu. Pharos, a collaborative infrastructure for web knowledge sharing. In: Serge Abiteboul and Anne-Marie Vercoustre, editors, *Research and Advanced Technology for Digital Libraries, Proceedings of the Third European Conference, ECDL'99, Paris, France, September 1999*, volume LNCS 1696 of *Lecture Notes in Computer Science*, page 215 ff. Springer-Verlag, 1999.
8. BRICKS Consortium. *BRICKS - Building Resources for Integrated Cultural Knowledge Services (IST 507457)*, 2004. `http://www.brickscommunity.org/`.
9. L. Brunie and H. Kosch. A communications-oriented methodology for load balancing in parallel relational query processing. In: *Advances in Parallel Computing, ParCo Conferences, Gent, Belgium*, 1995.
10. Edgar Casasola. Profusion personal assistant: An agent for personalized information filtering on the www. Master's thesis, The University of Kansas, Lawrence, KS, 1998.
11. L. Denoue and L. Vignollet. An annotation tool for web browsers and its applications to information retrieval. In: *Proceedings of RIAO 2000, Paris, April 2000*, April 2000.
12. DILIGENT Consortium. *DILIGENT - A DIgital Library Infrastructure on Grid ENabled Technology (IST 004260)*, 2004. `http://www.diligentproject.org/`.
13. I. Frommholz, H. Brocks, U. Thiel, E. Neuhold, L. Iannone, G. Semeraro, M. Berardi, and M. Ceci. Document-centered collaboration for scholars in the humanities - the COLLATE system. In: *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*. Lecture Notes in Computer Science 2769, Springer, Heidelberg, Germany, pages 434–445, 2003.
14. Ingo Frommholz, Ulrich Thiel, and Thomas Kamps. Annotation-based document retrieval with four-valued probabilistic datalog. In: Thomas Roelleke and Arjen P. de Vries, editors, *Proceedings of the first SIGIR Workshop on the Integration of Information Retrieval and Databases (WIRD'04)*, pages 31–38, Sheffield, UK, 2004.
15. A. Goker and H.I. Myrhaug. User context and personalization. In: *Proceedings of the European Conference on Case Based Reasoning (ECCBR 2002) - Workshop on Personalized Case-Based Reasoning, Aberdeen, Scotland, 4-7 September 2002*, volume LNCS 2416 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2002.

16. G. Golovchinsky, M. N. Price, and B. N. Schilit. From reading to retrieval: Freeform ink annotations as queries. In: F. Gey, M. Hearst, and R. Tong, editors, *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–25, New York, 1999. ACM Press.
17. Gene Golovchinsky. What the query told the link: The integration of hypertext and information retrieval. In: Mark Bernstein, Kasper Osterbye, and Leslie Carr, editors, *Proceedings of the 8th ACM Conference on Hypertext (Hypertext '97)*, pages 67–74, Southampton, UK, 1997. ACM Press, New York, USA.
18. GPU. A gnutella processing unit, 2004. `http://gpu.sf.net`.
19. Jon Atle Gulla, Bram van der Vos, and Ulrich Thiel. An abductive, linguistic approach to model retrieval. *Data & Knowledge Engineering*, 23(1):17–31, June 1997.
20. C. Kaplan, J. Fenwick, and J. Chen. Adaptive hypertext navigation based on user goals and context. In: *User Modeling and User-Adapted Interaction 3*, pages 193–220. Kluwer Academic Publishers, The Netherlands, 1993.
21. Predrag Knežević. Towards a reliable peer-to-peer xml database. In: Wolfgang Lindner and Andrea Perego, editors, *Proceedings ICDE/EDBT Joint PhD Workshop 2004*, pages 41–50, P.O. Box 1527, 71110 Heraklion, Crete, Greece, March 2004. Crete University Press.
22. T. Koch and I. T. Sølvberg, editors. *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*. Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany, 2003.
23. M.F. McTear. User modeling for adaptive computer systems: A survey of recent developments. In: *Artificial Intelligence Review*, volume 7, pages 157–184, 1993.
24. B. Mehta, C. Niederée, and A. Stewart. Towards cross-system personalization. In: *To appear in Proceedings of UAHCI 2005, Las Vegas, Nevada, July 2005*, 2005.
25. Xiannong Meng and Zhixiang Chen. Personalize web search using information on client's side. In: *Proceedings of the Fifth International Conference of Young Computer Scientists, August 17-20, 1999, Nanjing, P.R.China*, pages 985–992. International Academic Publishers, 1999.
26. Dejan Milojičić, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical report, 2002. `http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf`.
27. Wolfgang Müller and Andreas Henrich. Fast retrieval of high-dimensional feature vectors in P2P networks using compact peer data summaries. In: *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 79–86. ACM Press, 2003.
28. E. J. Neuhold, C. Niederée, and A. Stewart. Personalization in digital libraries: An extended view. In: *Proceedings of ICADL 2003*, pages 1–16, 2003.
29. C. Niederée, A. Stewart, B. Mehta, and M. Hemmje. A multi-dimensional, unified user model for cross-system personalization. In: *Proceedings of Advanced Visual Interfaces International Working Conference (AVI 2004) - Workshop on Environments for Personalized Information Access, Gallipoli (Lecce), Italy, May 2004*, 2004.
30. OASIS. *Universal Description, Discovery and Integration (UDDI)*, 2001. `http://www.uddi.org/`.
31. Ilia A. Ovsiannikov, Michael A. Arbib, and Thomas H. McNeill. Annotation technology. *Int. J. Hum.-Comput. Stud.*, 50(4):329–362, 1999.
32. M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
33. Alexander Pretschner and Susan Gauch. Personalization on the web. Technical Report ITTC-FY2000-TR-13591-01, Information and Telecommunication Technology Center (ITTC), The University of Kansas, Lawrence, KS, December 1999.

34. Thomas Risse and Predrag Knežević. Data storage requirements for the service oriented computing. In: *SAINT 2003 - Workshop on Service Oriented Computing*, pages 67–72, January 2003.
35. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.
36. J.R. Searle. A taxonomy of illocutionary acts. In J.R. Searle, editor, *Expression and Meaning. Studies in the Theory of Speech Acts*, pages 1–29. Cambridge University Press,, Cambridge, 1979.
37. Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
38. U. Thiel, A. Everts, B. Lutes, M. Nicolaides, and K. Tzeras. Convergent software technologies: The challenge of digital libraries. In *Proceedings of the 1st Conference on Digital Libraries: The Present and Future in Digital Libraries*, pages 13–30, Seoul, Korea, November 1998.
39. W3C. *Web Services Description Language (WSDL) 1.1*, March 2001. `http://www.w3.org/TR/wsdl`.

Query Trading in Digital Libraries

Fragkiskos Pentaris and Yannis Ioannidis

Department of Informatics and Telecommunications,
University of Athens, Panepistimiopolis, 15771, Athens, Greece
{frank, yannis}@di.uoa.gr

Abstract. In this paper we present a distributed query framework suitable for use in federations of digital libraries (DL). Inspired by e-commerce technology, we recognize CPU-processing and queries (and query answers) as commodities and model the task of query optimization and execution as a task of trading CPU-processing, queries and query-answers. We show that our framework satisfies the needs of modern DL federations by respecting the autonomy of DL nodes and natively supporting their business model. Our query processing conception is independent of the possible distributed architecture and can be easily implemented over a typical GRID architectural infrastructure or a Peer-To-Peer network.

1 Introduction

Digital Libraries' users may need to simultaneously use two or more libraries to find the information they are looking for. This increases the burden of their work as it forces them to pose the same query to different DLs multiple times, each time using a possibly different user interface and a different metadata schema. Digital Libraries are aware of this deficiency and have been trying for a long time to attack this problem by creating federations of DLs. Especially for small DLs, joining such federations is necessary for attracting more users and thus, ensuring their economic survival.

The architectures of these federations usually follow a wrapper-based centralized mediation approach. Nevertheless, the growth of DL collections and the increase in the number of DLs joining federations have rendered these architectures obsolete. Almost every major DL is evaluating new architectures to replace its old systems. For instance, a kind of P2P architecture will be evaluated within the framework of the BRICKS [2] European Integrated Project (peer nodes are called Bricks nodes in this project), whereas the GRID architecture will be evaluated within the DILIGENT [8].

Obviously, existing DL federations will benefit a lot by the above architectures as the improved search and browse response-time will enable them to form even larger federations. On the other hand, even today's hardware and software architectures (e.g., ultra fast SANs) do provide the means for building fast federations, yet DLs are still reluctant in unconditionally joining them. It seems that apart from the scalability problem, there are additional ones inhibiting the creation of large federations. DLs prefer to keep their systems completely autonomous. They want their nodes to be treated as black boxes, meaning that remote nodes should make no assumption on the contents, status and capabilities of their systems. Exporting this information to distant nodes hurts the autonomy of DLs, which in turn reduces their flexibility and increases the burden

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 223–235, 2005.

of their work. An additional problem is that DLs are usually competitive institutions, therefore, the proposed distributed architectures should natively respect and support their business requirements.

The main contribution of this position paper is the presentation of a query processing schema, which may be setup over a P2P or GRID-based network architecture. Our proposal respects the autonomy of existing DLs and natively addresses their business model. The rest of the paper is organized as follows: In section 2, we discuss the business requirements of DLs. In section 3, we present our query processing architecture. In section 4, we discuss any relevant work before concluding.

2 Digital Libraries Federations Requirements

In the introduction we argued that DLs are reluctant in forming federations because they are not sure that these systems comply with their business model and respect their autonomy. In the following paragraphs, we briefly examine these requirements focusing on the problems they create to the two most prominent future DL architectures, i.e, P2P networks and the GRID.

Business Model - Content Sharing. Economic prosperity of Digital Libraries is usually bound to their ability to sell information (content, annotation and metadata) and data processing services to their users. These are in fact the only assets DLs hold, making them reluctant to give away any data to third-party entities, especially if this is done over the Internet. For instance, in BRICKS many institutions will not export or mirror their data to the common BRICKS community network but instead will allow access to their data and legacy systems through the use of conventional wrappers. Employing a strict security and trust policy in every network node and using state-of-the-art content-watermarking techniques reduces the reluctance of DLs in sharing their data. Nevertheless, experience shows that no security system is perfect and DLs are aware of this fact.

The reluctance of DL to share their content creates a lot of problems in architectures following the GRID paradigm, since the latter model the process of evaluating queries as a task of moving the data (content) to one or more processing GRID-nodes, and then starting the actual execution of these queries. Obviously, a completely new query processing architecture must be used that will minimize the physical movement of (unprocessed) data.

P2P-based systems are also affected by the content sharing restriction problem. Building a metadata P2P indexing service, using a Distributed Hash Table (DHT), will distribute the metadata of a DL to multiple, potentially less trusted nodes, including other competitive DLs. This is not something that a DL's manager will easily approve. A solution would be to use a double hashing technique, i.e., build a DHT of the hash value of the metadata instead of the metadata themselves. In this way, nodes will know the hash value of the metadata but not the exact metadata. If the users make only keywords-based queries, this approach will be satisfactory. But if advanced query capabilities are required, a traditional query processor that uses DHT indices and requires the physical movement of data, just like the GRID architecture, will have to be used. Thus, even in the case of P2P-based systems, a new query processing framework is required that will also minimize the physical movement of unprocessed information.

Business Model - Integration of Query Processing and Accounting. Consider a small federation of two DLs where the first DL holds digital pictures while the second one has information concerning poetry. Assume that a user of this federated system is looking for pictures that were created by painters that have also written certain types of poems. This query combines pieces of information from both DLs, yet only retrieves/browses content from the first one. Since DL sell (any possible piece of) information, it is a matter of the billing policy of each DL, whether the user should be charged only for the retrieved content, or also for the information of the second DL that was used during query processing. If DLs choose to charge for any information they provide, which we expect in the future to be a typical business scenario, the query optimizer and the accounting system should be closely integrated.

Competitive Environment. The most important business requirement of future DLs federations is that these should be compatible with the competitive nature of the DLs market, i.e., information is asset and data should be treated as commodities for trading in a competitive environment. Competition creates problems in DL federations, as it results in potentially inconsistent behavior of the nodes at different times. The query processing architecture should be capable of handling cases where remote nodes expose imprecise information. Such behavior is typical (and allowed) in competitive environments.

Autonomy. A requirement of modern DL federations is that distributed architectures should respect the autonomy of DLs and treat them as black boxes. Middle-wares and wrapper-based architectures help in preserving the autonomy of remote nodes. However, during query processing and optimization, existing proposals require, *a priori*, certain pieces of information (e.g., data statistics, remote nodes status (workload), nodes capabilities (e.g., which variables must be bound), operators (e.g., union, join) cost functions and parameters) that clearly violate their autonomy. A proper query processing architecture that respects DL's autonomy and work only with information that nodes expose during query processing.

3 The Query Trading Architecture

3.1 Execution Environment

We have recently proposed a new query processing architecture [18] that meets the scalability and autonomy requirements of future DLs and perfectly matches their business requirements. Figure 1 presents a typical example of the proposed architecture. It shows a network of five autonomous DL nodes ($N_1 - N_5$). Each of them, stores its own information (data and metadata) and may additionally store copies of other nodes' data and metadata, if such a thing is allowed by the policy of these distant nodes. For instance, in Fig. 1, both nodes N_2 and N_3 have information on "Greek Documents" for the period 500BC-400BC. This redundancy may be necessary for load-balancing or robustness reasons, or it may be simply the natural result of competition between nodes N_2 and N_3.

The only requirement of our architecture is that there must be a kind of directory service holding information on which data each node locally holds. For small networks,

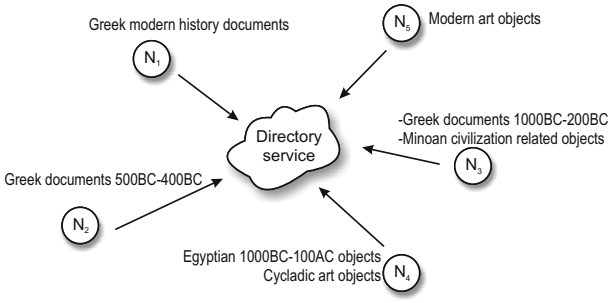


Fig. 1. Architecture overview

this can be implemented using a centralized mechanism (e.g. an LDAP server). For larger networks, the directory service can follow the P2P and DHT paradigm. Each node must register in the directory service a (high-level) description of the data it holds. In Fig. 1 we have drawn the contents of the directory service next to each node. We should note that the directory service is used only in the initial phase of queries' evaluation to locate relevant (to the queries) DL nodes. The selection of the actual nodes that will evaluate these queries is handled by the economics-based mechanism presented in the next subsection. This is different to a traditional P2P search engine. The latter contains information on all the objects (and their properties/attributes/terms) of the distributed system, whereas our directory service contains more high-level information such as which nodes hold information concerning (e.g.) middle-age pictures.

3.2 Query Evaluation Mechanism

During query evaluation, we treat DL nodes as black boxes, assuming nothing on their workload or capabilities. The only information required is the one available thought the directory service. Execution of distributed queries is handled by splitting them into pieces (sub-queries), forwarding them to nodes capable of answering them, and then combining the results of these queries to build the answer of the distributed query.

To make our architecture more concrete, continuing the previous example (Fig. 1) we assume that a user at node N_1 asks for the URLs of every ancient Greek document written in Athens between 500BC and 498BC. Since DL nodes are black boxes, N_1 can do nothing better than asking the rest DL nodes for any piece of information that might be of some help in evaluating the query. As Fig. 2 shows, network congestion is avoided by having N_1 sent its "request for help" only to the directory service, which in turn, only forwards this request to the relevant (to the query) nodes. Any node answering to this request sends its reply directly to the initial node (N_1), which further reduces network bandwidth consumption.

In the example of Fig. 2, we assume that N_2 offers to return to N_1 a URL list of all relevant documents written between 500BC and 499BC in 25 seconds and the rest documents (498BC) in 20 seconds. Similarly, node N_3 offers the same information in 50 and 15 seconds respectively. Obviously, node N_1 query optimizer will choose node N_2 for all URLs concerning documents written in 500-498BC and N_3 for the rest ones.

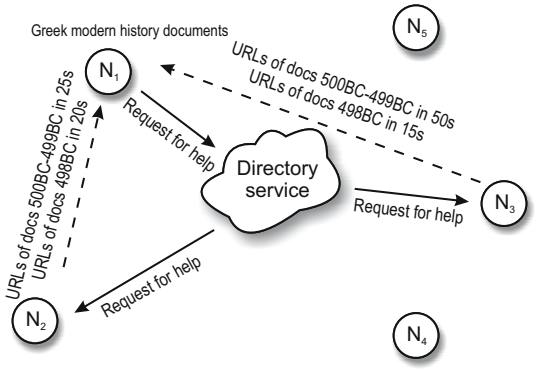


Fig. 2. Example of query processing

That is, the query processor of N_1 effectively purchases the answer of the original query from nodes N_2 and N_3 for 20 and 15 seconds respectively.

The above example shows the main idea behind the query processing architecture that we propose. It is inspired by e-commerce technology, recognizes queries (and query answers) as commodities and approaches DL federations as information markets where the commodities sold are data. Query parts (and their answers) are traded between DL nodes until deals are struck with some nodes for all of them. Distributed query execution is thus modeled as a trading of parts of query-answers. Buyer nodes (e.g., N_1) are those requiring certain pieces of information in order to answer a user query. Seller nodes (e.g., N_2 and N_3) are those offering buyers this missing information.

Although the idea is simple, it is difficult to construct an algorithm that can optimize the trading of queries and queries answers. For instance, assume that in the previous example, node N_3 offered the URLs of the documents written in 500BC,499BC and 498BC separately for 20s, 30s, and 15s respectively. In this case, node N_1 has many different ways of combining the offers of N_2 an N_3. In fact, it might worth for N_1 to negotiate with node N_2 the case of N_2 also returning the URLs of the documents written in 500BC and 499BC separately, before node N_1 decides on which offers can be combined in the best way.

3.3 General Trading Negotiation Parameters

There are a lot of parameters that affect the performance of a trading framework such as the one described in the previous sub-section. For details on these parameters see [1,3,4,11,13,15,17,18,21,22,25]. We briefly describe the most important ones:

Negotiation protocol. Trading negotiation procedures follow rules defined in a negotiation protocol [25]. In each step of the procedure, the protocol designates a number of possible actions (e.g., make a better offer, accept offer, reject offer, etc.) that a node may take. In the previous example, we assumed that the protocol used was *bidding* [23]. This protocol is simple but obviously cannot work when the number of nodes is large. In larger networks, plain bidding would lead to network flooding problems. In

this case, a better alternative is to use an agent-based or P2P-based [15] *auction*, which reduces the number of exchanged network messages. If the items/properties negotiated are minor and the nodes participating in the negotiation are few, then the oldest known protocol, *bargaining* can be also used.

Strategy. In each step of the negotiation procedure and depending on the negotiation protocol followed, nodes have multiple possible actions to choose from. It is the *strategy* followed by each node that designates which action is the best one. The strategy can be either cooperative or competitive (non-cooperative). In the first case, nodes try to maximize the join utility of all nodes that participate in the negotiation. In the second case, nodes maximize their personal utility. Our architecture supports both types of strategies. In cooperative ones, nodes expose information that is accurate and complete. In competitive setups, nodes expose information that is usually imprecise. For instance, a node may lie about the time required for the retrieval of a piece of information.

User preferences and items valuation. In section 3.2 we gave an example where the value of the commodities (i.e., the pieces of information) offered by remote nodes was expressed in term of the time required to fetch this information. More generally, offers of remote nodes will have many different properties, including (e.g.) the time required to retrieve the information, the precision and age of the data, and its cost in monetary units. That is, the valuation of an offer is multi-dimensional (a vector of values). The user must supply a preference relation over the domain of these vectors that orders the set of possible offers. This relation is known to buyer nodes and is used during the negotiation phase (e.g., during bidding) to select the offers that best fit the needs of the user.

Market Equilibrium. In competitive environments, nodes provide imprecise information. For instance, if the preference relation is the total cost (in monetary units) of the answer, then nodes will increase the value of all pieces of information that have more demand than supply. This will cause a decrease in the demand of this information and after some time, values will stop fluctuating and the market will be in equilibrium, i.e., demand will equal supply for all traded queries. This requires all other parameters affecting the value of items to be static [6]. The designer of a system can model the market in such a way that equilibrium values force the system to have a specific behavior (e.g, altruistic nodes are not overloaded). A nice property of our architecture is that according to the first welfare theorem [20] of economics, equilibriums will always be Pareto optimal, i.e., no node can increase its utility without decreasing the utility of at least on other node.

Message congestion mechanisms. Distributed implementation of the previous negotiation protocols have run into message congestion problems [23] caused by offers flooding. This can be avoided using several different approaches such as agent-based architectures, focused addressing, audience restriction, use-based communication charges and, mutual monitoring [17,23].

3.4 The Proposed Architecture

As it was mentioned earlier, we model query processing as a query trading procedure. Although there is a lot of existing work in e-commerce and e-trading (see previous sub-

section), there in an important difference between trading queries (and their answers) and the rest commodities. In traditional e-commerce solutions, the buyer receives offers for the complete items that he/she has asked for. However, in our case, it is possible that no DL node has every piece of information required to answer a user supplied query. Sellers will have to make offers for parts of the query (sub-queries) depending on the information that each DL holds locally. Buyers will have to somehow merge these offers to produce the answer of the initial queries. Since all nodes are black boxes, most sellers will make overlapping offers and buyers will have to make multiple rounds of communication with the seller nodes to ensure that the accepted offers are not overlapping. The problem of query optimization also complicates the task of the buyer since better offers not always improve the global distributed query execution plan. In the next paragraphs, we present how query optimization works in our framework. Further details on the proposed framework and its performance characteristics are given in [18].

The distributed execution plans produced by our framework consist of the query-answers offered by remote DL seller nodes together with the processing operations required to construct the results of the optimized queries from these offers. The query optimization algorithm [18] finds the combination of offers and local processing operations that minimizes the valuation (cost) of the final answer. For this reason, it runs iteratively, progressively selecting the best execution plan. In each iteration, the buyer node asks (Request for Bids -RFBs) for some queries and the sellers reply with offers that contain the estimations of the properties of these queries (query-answers). Since sellers may not have all the data referenced in a query, they are allowed to give offers for only the part of the data they actually have. At the end of each iteration, the buyer uses the received offers to find the best possible execution plan, and then, the algorithm starts again with a possibly new set of queries that might be used to construct an even better execution plan. The buyer may contact different selling nodes in each iteration, as the additional queries may be better offered by other nodes. This is in contrast to the traditional trading framework, where the participants in a negotiation remain constant.

In order to demonstrate our algorithm, we will use Fig. 3 that shows a typical message workflow among the buyer and seller nodes when the number of nodes is small (i.e., the bidding protocol is sufficient and we don't need to use auctions) and nodes follow a cooperative strategy. In this figure, a node receives a query Q that cannot be answered with the data that this node locally holds. For this reason it acts as a buyer node and broadcasts a RFB concerning query Q to some candidate seller nodes. The sellers in their turn, examine the query and if they locally have any relevant information concerning parts of it, they inform the buyer of the properties of these parts (offers). In our example, only two nodes return some offers back to the buyer.

The buyer node waits for a timer to expire (bidding duration) and then considers all offers it has received to construct an initial optimal distributed query execution plan. It then examines this plan to find any other possible part of the query that could help the buyer further improve the distributed plan. In Fig. 3 we assume that the buyer (e.g.) found some parts of the initial query that are offered by both sellers. For this reason it starts a second iteration of the bidding procedure, this time requesting for bids on these overlapping parts. Figure 3 shows that only one seller makes an offer in the second bidding procedure. After the bidding procedure of the second negotiation is over, the

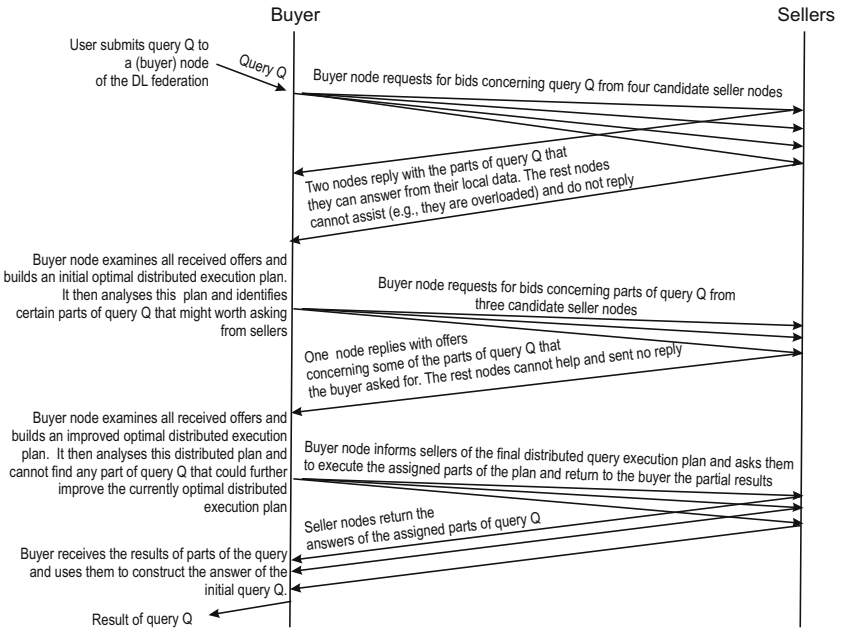


Fig. 3. Workflow of network messages between the buyer and seller nodes

buyer uses the new offer(s) to further improve the distributed plan and then re-examines it to find any other possible part of the query that can be improved. In our example, we assume that the buyer cannot find any such sub-query. Therefore, it asks from the selected remote nodes to evaluate the parts of the distributed plan that have been assigned to them and then return the results of these parts back to the buyer node. The latter uses these results to construct the answer of the initial query Q.

3.5 Processing-Tasks Trading

It is common in DLs to pose queries requiring substantial CPU processing to answer. For instance, a user may ask for all Cycladic-art picture objects that look *similar* to a specific one, where the similarity of two pictures is defined according to a user-supplied function. Answering this query will require substantial processing to evaluate the user-supplied function multiple times. For this reason in [19] we proposed a further enhancement to the previous query trading algorithm (QT) that allows it to also trade *processing-tasks*. There are three different ways to accomplish this:

Single processing-task trading. After the query trading has been completed and a distributed query execution plan has been produced, we can analyze this plan to identify processing-hungry operations that might worth assigning to distant nodes (e.g., a user-supplied picture similarity function, or some CPU-bounded join operators). We could then run a *single* round of processing-tasks trading to assign them to remote nodes.

Table 1. Comparison of different ways of query and process-task trading

Algorithm	Advantages	Disadvantages	Suitability
Plain query trading	Fastest optimization mechanism	Produces the worst query execution plans	Small queries without user-defined functions
Single processing-task trading	Fast optimization mechanism	Limited capabilities for assigning processing tasks	Large queries with non-complex user-defined functions
Iterative query and processing-task trading	Assigns many processing tasks to remote nodes	Slow optimization mechanism	Large queries with complex user-defined functions
Simultaneous query and processing-task trading	Full distribution of processing tasks	Very slow optimization mechanism	Very large queries with heavy processing requirements

Apart from the fact that we trade processing-tasks instead of queries, processing-tasks trading is similar to the plain query trading described in the previous section.
Single processing-tasks trading is especially useful when optimizing DL queries containing processing operators that should be applied after the matching rows/objects have been retrieved. An example of such a query would be the one asking for *thumbnail* pictures of all Cycladic-art objects found in 2004. Note that if this query was expressed in SQL, its *select* part would contain a user-defined function that converts the retrieved full-resolution pictures to thumbnails.

Iterative query and processing-task trading. The second approach is to run a query trading followed by a processing-task trading iteratively, until the distributed query execution plan cannot be further improved. This approach is better than the first one, since it partially integrates query and processing trading. However, it increases the time required for query optimization and thus should be preferred in queries involving processing of user-defined function before the matching DL objects have been identified. If these queries were expressed in SQL, they would have contained user-defined functions in the *where* part of the query.

Simultaneous query and processing-task trading. The last approach is to fully integrate query and processing trading, i.e., simultaneously request bids for both queries and processing. This approach yields the best query execution plans but requires excessive time for the query optimization. Therefore, it should be the preferred option for very large queries requiring substantial amounts of CPU-processing.

Table 1 summarizes the advantages and disadvantages of the three different ways of processing-task trading. This summary is the result of an excessive set of experiments presented in [19].

3.6 Parameters Affecting Query and Processing-Task Trading

In section 3.3 we discussed the parameters affecting all trading frameworks, including the ones presented in this paper. In this section, we focus our attention on the query and processing-task trading specific parameters.

Items valuation. In section 3.3 we argued the the valuation of an offer is multi-dimensional. However, this does not necessary mean that offers, made for a specific query, will differ in many attributes. It can be proved that under certain conditions in competitive (non-cooperative) environments, *if all offers for a query differ in only one property (e.g., query execution time), then in market equilibrium all nodes will make the same offer for that query*. For instance, if the query trading framework is used as a classical query optimization mechanism, then all offers for each query will differ only in the query execution time. Then, the previous proposition states that in competitive market equilibrium, all nodes for the same query will offer the same execution time. The proof of this proposition is based on the fact that (a) only a single equilibrium price exists and (b) no node will have an incentive to deviate from that market price.

Negotiation protocol. In the previous paragraph we argued that if offers differ in only one property, then in the end (at market equilibrium), all offers for the same query will be equivalent. In this case, the best negotiation protocol to use, is plain bidding or auctioning, since all offer's properties are constant and thus non-negotiable. This does not hold if offers may differ in more than one property. For instance, if the query trading framework is used for the purpose of charging users for the search and browse facilities used, then nodes' offers for the same query will differ in (e.g.) both the price and in the quality of the data offered. In this case, the preferences of the user can be better satisfied using a multi-lateral bargaining protocol, that will allow users to efficiently and simultaneously negotiate all possible price-quality combinations.

Strategy. In cooperative environments where offers for the same query differ in only one property, there is no reason to implement any strategy. If the environment is competitive, then results from the theory of games [21] should be used. Finally, the most difficult case is when offers for the same query differ in multiple properties. Then, the resulting trading framework solves the problem of *multi-objective* query optimization. Examples of strategies that work in this scenario are given in [14,7].

Market Equilibrium. In section 3.3 we argued that in market equilibrium, the allocation of resources is Pareto optimal. It can be proved [19] that usually, the requirements of the second theorem of microeconomics [7] hold for the query and processing-tasks trading framework. This theorem is the opposite of the first theorem of microeconomics mentioned in section 3.3 and in our case, states that *any load-balancing algorithm achieving Pareto optimal resource distribution in distributed DLs can be implemented using the query trading algorithm*. This last proposition shows the power of our query and processing-tasks trading algorithm.

Subcontracting. The example of section 3.2 was a rather simple case, since sellers nodes did not considered the case of constructing offers using data retrieved from third party nodes, i.e., subcontracting parts of offers. In experiments presented in [19], it was shown that subcontracting increases network messages exchanges and thus, does not

always increase the performance of the distributed system. However, in many cases, this technique is unavoidable as it is the only one allowing buyers to acquire data residing in nodes that are only indirectly (though a third node) accessible to them.

Contracting. Contracts are used in microeconomics to describe the obligations of sellers and buyers and usually define a penalty that a buyer/seller will pay to the seller/buyer if it unilaterally breaks the contract. In the trading framework, contracts can be used to model the notion of adaptive query optimization. As mentioned in section 3.4, the query trading is an iterative algorithm that initially finds and then progressively optimizes the distributed execution plan of queries. We may use the notion of contracting to allow buyers to early start the evaluation of queries (i.e., make an early contract), before the final iteration of the trading algorithm has completed. Early contracting reduces the algorithm execution time, yet, it risks the contracted query execution plan to be much more worsen than the optimal one. In this case, buyers will have to stop the evaluation of the query, wasting a lot of resources (the penalty of breaking the contract), and then restart query evaluation using the optimal execution plan (found in the last iteration of the algorithm). Note that this is one of the ideas behind the notion of adaptive query optimization. Thus, with the proposed contracting modelling, we can use the existing microeconomic theory (e.g., [14]) to predict the performance of this type of adaptive query optimization.

Quality of Service. Previously, we discussed the possibility of a buyer breaking a contact. More generally, the opposite can also happen, i.e, a seller may unilaterally break a contract. For instance, if a network failure occurs, then some sellers may not be able to fulfill their contracts with distant buyers. This possibility is handled in microeconomics using the notion of *insurance*, which can also be used in our trading framework. The role of insurance companies will be played by certain network nodes and links that will be ready to assist in query evaluation if a seller runs into some predefined difficulties (e.g., out of processing resources). Existing microeconomic theory and the theory of *choice under uncertainty* can be used to calculate the exact amounts of resources that must be reserved by the insurance nodes, so that the whole DL network exhibits a certain levels of QoS.

4 Related Work

There is a lot of work in distributed query execution and optimization over P2P systems. However, query-processing techniques employed by these systems cannot be used (directly) in DL systems, as P2P systems are typically/often limited to keyword-based searches, and thus cannot support advance predicate- or ontology-based queries. As far as grid architectures are concerned, DL node autonomy and diversity result in lack of knowledge about any particular node with respect to the information it can produce and its characteristics, e.g., query capabilities, cost of production, or quality of produced results. If inter-node competition exists, it additionally results in potentially inconsistent node behavior at different times. All these problems make traditional query optimization techniques [9,10,16] inappropriate [5,12,24] for autonomous systems such as the ones encountered in Digital Libraries. Our proposed trading framework natively handles these problems without any difficulty.

As far as the authors are aware of, the only architecture that closely resembles ours is Mariposa [24]. Nevertheless, Mariposa's optimization algorithm produces plans that exhibit unnecessarily high communication costs [12] and are arbitrarily far from the desired optimum [16,18]. Furthermore, Mariposa violates the autonomy of remote nodes as it require all nodes to follow a common cost model and asks remote nodes to expose information on their internal state (e.g., their current workload).

5 Conclusion

We propose a query processing paradigm, that respects the autonomy of DL nodes and natively supports their business model (information trading). Our framework natively supports distributed query optimization and allows for Pareto Optimal allocation of DL resources. It can be easily implemented over a typical GRID architectural infrastructure, where the GRID nodes will act as sellers and/or buyers of information and processing. For scalability reasons, a decentralized (P2P) agent-based auction mechanism and/or a P2P DHT for the directory service implementation can be used.

References

1. M. Bichler, M. Kaukal, and A. Segev. Multi-attribute auctions for electronic procurement. In *Proc. of the 1st IBM IAC Workshop on Internet Based Negotiation Technologies, Yorktown Heights, NY, March 18-19*, 1999.
2. BRICKS integrated project. `http://www.brickscommunity.org/`, 2004.
3. John Collins, Maksim Tsvetovat, Rashmi Sundareswara, Joshua van Tonder, Maria L. Gini, and Bamshad Mobasher. Evaluating risk: Flexibility and feasibility in multi-agent contracting. In *Proc. of the 3rd Annual Conf. on Autonomous Agents , Seattle, WA, USA.*, May 1999.
4. V. Conitzer and T. Sandholm. Complexity results about nash equilibria. *Technical report CMU-CS-02-135,* `http://www-2.cs.cmu.edu/~sandholm/Nash_complexity.pdf`*, 2002.*
5. Amol Deshpande and Joseph M. Hellerstein. Decoupled query optimization for federated database systems. In *Proc. of 18th. ICDE, San Jose, CA*, pages 716–727, 2002.
6. Donald Ferguson, Christos Nicolaou, and Yechiam Yemini. An economy for managing replicated data in autonomous decentralized systems. In *Proc. of Int. Symposium on Autonomous and Decentralized Systems*, 1993.
7. Hugh Gravelle and Ray Rees. *Microeconomics (3rd edition)*. Pearson Education, England, 2004.
8. Diligent integrated project. `http://http://diligentproject.org/`.
9. Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 312–321. ACM Press, 1990.
10. Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. *VLDB Journal*, 6(2):132–151, 1997.
11. John H. Kagel. *Auctions: A Survey of Experimental Research.* The Handbook of Experimental Economics, edited by John E. Kagel and Alvin E. Roth, Princeton: Princeton University Press, 1995.

12. Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, September 2000.
13. Sarit Kraus. *Strategic Negotiation in Multiagent Environments (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2001.
14. Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.
15. E. Ogston and Stamatis Vassiliadis. A Peer-to-Peer Agent Auction. In *Proc. of AAMAS02, Bologna, Italy*, July 15–19 2002.
16. Christos H. Papadimitriou and Michalis Yannakakis. Multiobjective query optimization. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on PODS, May 21-23, 2001, Santa Barbara, CA, USA*. ACM, ACM, 2001.
17. H. Van Dyke Parunak. *Manufacturing experience with the contract net.* Distributed Artificial Intelligence, Michael N. Huhns (editor), Research Notes in Artificial Intelligence, chapter 10, pages 285-310. Pitman, 1987.
18. Fragkiskos Pentaris and Yannis Ioannidis. Distributed query optimization by query trading. In *Proc. of Int. Conf. on Extending Database Technology (EDBT), Herakleio, Greece*, 2004.
19. Fragkiskos Pentaris and Yannis Ioannidis. Query optimization in autonomous distributed database systems. submitted, 2004.
20. Mark Pingle and Leigh Tesfatsion. Overlapping generations, intermediation, and the first welfare theorem. *Journal of Economic Behavior and Organization.*, 3(5):325–345, 1991.
21. J. S. Rosenchein and G. Zlotkin. *Rules of Encounter : designing conventions for automated negotiation among computers*. The MIT Press series in artificial intelligence, 1994.
22. Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
23. Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.
24. Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeller, Adm Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48–63, 1996.
25. Stanley Y.W. Su, Chunbo Huang, Joachim Hammer, Yihua Huang, Haifei Li, Liu Wang, Youzhong Liu, Charnyote Pluempitiwiriyawej, Minsoo Lee, and Herman Lam. An internet-based negotiation server for e-commerce. *VLDB Journal*, 10:72–90, 2001.

Moving Digital Library Service Systems to the Grid

Leonardo Candela[1,2], Donatella Castelli[1], Pasquale Pagano[1,2], and Manuele Simi[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo",
Consiglio Nazionale delle Ricerche,
Via G. Moruzzi, 1 - 56124 PISA - Italy
{candela, castelli, pagano, simi}@isti.cnr.it
[2] Università degli Studi di Pisa, Dipartimento di Ingegneria dell'Informazione,
Via G. Caruso, 1 - 56122 PISA - Italy
{candela, pagano}@iet.unipi.it

Abstract. The architecture of a digital library service system strongly influences its capabilities. In this paper we report on our experience with a distributed digital library system, OpenDLib, and we describe why, in the attempt to better satisfy the user requirements, we decided to develop DILIGENT, a service-oriented digital library infrastructure on the Grid. The paper describes and compare the two systems by focusing, in particular, on that part of the architecture that controls and supplies the necessary features for creating and managing digital libraries.

1 Introduction

Four years ago, the DLib group at ISTI-CNR began to develop a Digital Library Service System (DLSS), i.e. a system for creating and managing digital libraries (DLs). When designing this system, named *OpenDLib* [8], our goal was to create a customizable system that, if appropriately configured, could satisfy the needs of different application frameworks.

Our first step was to clarify what we meant for DLs and to identify the features that a system able to implement DLs should provide. This analysis brought us to understand that a DLSS is a complex system that must not only offer powerful user functionalities (e.g. search, browse, annotation) but also implement basic functions for supporting the fruition of the user functionality and for guaranteeing the quality of the overall DL service, e.g. its availability, scalability, performance. Moreover, it must satisfy a number of other desiderata, like being extensible, easy to install and to maintain.

In order to create the conditions for achieving the required level of quality we analyzed a range of possible system architectures and, finally, we decided to adopt a distributed, dynamically configurable, service-oriented architecture (SOA). The implementation effort spent in developing a DL system based on this architecture turned out to be much greater than that required for implementing a centralized one since a number of services dedicated to the co-ordination, management and optimal allocation of the different service instances had also to be provided. OpenDLib is now an operational system that has been used for building a number of DLs [21–24]. Each of these DLs has its own specific distributed architectural configuration that reflects the needs of the application scenario where it operates. In all these different experiences the distributed

C. Türker et al. (Eds.): P2P, Grid, and Service Orientation ..., LNCS 3664, pp. 236–259, 2005.

service architecture has proved to be a valid instrument to satisfy a number of requirements that could not have been met otherwise. The greater development effort has thus be highly compensated by the better quality and organization of the DL functionality exposed to the users.

New architectural approaches have emerged, or have been consolidated since we designed the OpenDLib system, e.g. Web services [5, 10, 20], P2P [19], Grids [15, 16]. All these approaches provide features that simplify the realization of a distributed DL architecture by offering a standardized means of building software services that can be accessed, shared, and reused across a network.

The DLib group at ISTI-CNR, with a number of other European research organizations and software companies, has recently initiated a new project, *A test-bed Digital Library Infrastructure on Grid ENabled Technology* (DILIGENT), which, by exploiting these new approaches, will develop an infrastructure for supporting a new method for the creation and operation of DLs. By using this infrastructure user communities will be allowed to create multiple on-demand transient DLs active on the same set of shared resources. This infrastructure will have a new architecture which integrates the service-oriented approach with the middleware provided by EGEE [12], the project that will deliver the largest European Grid Infrastructure.

In this paper we introduce the OpenDLib architecture by focusing on the aspects related to the services management and we report on our experience in operating this system. Then, we describe why we decided to move towards the new DILIGENT infrastructure, and present its architecture and the services that are responsible of the dynamic creation and optimal handling of the DLs.

The rest of this paper is structured as follows: Section 2 introduces the main requirements that motivated our choice of a distributed service architecture; Section 3 illustrates the main functional areas of an architecture for DLSSs; Section 4 describes OpenDLib and, in particular, its architecture in terms of the identified areas; Section 5 discusses the motivation that brought to the introduction of the new DILIGENT infrastructure; Section 6 describes the architecture of this infrastructure and, finally, Section 7 concludes.

2 Requirements for a Digital Library System

We have been working on DL applications since 1995. During this period we have met several potential user communities and have discussed their requirements with them. Some of these requirements have strongly influenced the architectural design of both OpenDLib and DILIGENT. In particular:

1. There is a set of core DL functionalities, such as search, retrieval, access to information objects, that any DL should provide. The format in which each of these functionalities is presented to the user is usually different since it complies with the application specific vocabularies and rules. In addition to the core functionalities, each DL, usually, *must provide other specific functionalities for serving application-specific requirements*.

2. During the DL lifetime new organizations may join the DL by bringing their content and additional functionalities may be required to satisfy new needs. *Therefore, a DL must be able to dynamically evolve by adapting itself to these new situations*.
3. The handling of a DL can be expensive in terms of financial, infrastructure and human resources. The adoption of a *DL federated model* is proposed as a solution to this problem by many organizations. By following this model, multiple organizations can set up a DL by sharing their resources according to their own policies. For example, they can decide to share and distribute the search services but store their information objects locally.
4. *Access to content and services is usually regulated by policies*. These policies can, for example, specify that a collection of objects is only visible to a particular group of users, or that a service can only be accessed free of charge for a given time interval.
5. *The users of a DL require a good quality of service (QoS)*, i.e. an acceptable level of non-functional properties such as performance, reliability, availability and security.

In order to satisfy these requirements, we chose to rely on a service-oriented architecture [13], i.e. on an architecture in which all the functionalities are realized as independent services with well-defined interfaces. This organization, where services are enabled to expose their interface and service consumers are entitled to find the more appropriate ones, provides the necessary conditions for supporting federation, openness and dynamic evolution.

Having chosen this architectural paradigm, we began to investigate the typology of functionalities that a DLSS should have been provided with. As a result of this investigation we introduced the logical reference layered architecture described in the next section.

3 A DLSS Layered Architecture

As a result of our study on the functionality that should be provided by a DLSS, we decided to conceptually organize its services into layers as shown in Figure 1.

The lower layer of our architecture, the *Collective Layer*, consists of the services that implement the basic functionalities needed for creating and operating the DLs. These functionalities are global in nature, i.e. they are not related with a particular service. In particular, they support the gathering, storage, and publishing of information about services in order to supply the necessary integration and mediation among them.

The *DL Components* layer contains the services that provide the specific DL functionalities. In particular, this layer comprises a set of services that implement mandatory DLs functionalities i.e. submission, indexing and discovery of mixed-media objects (e.g. texts, videos, images, environmental data), management and processing of these objects through annotation, composition, and cooperative editing. This layer is highly *open* as it accommodates all the content and application services that will be made available by the system during its lifetime. The addition of new services and their usage within the system is under the control of the functionalities offered by the services of the Collective Layer area.

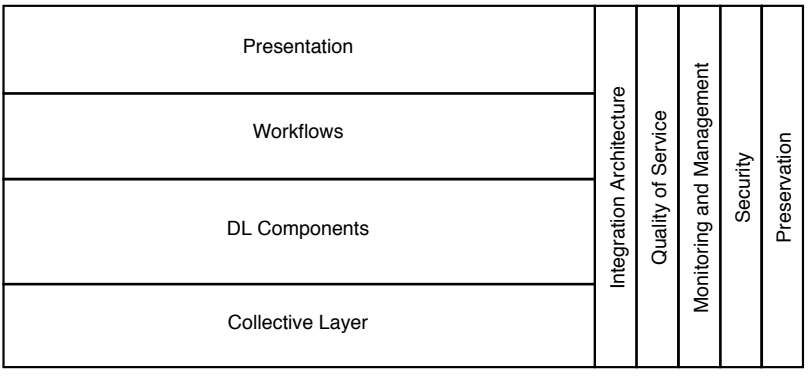


Fig. 1. The Layers of a SOA based DLSS

The *Workflows* layer offers the mechanisms for combining the underlying services in order to form a complex flow of communication among them and thus enable the construction of "new functionalities" as compositions of services that act together as a single application.

The upper layer, i.e. the *Presentation* layer, is more user-oriented. It permits to decouple the user interface from the system components that implement the DL. In particular, it represents the access point of the users to the system thus covering one of the main functionality that each DL must offer, i.e. the mediation between information objects and information users.

In order to manage the set of dynamic, customizable and independent services described, the DLSS should provide an additional set of functionalities which are depicted as vertical layers in Figure 1. This management activity, aimed at ensuring the desired quality of service, is not a trivial task. It involves many functions such as: *security*, e.g. authorization of the request, encryption and decryption as required, validation, etc.; *deployment*, allowing the service to be redeployed (moved) around the network for achieving a better performance, and a greater redundancy for availability, or other reasons; *logging* for auditing, metering, etc.; *dynamic rerouting* for fail over or load balancing and *maintenance*, i.e. management of new versions of a service or new services to satisfy new users needs. These management functionalities are offered by the system and are inherited by all the managed services, in particular by those provided by third-parties.

The *Integration Architecture* enables the integration of the various services through the introduction of a reliable set of capabilities often called Enterprise Service Bus (ESB) [13]. These capabilities, includes (*i*) message transformation, i.e. transforming data into a common data format that is understandable by both the sending and the receiving services, (*ii*) intelligent routing, i.e. a mechanism freeing the sending service from having knowledge of the location of the service a message is direct to, and (*iii*) publish and subscribe mechanisms, i.e. support for an event-driven model in which an event that occurs in a service can trigger an action in another one that is registered for that notification.

The *Quality of Service* represents a sort of contract among the consumer and the producer of a service. QoS requirements are conveyed in terms of high-level parameters that specify what the user requires. The requirements we mean are: *performance*-expected performance characteristics are needed to establish service commitments; *synchronization* - characterizes the degree of synchronization required between related services, events, or information flows; *level of service* - specifies the degree of service commitment required to maintain performance guarantees; *cost of service* - is the price a user is willing to incur to obtain a level of service; *QoS management* - is the degree of QoS adaptation that can be tolerated and scaling actions to be taken in the event the contracted QoS cannot be met.

The *Monitoring and Management* represents the set of activities needed for supporting the QoS. In particular the monitoring functionality allows the system to have a picture on the status of the services and the management functionality allows the system to rearrange the status of the services, e.g. create new service instances.

Security is one of the most critical functionality in an environment based on controlled sharing where services and/or users may belong to multiple organizations and spans across multiple locations.

Preservation is a broad area. In this context we mean the set of policy, protocols, best practices and activities that aims at making a resource, in particular the digital objects and the knowledge, available for future uses.

In this article we focus our attention on the most basic layer of the introduced architecture model, the Collective Layer. We show how its notion and functionality evolved from the OpenDLib system to DILIGENT and the effect that this change had on the overall functionality perceived by the final DLs users.

The next section begins this presentation by describing the OpenDLib system and its architectural framework.

4 The OpenDLib System

The objective of the OpenDLib project was to create a software toolkit that could be used to set up a digital library by instantiating the software according to the requirements of a given user community appropriately and then explicitly submitting new documents or harvesting the content from existing sources.

As regards archives and documents, the OpenDLib can handle a wide variety of document types with different format, media and structure. In particular, it can manage new types of documents that have no physical counterpart, such as composite documents consisting of the slides, video and audio recordings of lectures, seminars or courses. OpenDLib can also maintain multiple editions, versions, and manifestations of the same document, each described by one or more metadata records in different formats. The documents can then be organized in a set of virtual collections, each characterized by its own access policy. Authorized people can define new collections dynamically by specifying appropriate definition criteria. Regarding the DL functionality, the basic release of OpenDLib provides services to support the submission, description, indexing, search, browsing, retrieval, access, preservation and visualization of documents. In the rest of this section we first present the organization model underlying these services

and, then, we detail two main components of the system that play a fundamental role in the realization of such model.

4.1 The Architecture

The OpenDLib architecture consists of an open and networked federation of cooperating services[1] compliant with the reference architecture reported in Section 3.

The coordination among the services has a central role since in OpenDLib the services interactions are more complex than in a traditional client-server application. In fact, the role of a service in different contexts may be different since it can act both as a provider and as a consumer. Moreover, use relationships may exist a priori among any subset of the services and the services can be combined in different ways to support different functionality; the same services may be used in different ways, in accordance with the restrictions placed on their use and the goal of use.

A communication protocol, named OpenDLib Protocol (OLP) [9], has been designed in order to regulate the communication among the services. This protocol imposes an established set of rules governing how information is exchanged in the system. These rules must be satisfied both by the consumer and by the producer of the information.

From the conceptual point of view the services that implement the OpenDLib infrastructure are organized in the layered architecture shown in Figure 2.

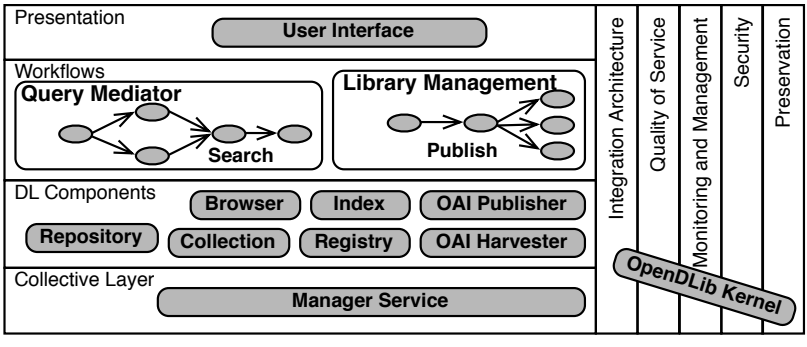


Fig. 2. The OpenDLib Layered Architecture

The *Presentation* layer, composed by the User Interface (UI) service, has been designed to simplify human-DL interactions. It is highly configurable so that it is possible to choose among multiple modes to access to the documents and the result sets. Moreover, the working session can be personalized by specifying the information space

[1] Hereafter, with the term "service" we mean an OpenDLib software module that supplies a certain task via a well-defined interface. Each module is able to communicate with other modules and can be deployed and hosted on a server.

where the user requests must be evaluated, the query language to be used, and the language of the user requests.

The *Workflows* layer is composed by two services, the Library Management and the Query Mediator services. The former service supports the document publishing and revision workflows. This service can also be personalized to support specific management workflows for other user communities. It interacts with the user either through the UI or through the support of the e-mail notification toolkit, with the Registry to authenticate and authorize user requests, and with the Repository to finalize the user requests. The Query Mediator is a generator of query management workflows; it satisfies user requests by coordinating the appropriate set of Index, Collection, and Registry service instances.

The *DL Components* layer is composed by the services that provide the DL basic functionalities. This set of functionalities includes: management and translation of metadata to achieve interoperability among heterogeneous content providers; archive distribution and virtualization; and distributed search, access, and discovery. All the OpenDLib services of the current release are highly configurable. It is possible, for example, to select the metadata formats and the query language operators that are to be accepted by an Index, the publishing and service hosting institutions, the number of service replica, etc. This provides a great flexibility and permits the system to be used in a variety of different DL application frameworks.

A brief description of the services belonging to the *Workflows*and *DL Components* layers is reported in Table 1.

The *Collective* layer, composed by the *Manager Service*, maintains a continuously updated status of the networked federation of services, checks their consistency and controls the flow of the communication. A detailed description of this service is reported in Section 4.2.

In OpenDLib the *vertical* layers functionality is provided by the *OpenDLib kernel* module which is embedded in each application service. This module implements a high level abstraction of the communication protocol. A detailed description of this module is given in Section 4.3.

The OpenDLib services can be centralized, distributed or replicated on different hosting servers. These services are organized in *regions* each of which implements a DL application. An OpenDLib DL application thus usually comprises multiple instances of the same service type hosted on remote servers of different organizations. Instances of services like Repository, OAI Harvester and Publisher, Library Management, Index, and Browser may belong simultaneously to multiple regions, and therefore to multiple DL applications, whereas the instances of UI, Collection, Registry, and Query Mediator, being DL application specific, belong to one single region.

This distribution provides an appropriate context for supporting the user demand of DLs following the federated organizational model and for ensuring quality attributes such as performance and scalability.

OpenDLib supports three kinds of dynamic expansion: 1) new classes of services can easily be added to the federation; 2) new instances of a replicated or distributed service can be mounted on either an existing or a new hosting server; 3) the configurations of the services can be modified so that they can handle new document types, new

Table 1. OpenDLib application services

Service name	Main performed tasks
Repository	Stores and disseminates documents that conform to the powerful DoMDL [7] document model able to represent structured, multilingual and multimedia documents. Documents can be associated with different publishing institutions. Each institution can organize its documents into collections regulated by specific access policies. Physical manifestations of the documents can be stored locally, replicated on multiple storage systems, or maintained on specialized storage systems, as video streaming server; they can be accessed using the common and independent protocol that hides the heterogeneity of the real storage system used as back-end.
Library Management	Supports the submission, withdrawal, and replacement of documents through complete publish and revision workflows that can be chosen among a predefined set delivered with the standard configuration of the service. New specific workflows can be easily defined and integrated in the service. Automatic generation of wizards, constructed starting from the XML metadata schema and the configuration file, allows simplifying the interaction among authors, librarians, and the digital library.
Index	Indexes documents, accepts queries and returns documents matching those queries. It is parametric with respect to the metadata formats, to the set of indexed fields, to the set of result set formats and to the language of terms. Queries can be expressed as simple conditions or as a pool of queries interrelated by boolean and probabilistic operators. Result sets are automatically filtered to take into account virtual collections and access rights.
Query Mediator	Retrieves documents by dispatching queries to the appropriate Index service instances and by merging the result sets. It takes into account the peculiarities of the available Index instances. Queries are analyzed, optimized, and decomposed to form workflows that allow distributing parts of the human request to the correct multitudes of services.
Browser	Constructs and uses appropriate data structure for browsing the library content. It is customizable with respect to the metadata formats, the set of browsable fields, and the result set format. Result sets are automatically filtered to take into account virtual collections and access rights.
Collection	Mediates between the virtual and dynamic organization of the content space, defined by the DL community of users, and the concrete organization into basic collections of documents hold by publishing institutions.
Registry	Maintains information about the users, groups, and communities. It stores the user credentials using cryptographic techniques and allows user requests to be authenticated and user requests to be authorized.
OAI Harvester	Allows harvesting content published by archives compliant with the OAI-PMH protocol for metadata harvesting. Harvested content can be automatically elaborated, manipulated, and published in the Repository or can simply be discovered and accessed in its original storage system.
OAI Publisher	Allows the content of an OpenDLib DL to be published through the OAI-PMH protocol for metadata harvesting. It allows a multitude of geographically distributed Repositories to be published as a single archive. The response to the OAI protocol calls are built on demand without duplicating information.

metadata formats and support new usages. By exploiting these kinds of expansion new application specific needs can easily be satisfied.

In the following two subsections we illustrate the mechanisms that have been introduced in OpenDLib to support the management functionalities of the Collective and vertical layers. In particular, we describe the *Manager Service* and the *OpenDLib kernel* module.

4.2 The Manager Service

As previously pointed out, OpenDLib supports different kinds of dynamic expansion. When one of them occurs, even if it concerns a specific service instance, an automatic service reconfiguration of other instances is required in order to make them capable to take into account the characteristics of the new service. For example, when a Repository instance is modified in order that it can accept a new metadata format, at least one Index instance must be updated to index the new format; when a new Query Mediator instance is set up to reduce the workload on the existing Query Mediator instances, then a certain number of UI instances must change their communication flow and address their service requests to the new instance.

Similar updates are needed when the conditions of the underlying network change, e.g. when there is a network failure, when the number of requests sent to a service instance exceeds an established threshold, etc.

All the above automatic updates in the configuration of the federation are controlled by the Manager Service, which derives the best routing strategy required to achieve a good QoS following established algorithms.

The Manager maintains a continuously updated status of the networked federation of service instances, checks their consistency and controls the flow of the communication through intelligent routing. From the architectural point of view, it can be replicated and distributed on a multitude of instances in order to enhance its robustness, availability, and responsiveness. The Manager service instances can be configured as master or slave instances. The former type of instances maintain and manage information about all the service instances despite of their organization in regions. They can be replicated and their consistency is maintained by appropriate synchronization mechanisms. The slave instances harvest information about a specific region from one of the master instances and manage only the set of the service instances belonging to their region. Multiple slaves can be configured for the same region and also in this case the same synchronization mechanisms are used to update the information between slave pairs.

The Manager master is partially configured by the DL administrator at the start-up of a DL application. Its configuration parameters contain the minimum information required to specify the topology of the region that represents the DL application, e.g. the address of the hosting servers; the list of the services and whether they are centralized, replicated or distributed (each service can be configured in one of these ways to serve user communities at best); the number of instances for each service; their allocation to the servers, etc. Configuration parameters contain also a number of consistency rules that specify the legal configurations of the service instances in the federation. These rules strictly depend on the type of service and on the "use" relation that links them. For example, the language of the terms in a query that is processed by the Query Mediator

must be one of the languages indexed by the used Index services, the document and metadata descriptions submitted to a Library Management Service must conform to those managed by the corresponding Repository.

By exploiting the information about the DL architecture acquired at the DL application start-up time, the master Manager begins to collect more detailed information about the service instances by periodically sending them appropriate protocol requests. It stores and processes the information gathered, controls its consistency, and takes decisions about the organization of the federation, for example about the rerouting of the communication among the service instances. These decisions are continuously updated according to the new value of different parameters returned by the monitoring activities on the node and service instances forming the DL.

All service instances notify the Manager of any changes in their configurations and of the status of the whole service federation. The Manager updates the stored architectural map and executes the necessary steps to collect information about any new instance. On the other hand, the service instances periodically harvest information about the federation from the slave instances of the Manager. For example, each service that uses a distributed service *X* asks the Manager the address of the most appropriate instance of *X* that can serve its requests.

The configuration of each OpenDLib service is automatically generated by the service itself by exploiting the information that it receives from the Manager and the information that it collects by sending requests to the instances known through the Manager. Once configured, the various instances can start the co-operation required to process the DL user requests.

4.3 The OpenDLib Kernel

When designing a service-oriented architecture a common task is to provide a high level abstraction of the communication among the distributed services. In OpenDLib this abstraction is implemented by the OpenDLib kernel. This is the basic software layer upon which the OpenDLib services are built. It includes tools and software modules that enable any service to use the OpenDLib Protocol and to communicate with each other, no matter where its physical location is.

In order to achieve this goal, a small *Message Dispatcher* module is installed on each OpenDLib network node. This module is in charge to dispatch the incoming messages to the appropriate services hosted on the node. When a service sends a request to a remote service, it actually interacts with the dispatcher module of the target node. The request received is interpreted and converted into a format that can be processed by a generic OpenDLib service.

Another main component of the OpenDLib kernel is the *ProtocolManager* module. It is in charge of building a service request according to the rules of the protocol. It also permits other services to interact both with remote and local services in a transparent way. The remote service instance to which the request is to be addressed is identified through the *Router* module; this is the kernel component that provides features to support the automatic routing of the requests. The dispatching of the requests to local services has been achieved through the careful design of the services interface. This

design produced a solution that facilitates the conversion of a network message into a local one if the producer and the consumer of the message are hosted on the same node.

Finally, the kernel provides the *Publish* and the *Subscribe* modules. The former provides functionalities, accessible through the Info kernel service, to discover information about the configuration of the hosting node, the list of services deployed on it, and the status of each service, enabling the Manager service to monitor each service instance. The latter provides the mechanisms that allow service instances to contain always updated information on the configuration and status of the federation of service instances disseminated by the Manager Service.

Figure 3 shows a portion of a DL application where the same services are deployed on different nodes. In the depicted diagram we have numbered the associations among software components by presenting three typical situations:

1. The Manager service wants to collect information about the status of an hosting node. It sends a request to the identified node using Protocol Manager. This request is received by the Message Dispatcher that forwards the request to the Info service entitled to answer to it.
2. The Index service wants to maintain an updated knowledge about some information distributed by the Manager . It uses the Subscribe module that generates a request to the Manager via the Protocol Manager, Router, and Message Dispatcher modules.
3. The Query Mediator wants to solve a user query. It elaborates a query pool and sends it to the correct Index service identified through the Protocol Manager, Router, and Message Dispatcher modules.

As shown by the illustrated examples, by exploiting the kernel modules deployed on each node the format of the protocol is completely hidden to the services. All the services use the Protocol Manager to send their requests. They do not care about the location of the correct service instance because it is discovered by the Protocol Manager using the Router. Finally, all service requests are collected by the Message Dispatcher that pass them to the correct service instance.

Major results of this effort of making the protocol handling transparent to the services are:

- if the communication protocol needs to be changed for any reason, the modifications are localized in some specific points only;
- an abstraction layer is provided, so new services can be added more easily, avoiding to deal with communication details;
- a mechanism for optimizing the network traffic among the different nodes is provided.

5 Lesson Learned

OpenDLib is now a running system. A number of DLs [21–24] have been built using it and several others are under construction. In all these DLs the SOA architecture provides an extensible framework where application specific services can be added to satisfy the local requirements. Moreover, by supporting a federated maintenance model,

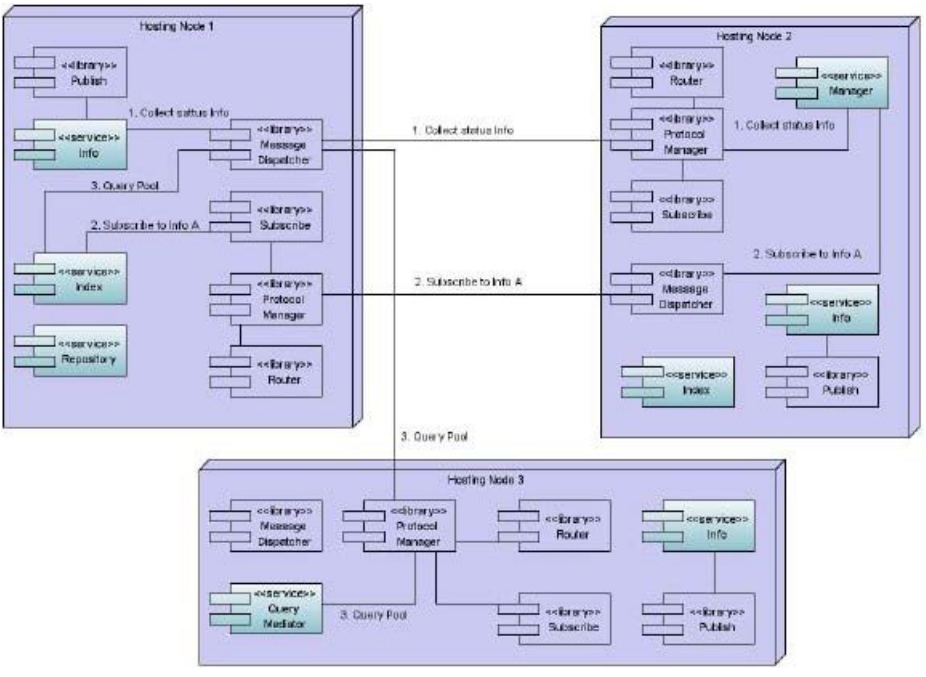


Fig. 3. OpenDLib Deployment Diagram

this architecture naturally supports the creation of large DLs by dynamically and progressively aggregating resources provided by many disperse organizations. The experience made so far has validated our architectural design choices and we now firmly believe in the appropriateness of SOA architectures as instruments for satisfying many of the requirements described in Section 2.

In the last few years, stimulated by systems like OpenDLib, which concretely demonstrates the possibility to depart from systems that simply resemble traditional library functionality, new user requirements have progressively emerged. These requirements cannot completely be satisfied by OpenDLib. In particular:

- Multimedia and multi-type content information objects are now clearly emerging as alternative and powerful communication vehicles. In OpenDLib these objects are only stored and retrieved. Complex processing of such objects is not supported because it requires a very high computational capacity that cannot be supported by most of the typical OpenDLib clientele.
- Many of the user communities that now demand DLs are small, distributed, and dynamic; they use a DL to support temporary activities such as courses, exhibitions, projects. Despite the use of a DLSS, like OpenDLib, reduces the cost with respect to an ad-hoc implementation, this cost is still too high for certain communities of users. In fact, dedicated computing resources - sometimes quite powerful as in the case of video DLs - have to be acquired in order to store and process the documents. These dedicated resources must be dimensioned to support the highest

peak of activities, even if these are executed rarely, e.g. at the start-up of the DL or periodically for preservation purposes. Furthermore, specialized technical staff with appropriate skills is required to configure, install and maintain the system.

In the four years that have been spent in implementing and experimenting OpenDLib, also the enabling technologies and the standards have evolved. Today, web services and SOA are highly diffused quite *standard* solutions for web applications. Many specifications dealing with common issues of this framework, e.g. security, resource description, etc., springs up in the WS-* family. Grid technology, offering computational and storage power on demand, is now meeting these technologies with the WSRF. P2P technologies, popularized by (music) file sharing and highly parallel computing applications (e.g. SETI@home), can be employed successfully to reach some design goals such as scalability, availability, anonymity. Certainly, these new standards and technologies provide more powerful and advanced solutions than the ad-hoc ones originally implemented by OpenDLib.

These recent advances of the technology enables the development of new systems that can better respond to the emerging user requirements. Our group, with other European research and industrial organizations, is currently involved in an EU FP6 integrated project, DILIGENT, which aims at building one of these new systems. The next section briefly introduces the DILIGENT project and discusses the elements of its architecture that will mainly contribute to the satisfaction of these new requirements.

6 The Next Step: DILIGENT

DILIGENT aims at developing a test-bed DL infrastructure. This infrastructure represents an evolution of a DLSS. While a DLSS is a system that, once set up, manages the resources of a DL (possibly provided by multiple organizations), and operates the DL during its lifetime, the DILIGENT infrastructure provides these functionalities for creating a multitude of DLs on-demand. These DLs are active on the same set of shared resources that comprises: *content sources* (i.e. repositories of information searchable and accessible), *services* (i.e. software tools, that implement a specific functionality and whose descriptions, interfaces and bindings are defined and publicly available) and *hosting nodes* (i.e. networked entities that offer computing and storage capabilities and supply an environment for hosting content sources and services).

By exploiting appropriate mechanisms provided by the DL infrastructure, producer organizations register their resources and provide a description of them. The infrastructure manages the registered resources by supporting their discovering, reservation, monitoring and by implementing a number of functionalities that aim at supporting the required controlled sharing and quality of service.

A user community can create a new DL by specifying it through a number of characterizing criteria and by invoking the functionality provided by the infrastructure. These criteria specify conditions on: the information space (e.g. publishing institutions, subject of the content, documents type); the operations that manipulate the information space (e.g. type of search, tool for data analysis); the services for supporting the work of the users (e.g. type of personalized dissemination, type of collaboration); the quality

of service (e.g. availability, response time), and on many other different aspects, like the maximum cost, lifetime, etc. As a result of this request a new DL is created by selecting, and in many cases also deploying, a number of resources among those accessible by the user community, gluing them appropriately and, finally, making the new DL application accessible through a portal. The composition of a DL is dynamic since the services of the infrastructure continuously monitor the status of the DL resources and, if necessary, change the components of the DL in order to offer the best quality of service. By relying on the shared resources many DLs, serving different communities, can be created and modified *on-the-fly*, without big investments and changes in the organizations that set them up.

The DILIGENT infrastructure is being constructed by implementing a service-oriented architecture in a Grid framework. In particular, DILIGENT exploits the the Grid middleware, *gLite*, and the Grid production infrastructure released by the Enabling Grid for E-Science in Europe (EGEE) project [12]. This is a two-year European funded project, conceived as part of a four-year programme, where the results of the first two years will provide the basis for assessing subsequent objectives. This project, which builds on recent advances in Grid technology and exploits the EU Research Network GÉANT [17], is developing the largest European Grid Infrastructure that has ever being built. By merging a service-oriented approach with a Grid technology we can exploit the advantages of both. In particular, the Grid provides a framework where a good control of the shared resources is possible. Moreover, it enables the execution of very computational demanding applications, such as those required to process multimedia content.

6.1 The DILIGENT Architecture

The positive outcomes gained with the adoption of a SOA [13] in OpenDLib convinced us to retain this architectural choice in DILIGENT as well. This choice was also reinforced by the observation of the trend in the development of the new Grid technologies that, more and more often, are based on Web Services and where plans exist to use the emergent Web Service Resource Framework [14] which unifies SOA and Grid concepts.

From the conceptual point of view the services that implement the DILIGENT infrastructure are organized in a layered architecture as shown in Figure 4.

The top layer, i.e. the *Presentation* layer, is user-oriented. It supports the automatic generation of user-community specific portals, providing personalized access to the DLs.

The *Workflows* layer contains services that makes it possible to design and verify the specification of workflows, as well as services ensuring their reliable execution and optimization. Thanks to these set of services it is possible to expand the infrastructure with new and complex services capable to satisfy unpredicted user needs.

The *DL Components* layer contains the services that provide the DL functionalities. Key functionalities provided by this area are: management of metadata; automatically translation for achieving metadata interoperability among disparate and heterogeneous content sources; content security through encryption and watermarking; archive distribution and virtualization; distributed search, access, and discovery; annotation; cooperative work through distributed workspace management.

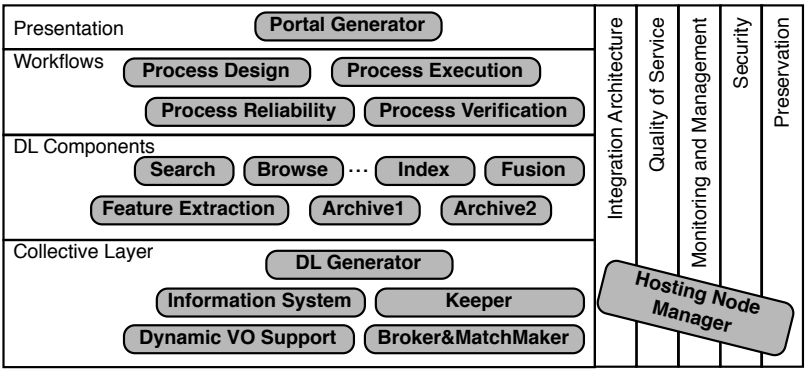


Fig. 4. The DILIGENT Layered Architecture

A list of services belonging to this layer with a brief description of the functionality they offer is reported in Table 2.

The services of the lower architectural layer, the *Collective Layer*, jointly with those provided by the gLite Grid middleware released by the EGEE project, play the same role of the Manager Service and of the Kernel in OpenDLib as they manage the resources and applications needed to run DLs. In this environment, however, the management is more tricky than in OpenDLib. The set of resources and the sharing rules are more complex since multiple transient DLs are created on-demand and are activated simultaneously on these resources. In particular, the functionalities provided by the Collective Layer are:

1. the monitoring and discovering of all the available DILIGENT resources (*Information System*)
2. the creation of the trusted environment needed for ensuring a controlled sharing of these resources (*Dynamic VO Support Service*)
3. the implementation of a global strategy offering the optimal use of the resources supplied by the DILIGENT infrastructure (*Broker & Matchmaker Service*)
4. the orchestration needed to maintain up and running the pool of resources that populate the various DLs and to ensure certain levels of fault tolerance and QoS (*Keeper Service*)
5. the support for users that want to define their DLs (*DL Generator*)

In the rest of this section we focus our attention on the services that implement the functionalities 1), 4) and 5) since they are the ones that mainly contribute to the realization of the framework that is required to satisfy the highlighted user requirements. In addition we describe the Diligent Hosting Node entity which guarantees the vertical layers functionalities. The services that realize the other functionalities are briefly described in Table 3.

Table 2. The Dynamic VO Support and Broker & Matchmaker services

Service name	Main performed tasks
Metadata Management	Supports the management of metadata for digital objects. It enables: i) the creation and modification of metadata schemas, ii) the add, update, and delete metadata functions according to the defined schemas, iii) the association of metadata sets with objects kept within the Content Management service, iv) the efficient search and access on stored metadata, and v) a notification mechanism to inform other services and applications about changes of metadata.
Metadata Broker	Provides a broker for achieving metadata interoperability among disparate, heterogeneous content sources. It provides i) wrappers for reconciling modelling differences and ii) mediators for reconciling semantic differences by means of integrated schemas.
Content Security	Supports partial encryption for distributed content in DLs and content protection by digital watermarking.
Content Management	Supports the transparent access to the DILIGENT content storage nodes as well as to external content providers. Furthermore this service also handles the update and the creation of new objects. In particular, by supplying a notification mechanism, this service publishes changes on digital objects, thus making it possible to maintain consistency among replicas, to improve freshness of search indexes, and to provide advanced filtering techniques for end-users.
Annotation	Supports flexible annotation authoring, services for annotation management as well as annotation-based information services in distributed environments.
Search	Supports the exploitation of the DL content. It supports user queries and ultimately delivers the desired results to the query author. Within DILIGENT this task is challenging due to the distributed nature of the information sources, of the heterogeneity of the ranking models and the document nature, etc. As a consequence, there is the need to combine different services to deliver that functionality, i.e. distributed indexes, source selection and data fusion.
Query Process Optimization	Receives a process workflow specification produced by the Search service and attempts to produce an optimized revision.
Index	Speeds up the retrieval process. In order to distribute the load of user queries and to parallelize expensive retrieval process, indices are stored at different Grid nodes (replication, partitioning).
Features Extraction	Supports simple, straightforward functions to make mixed-media data be quickly analyzed and transforms in to the optimal structure data for future computation.
Data Fusion	Supports the merging of different lists of retrieved documents coming from different information sources and based on different retrieval models into a single ordered list.
Content Source Description & Selection	Supports the automatic creation of descriptions of the different content resources of the DL. Based on these descriptions, given a content-based query, a selection of the most appropriate content resources to be queried is carried out at search time. This selection is based on a multi-criteria decision model, where different parameters in addition to content relevance, such as cost, connection time, etc. are considered.
Personalization Content	Enables adaptation of Grid information and services access to the needs and goals of individuals and groups. For this purpose, information about users and groups, including skills, expertise, preferences on content, services, quality of service etc., are maintained in profiles.
Distributed Visualization	Supports users in analyzing and understanding, in a graphical way, large and multi-disciplinary data collections distributed worldwide and accessible through the Grid.
Collaboration Support	Enables collaboration between members of communities by providing functionality for creating shared working spaces.

Table 3. DILIGENT DL Components services

Service name	Main performed tasks
Dynamic VO Support Service	Make it possible to create the Grid *operational context* by associating users, services, and set of resources. It enables DLs to work accordingly with the resource sharing policies and agreements. In particular, this service supplies functionality for: associating entities such as users and groups with the DL (registration, managing, monitoring, etc.); manipulating user roles within the DL; specifying agreements and policies to the DL as a whole or to individual services within the DL.
Broker & Matchmaker Service	Promotes an efficient usage of the resources offered by the DL infrastructure. In particular, it operates to realize an optimal distribution of services and resources across computing and storage nodes and to identify the more appropriate resources needed for meeting the DL requirements. This choice is achieved realizing a load balancing mechanism that takes care of the infrastructure workload and of the user community characteristics.

6.2 The DL Generator Service

In OpenDLib the DL definition phase is manually based, i.e. the DL administrator must define a DL application by specifying the list of all the services and content sources constituting the DL and their physical locations. Therefore, the DL administrator is responsible to plan the best topology of the DL application. In order to do that it must have the appropriate skills about distributed system configurations and a certain level of knowledge about the available resources. The complete virtualization of the DL application through the separation of the providers of the hardware, (i.e. hosting nodes), of the services, (i.e. DL components), and of the content engaged by DILIGENT opens new frontiers and promises to satisfy new class of user communities and unresolved user needs. However, the same virtualization raises a much greater complexity in the DL definition. In order to simplify the work of the DL creator a new service type, the DL Generator Service, has been introduced in DILIGENT. It support users in creating their own DLs by providing a semi-automatic process that is capable of supporting an efficient selection among a dynamic pool of shared resources. In particular, it:

- supports users in specifying the criteria that characterize the new DL, e.g. the information space, the required functionality, the QoS;
- selects the more appropriate federation of services and information sources required to implement a DL that satisfies the specified criteria;
- notifies the Keeper service of the identified services so that it can start the real instantiation of the DL.

In order to support these functions, the DL Generator assumes that DL user requirements and the information related with the various DL Components are modeled in terms of the elements of a Description Logic [4]. In particular, DL components are characterized by three elements:

- *Types* allow organizing components into a hierarchy that can be used during configuration;

- *Attributes* specify descriptive features, such as functional or technical characteristics, configuration parameters, etc. Each attribute has a single value or can take values from a predefined range.
- *Ports* are used to establish connections between components. Usually when defining ports, restrictions and constraints may be imposed on the type and number of components that can be connected to it. These constraints express conditions on attributes and ports that must hold in the model built to satisfy the DL requirements.

By adopting these knowledge representation mechanisms plus the inference mechanism, the DL Generator is capable to identify the set of DLComponents needed to satisfy the DL Definition.

At our best knowledge for the time being only, one formal digital library ontology exists [18]. DILIGENT plans to define its own DL ontology to model DL definitions as well as to annotate DL components accordingly.

6.3 The Keeper Service

After having identified the type of resources required to satisfy the user requirements, the DL Generator passes this information to the Keeper that concretely creates and brings together the set of DILIGENT resources that compose a DL. As its name suggests, the Keeper is the real manager of a DL; at least there must exist one Keeper for each DL and, for this purpose, when a Keeper receives the order to create a new DL, as a first step it duplicates itself to generate a specific Keeper that manages the rising DL.

The meaning of "manage a DL" in DILIGENT is really different from the one meant in OpenDLib. In OpenDLib there exists a static Manager Service that manages multiple DLs at the same time. These DLs are statically (and manually) configured by the DL administrator and the Manager Service can only optimize the usage of the preexisting and preassigned resources. In DILIGENT the resources are dynamically created when they are needed thanks to the exploitation of the Grid environment upon which DILIGENT is built. This means that when a Keeper has to create a new DL, it really creates its resources. Then, in addition, it coordinates and disseminates the *operational context* that transforms this set of distributed resources in to a single application. This context is the equivalent of the map of the region that the Manager Service disseminates in an OpenDLib DL. In the DILIGENT terminology, this context is named *DL Map* and, basically, it specifies the DL resources locations and their configurations. Any other dynamic information about a resource (e.g. its status) is maintained and disseminated by the Information System.

From the technological point of view, the resources we are discussing here are Web Service instances and related software components. All the software that we want to dynamically instantiate must be registered in the DILIGENT infrastructure using the provided functionality and must be compliant with the *DILIGENT package model specification* released with DILIGENT. If a "piece of software" respects the rules of this specification, it can be *(i)* uploaded in the DILIGENT Packages Repository, *(ii)* handled by the Keeper, *(iii)* ... and *(iv)* dynamically discovered and used by other services.

The creation of new service instances is achieved thanks to the presence of the Hosting Node Manager on each node of the DILIGENT infrastructure; among the others, the

Hosting Node Manager plays the role of a "factory" that creates new DILIGENT service by retrieving and deploying the appropriate software packages on an infrastructure node. This component is described in the Section 6.5.

Once the DL is up and running, the work of the Keeper is not finished. In fact, it is also in charge to guarantee the overall set of functionalities of the DL at any time by dynamically reallocating resources/archives and checking periodically their status. In order to support this functionality it is capable to access and investigate the state of services and resources and destroy and/or relocate them in an appropriate way making use of the information offered by the Information System.

It is clear that this behavior is a big step forward with respect to the DLSSs described in the previous sections. The possibility to dynamically move and relocate resources is a huge improvement that gives new opportunities that are not available in any non-Grid based DLSS. For instance, if the number of queries executed in a DL is very high and the execution time of each query is slow because the actual configuration of the DL does not support this number, the Keeper can simply create new Search or Index services and add them to the DL in order to improve the performance of future queries and solve the problem.

6.4 The Information System

In the DILIGENT context the different kind of resources, the hardware, the software, and the content, are provided by different organizations and with different policies and lifetime. This approach guarantees a high flexible allocation of resources with a very low cost but imposes a continuous monitoring of the state of each resource owned and managed by third parties administration.

The Information System is the service in charge to support the discovering and monitoring of the distributed resources forming the DILIGENT infrastructure. By maintaining a *ever-updated* monitoring of the whole set of available resources, single services and DLs can be enabled for self-tuning resources usage and workload-balancing maximizing the use of available resources. In particular, the functionality offered by the service are:

- gathering, storing and supplying information about the resources and services constituting the infrastructure;
- monitoring these resources and services with the appropriate level of freshness in order to be efficient and reliable.

The Information System gathers and supplies information following an approach inspired by the well-known Grid Monitoring Architecture (GMA) [25] proposed by GGF[2] that models an information infrastructure as composed by a set of *producers* (that offer information), *consumers* (that request information) and *registries* (that mediate the communication between producers and consumers).

Following this approach, the Information System acts as the registry of the infrastructure while resources and services belonging to DILIGENT act both as producers, advertising their descriptive and status information, and as consumers that discover the

[2] `http://www.ggf.org`

services and resources that they need to perform their task. The GMA solution adopted is enhanced in the sense that the DILIGENT Information System acts also as collector and it maintains information on the status of services locally, thus avoiding that consumers interact directly with producers after the discovery phase in order to acquire the information they are looking for.

The Information System contains also information about the QoS of the resources. This information is expressed by a set of attributes reporting various quality aspects encountered in distributed system. The set of parameters that can be advertised by each DILIGENT resource includes:

- Availability, i.e. the probability that a resource can respond to requests;
- Capacity, i.e. the limit on the number of requests a resource is capable to handle;
- Security, i.e. the level and kind of security a resource provides;
- Response time, i.e. the delay from the request to getting a response;
- Throughput, i.e. the rate of successful request completion.

Thanks to this information all the choices performed within the DILIGENT infrastructure about the usage of a resource w.r.t. another having comparable characteristics can be performed in a optimal way.

6.5 The DILIGENT Hosting Nodes

In DILIGENT we have introduced the concept of *DHN* (DILIGENT Hosting Node). A DHN is simply a node able to host DILIGENT services and related components. In order to become a DHN, a node (that is already part of the DILIGENT infrastructure) must have installed the following software:

Java WS Core - As DILIGENT follows the SOA paradigm, it is cleary composed by services that must be accessible via a network interface. It is not a good approach to build such software from scratch since in the the web development area there exists a great number of sophisticated open source solutions dedicated to host services. These solutions are usually referred with the term *hosting environments*. The adoption of a specific hosting environment influences the way in which services are designed, developed, packaged and distributed. Having one unique environment simplifies the distribution of the DILIGENT Services since it makes it possible to install them in the same way without providing an *ad hoc* solution for each service.
The selected hosting environment by which we are deploying the services is the Java WS Core and developed by the Globus Alliance[3]. This container provides a complete implementation of the WSRF [11] and WS-Notification working draft specifications plus WS-Addressing and WS-Security support based on the Axis Web Services engine developed by the Apache Foundation.
Java WS Core provides also a framework that supports both authentication and authorization mechanisms. In particular, the authorization features can be extended by each service in order to provide a customized level of authorization policies.

[3] `http://www.globus.org`

Hosting Node Manager - The Hosting Node Manager (HNM) is the minimal DILIGENT software that must be present on a DHN. Any other software can be dynamically deployed starting from this component. The HNM is the manager of the node on which it is hosted. The node management involves the following tasks:

- collaborate with the Keeper Service to deploy new services;
- publish the DHN configuration and status in the Information System;
- exchange data with the Keeper Service of the DL;
- maintain and expose the node configuration to the hosted services.

Each of them contributes to the dynamic creation and optimal handling of the DLs in DILIGENT. The first task provides a support to the Keeper for the creation of new services on the node, while the second one disseminates the status of the node, thus enabling the rest of the infrastructure to know the availability of the DHN, its configuration, its status and which resources it is actually hosting. These information are exploited by the Broker & Matchmaker Service that decides where the DL resources have to be deployed. The third and fourth tasks make the services that reside on the node aware of their running environment. In particular:

- *by exchanging* data with the Keeper Service, the DL Map is retrieved by the HNM that also provides mechanisms to maintain this Map updated and to export the DL Map data to the local services; thus, each service that is part of a DL has not to be aware of service relocation. It simply queries these local data and then accesses any remote service.
- *by managing* the DHN configuration, the Hosting Node Manager is the only component that must be configured. Any other software installed on the node retrieves this configuration via the subscribe/notification mechanisms implemented by the HHN.

The DHNs represent the key that gives the possibility to move from the quite static environment we have in OpenDLib to the very high dynamic one we have in DILIGENT. Using them, resources can be created and moved at any time and their consumers do not deal with this behavior since the infrastructure (the combination of Keeper and HNMs functionality) guarantees the correct dissemination of new DL Maps on all DHNs.

6.6 The Test-Bed Infrastructure

The services of the Collective Layer described in the previous sections are based on and largely exploit the functionality provided by gLite.

Its adoption has been simplified by the fact that its architecture is based on SOA whose services are compliant with WS-I[4] specifications. Thus, for example, it has been possible: i) to design the Information System as a set of interoperating services; part of them are completely new, designed and implemented by the DILIGENT partners; others are provided by the R-GMA service included in the gLite middleware; ii) to base

[4] `http://www.ws-i.org/`

the Keeper Package Repository on the File Transfer and Placement Service, the File & Replica Catalog, and the I/O server and client in order to reliably maintain and preserve packages on a set of geographically distributed Grid nodes.

From the point of view of the DL infrastructure, the resulting DILIGENT application has been initially deployed on the hosting nodes provided by the project partners. On this proprietary infrastructure the software is currently under testing and it is used by two large user communities that participate to the project.

However, as a suitable production infrastructure is a basic step in delivering any application, like DILIGENT, that aims at exploiting Grid opportunities to satisfy real user communities needs, the project will complement its proprietary infrastructure with the one released by the EGEE project. This latter infrastructure, by aggregating experts and resources, has delivered an operational grid infrastructure service available on regular and reliable basis to scientists. As reported by the EGEE information system, this production service has given access to a total of about 4.7 petabytes data and it has provided computing time corresponding to approximately 3,700 years of a single PC during the period starting from April to November 2004. Moreover, it is also expected that in a near future these numbers will grow considerably thanks to the broad diffusion of this network to new communities.

The resources made available through the EGEE infrastructure are accessible by user communities and virtual organisations according to access management policies and service level agreements to be negotiated. Once these "political" issues will be solved, i.e. an agreement between an EGEE partner/organization/VO (owner of a pool of resources) and DILIGENT will be reached, the DILIGENT application will benefit of the great number of hosting nodes available in that infrastructure and the EGEE partners will benefit of the service and content resources provided by DILIGENT. In particular, for the DILIGENT application, computational intensive functions, like content feature extraction, summarization, automatic content source description, on video, images, and sounds, which are based on complex and time consuming algorithms, will become viable with a reasonable performance with the exploitation of this powerful Grid network.

7 Conclusions and Next Steps

The paper has described the architectures of OpenDLib and DILIGENT by focussing in particular on the services of the Collective Layer. In carrying out our experience we have learnt that the quality of the services provided by a DL system not only depends on the implementation of the functionality that are directly perceived by the users but it is also strongly influenced by the several management functions implemented by the Collective Layer services. Certainly, the more distributed, flexible, dynamic, customizable the DLSS is, the more complex is the realization of these functions.

In the paper we have shown how this notion of Collective Layer services has evolved from OpenDLib to DILIGENT. In particular, we have seen how we moved from the ad-hoc solutions implemented in OpenDLib to more standard ones that exploit, among the others, the capabilities provided by a Grid middleware.

Let us conclude this paper by emphasizing that, despite the great contribution that a Grid middleware provides to achieve our purposes, work is still needed to adapt and extend its concepts to cover the functionality needs of our Collective Layer. In particu-

lar, in addition to what has been discussed until now, there are further aspects that need particular attention:

- In the DL application context, sharing of resources is acceptable only if it is highly controlled. The strong and valid security and policy mechanisms provided by the chosen Grid middleware must be extended to take into account the rules of the DL providers and consumers.
- In order to dynamically create a DL the system must be able to automatically select and retrieve the resources that better match the demand of the library creator. This requires an appropriate description of the DL resources and powerful discovery mechanisms.
- The SOA approach proposed for the DILIGENT infrastructure defines only the higher level structure of the architecture. Each service belonging to the library can internally adopt a different architecture, e.g. an Index can be realized both using a P2P technique as well as a centralized approach. This heterogeneity certainly complicates the digital library management and demands for more sophisticated algorithms.

Our next future activity will be aimed at studying these aspects in detail and at introducing appropriate solutions for handling them.

Acknowledgements

This work was partially funded by DILIGENT project (IST-004260). The design of the DILIGENT system is a collaborative effort carried out by all the partners of the DILIGENT project. We thanks all of them for their contribution to the work described here.

References

1. William Y. Arms. Key Concepts in the Architecture of the Digital Library. *D-Lib Magazine*, July 1995.
2. William Y. Arms. *Digital Libraries*. The MIT Press, September 2001.
3. William Y. Arms, Christophe Blanchi, and Edward A. Overly. An Architecture for Information in Digital Libraries. *D-Lib Magazine*, February 1997.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
5. David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. W3C Working Group Note, http://www.w3.org/TR/ws-arch/.
6. Christine L. Borgman. What are digital libraries? competing visions. *Information Processing & Management*, 38(3):227–243, 1999.
7. Donatella Castelli and Pasquale Pagano. A flexible Repository Service: the OpenDLib solution. In J. Á. Carvalho, Arved Hübler, and Anna A. Baptista, editors, *Proc. of the* 6^{th} *International ICCC/IFIP Conference on Electronic Publishing*, pages 194–202, 2002.
8. Donatella Castelli and Pasquale Pagano. OpenDLib: A Digital Library Service System. In *Proceedings of the* 6^{th} *European Conference on Digital Libraries (ECDL2002)*, pages 292–308. Springer-Verlag, 2002.

9. Donatella Castelli and Pasquale Pagano. The OpenDLib Protocol. Technical report, Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, 2004.
10. Ethan Cerami. *Web Services Essentials (O'Reilly XML)*. O'Reilly & Associates, 2002.
11. Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The WS-Resource Framework. *White paper*, 2004.
12. EGEE Team. EGEE: Enabling Grids for E-science in Europe. http://public.eu-egee.org.
13. Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Petterns: Service-Oriented Architecture and Web Services*. IBM Redbooks. 2004.
14. Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Donald F. Ferguson, Frank Leymann, Martin Nally, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling Stateful Resources with Web Services. *White paper*, 2004.
15. Ian Foster, Carl Kesselman, Jeffrey Nick, and Steve Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
16. Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.
17. GÉANT Team. GÉANT Web Site. http://www.geant.net.
18. Marcos André Gonçalves, Edward A. Fox, Layne T. Watson, and Neill A. Kipp. Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. *ACM Transactions on Information Systems (TOIS)*, 22(2):270–312, 2004.
19. Nelson Minar, Marc Hedlund, Clay Shirky, Tim O'Reilly, Dan Bricklin, David Anderson, Jeremie Miller, Adam Langley, Gene Kan, Alan Brown, Marc Waldman, Lorrie Cranor, Aviel Rubin, Roger Dingledine, Michael Freedman, David Molnar, Rael Dornfest, Dan Brickley, Theodore Hong, Richard Lethin, Jon Udell, Nimisha Asthagiri, Walter Tuvell, and Brandon Wiley. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.
20. Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002.
21. The OpenDLib Team. e-Library: a public OpenDLib instance. http://elibrary.isti.cnr.it.
22. The OpenDLib Team. The ARTE Library: an OpenDLib instance. http://arte-sns.isti.cnr.it.
23. The OpenDLib Team. The DELOS Library: an OpenDLib instance. http://delos-dl.isti.cnr.it.
24. The OpenDLib Team. The e-Science Library: an OpenDLib instance. http://e-science.isti.cnr.it.
25. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Performance Working Group, Global Grid Forum, March 2000.

Author Index