Query Processing for Peer Mediator Databases

BY

TIMOUR KATCHAOUNOV
Abstract


The ability to physically interconnect many distributed, autonomous and heterogeneous software systems on a large scale presents new opportunities for sharing and reuse of existing, and for the creation of new information and new computational services. However, finding and combining information in many such systems is a challenge even for the most advanced computer users. To address this challenge, mediator systems logically integrate many sources to hide their heterogeneity and distribution and give the users the illusion of a single coherent system.

Many new areas, such as scientific collaboration, require cooperation between many autonomous groups willing to share their knowledge. These areas require that the data integration process can be distributed among many autonomous parties, so that large integration solutions can be constructed from smaller ones. For this we propose a decentralized mediation architecture, peer mediator systems (PMS), based on the peer-to-peer (P2P) paradigm. In a PMS, reuse of human effort is achieved through logical composability of the mediators in terms of other mediators and sources by defining mediator views in terms of views in other mediators and sources.

Our thesis is that logical composability in a P2P mediation architecture is an important requirement and that composable mediators can be implemented efficiently through query processing techniques. In order to compute answers of queries in a PMS, logical mediator compositions must be translated to query execution plans, where mediators and sources cooperate to compute query answers. The focus of this dissertation is on query processing methods to realize composability in a PMS architecture in an efficient way that scales over the number of mediators.

Our contributions consist of an investigation of the interfaces and capabilities for peer mediators, and the design, implementation and experimental study of several query processing techniques that realize composability in an efficient and scalable way.

Keywords: data integration, mediators, query processing

Timour Katchaounov, Department of Information Technology. Uppsala University. Box 337, SE-75105 Uppsala, Sweden

© Timour Katchaounov 2003

ISBN 91-554-5770-3
ISSN 1104-232X
urn:nbn:se:uu:diva-3687 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-3687)
To my wife Adela.
Acknowledgements

The person I owe most for the completion of this dissertation and from whom I learned most about research is my advisor Tore Risch. My gratitude and appreciation to him for his constant help and energy, being always available for an advice, a tough discussion, or even a bug-fix. I would like to thank my opponent Tamer Özsö and the dissertation committee Nahid Shahmeri, Per Svensson, and Per Gunningberg who generously gave their time and expertise to evaluate this work.

I would also like to thank Marianne Ahrne who proof-read parts of my dissertation. Marianne, Eva Enefjord, and Gunilla Klaar were of great help in many organizational issues.

My former advisors and mentors Vanio Slavov and Vassil Vassilev not only taught me the foundations of Computer Science but also encouraged me to pursue a doctoral degree. Stefan Dodunekov suggested that I apply for a doctoral position in Sweden.

Special thanks to my fellow graduate student and friend Vanja Josifovski who helped me with my first steps in the AMOS II code, and with whom we co-authored several papers. Vanja was also very helpful in finding my first database job at the IBM Silicon Valley Lab.

Many thanks to the former members of EDSLAB Jörn Gebhardt and Hui Lin for becoming my friends and making my stay in Linköping more enjoyable. Thanks also to all current members at Uppsala Database Lab whose constant questions pushed my understanding to the limits and helped me clarify many research and technical issues.

While in Uppsala I was lucky to meet Monica, Zeynep, Brahim, Karim, Elli, Lamin and Russlan who became my friends for life. Thanks to them now I see the world from a much wider perspective. Mo, Elli and Brahim, some of my best time in Sweden was when we shared a place together.

During my first months in Sweden I was very happy to meet again Plamen, years after school. Thanks to him and Pepa I got in touch with the Bulgarian group in Uppsala who were always ready to help.

I am grateful to all my old friends from Bulgaria, they made me understand that home is where one is loved, so home for me is wherever they are.

Most of all I owe the inspiration to follow a career in research to my parents. They, and my brother, always encouraged me to follow new challenges. It is
thanks to their unconditional love and support I managed not to give up at times of deep home-sickness and I stayed to complete this project. My gratitude and love to all of you.

My dear Adi, I had to go all the way to Sweden so that we can meet. So I believe that the stress and the many lonely evenings you had to endure during the last two years were a natural part of our being together. I will never be able to thank you enough for your love and patience during this time and I look forward to our future together.

This work was funded by the Swedish Foundation for Strategic Research (contract number A3 96:34) through the ENDREA research program, and by the Swedish Agency for Innovation Systems (Vinnova), project number 21297-1.
List of Papers

This dissertation comprises of the following papers. In the summary of the dissertation the papers are referred to as Paper A through Paper F.


Papers reprinted with the permission from the respective publisher:
Other papers and reports

In addition to the papers included in this dissertation, during the course of my Ph.D. studies I have authored or co-authored the following papers and reports listed in chronological order.


## Contents

1 Introduction ................................................. 1
2 Background ................................................. 5  
   2.1 Data Integration ..................................... 5  
   2.2 Data Warehouses. .................................... 12  
   2.3 Mediator Database Systems ......................... 13  
   2.4 Peer-to-peer Systems ................................. 16  
   2.5 Query Processing and Optimization .................... 18  
3 A P2P Architecture for Mediation ......................... 23  
   3.1 Design Motivation .................................. 23  
   3.2 Requirements ..................................... 25  
   3.3 External System Components ......................... 28  
   3.4 Mediator components and their functionality ............ 33  
   3.5 Systems of Peer Mediators. ......................... 40  
4 The Problem of Query Processing in Peer Mediator Databases . . . 45  
5 Related Work ............................................. 51  
   5.1 Distributed Database Systems ....................... 51  
   5.2 Mediator Systems .................................. 51  
   5.3 Peer Data Management Systems ...................... 53  
6 Summary of Contributions .................................. 57  
7 Summary of Appended Papers ............................... 61  
   7.1 Paper A: Functional Data Integration in a Distributed Mediator System ........................................... 61  
   7.2 Paper B: Interface Capabilities for Query Processing in Peer Mediator Systems ................................. 62  
   7.3 Paper C: Scalable View Expansion in a Peer Mediator System ........................................... 64  
   7.4 Paper D: Optimizing Queries in Distributed and Composable Mediators ........................................... 66  
   7.5 Paper E: Evaluation of Join Strategies for Distributed Mediation ........................................... 68  
   7.6 Paper F: Object-Oriented Mediator Queries to Internet Search Engines ........................................... 69  
8 Future Work ............................................... 73
Introduction

The pervasive use of wide-area computer networks and ultimately the Internet provides the capability to physically interconnect millions of computing devices\(^1\). However the nodes in such global networks are designed and evolve independently of each other which results in heterogeneity at various levels, starting from the hardware platforms and operating systems to the abstract models used to describe reality. The ability to physically connect distributed, autonomous and heterogeneous computing systems on a large scale presents new opportunities for better sharing and reuse of existing computational resources and information and for the creation of new computation services and new information from the combination of existing ones. One approach to realize these opportunities is to provide abstractions above the physical network in a separate layer called \textit{middleware} that shields the users from various aspects of the heterogeneity and distribution in a global network.

A particular kind of middleware systems are \textit{data integration} systems that address the problem of heterogeneity and distribution of large amounts of data in a computer network. The main purpose of data integration systems is to provide a logically unified view of distributed and diverse data so that it can be accessed without the need to deal with many systems, interfaces and syntactic and semantic data representations. The need for data integration occurs in many diverse contexts that vary in the degree of distribution and autonomy, the level of diversity of the data sources in terms of their data model and computational capabilities, the complexity of the modeled domain, the amount and dynamics of data, performance and data timeliness requirements, and the type of queries posed.

Various data integration solutions are suitable depending on the combination of values for each of these parameters. Database technology provides high-level abstractions of data, data retrieval, and manipulation operations. Naturally, the ideas from database technology are applied to the problems of data integration so that unified views of many data sources can be specified in terms of declarative query languages. Two main approaches exist for the design of data integration systems based on database technology - the \textit{materialized approach} based on data warehouse technology and the \textit{virtual ap-}

\(^1\)According to the Internet Software Consortium (http://www.isc.org/) the number of hosts advertised in the DNS in January 2003 is 171,638,297.
approach based on the mediator concept. Data warehouse systems are centralized repositories where distributed data is collected, unified and stored in the same physical database, and is accessed without accessing the original data sources. Mediator systems [55] provide a logically unified view of the data sources (a virtual database) and the means to access and combine relevant data “on the fly” directly from the data sources. We describe the materialized and the virtual data integration approaches in more detail in Sect. 2.

Typically, database systems are designed to work in an enterprise context with a centralized organizational structure where scalability is sought in terms of the data size or number of concurrent users. Since data warehouse systems are essentially traditional DBMSs and their main concept is that of a centralized data repository for all unified data, they are suitable mainly for centralized organizations. While the mediator approach itself does not imply a centralized architecture, most existing mediator systems have either centralized or two-tier architectures that make them suitable for the same type of centralized organizations as data warehouses.

However, due to the wide-spread use of computing technology and wide-area networks, the need for data integration and the opportunities it brings are relevant in many other social contexts than centralized organizations where database technology is commonly used. Some typical examples are scientific communities, alliances of companies, groups of individuals, to name a few. These social contexts are characterized by many independent and distributed units ready to share some of the data and services they own, so that when combined with other sources, new valuable information is produced. This information can be used by others either to satisfy their needs or to further integrate more data, services and information to provide higher-level integration services. Another important characteristic is the complexity and diversity of data in terms of its degree of structure. In contrast with traditional enterprise environments where data is well structured and mostly of a tabular nature easy to represent in terms of the relational data model, many new application areas need the integration of both complex and highly nested data such as product models, and of semi-structured data such as HTML or XML documents.

Based on these observations we conclude that there is a need for a new type of data integration systems based on database technology that are suitable for the sharing and integration of large number of autonomous, distributed and heterogeneous data sources and computation services with complex data. Such a system should fulfill several high-level requirements:

R1 (autonomy): The autonomous and distributed nature of the participating entities (e.g companies or research units) should be preserved because no one owns all data sources, and most likely no single entity has the knowledge how to integrate all data sources.
R2 (decentralization): There should be no need for centralized administration because in most cases no participant would like to relinquish control to someone else.

R3 (evolution): Each of the participants' knowledge and information needs may evolve at various rates, which requires that separate parts of the system evolve independently.

R4 (flexibility): It is hardly possible to predict all social contexts where data integration may be useful. Therefore a data integration system should lend itself to easy adaptation and customization by various types of users and in various social environments.

R5 (self-management): With a large number of autonomous participants, the cost of human maintenance of a large number of integrated views of many data sources can be prohibitively high. Therefore, a large scale data integration system should be able to maintain itself automatically, ideally with no human participation beyond the management of the data sources by their owners.

R6 (scalable integration): The process of data integration requires a lot of domain knowledge and is a complex and time consuming activity that will be mainly a human task in the foreseeable future. It is important that this process can be scaled to large number of autonomous sources.

R7 (abstraction): The heterogeneity of the data sources in terms of their data models and capabilities requires that a data integration system has powerful modeling capabilities so that it can represent and integrate the contents of diverse sources without losing semantics.

R8 (scalable performance): Finally, and most importantly, a data integration system should provide high overall scalable performance in terms of both the number of nodes and data size.

While requirements R1 - R7 are related to the high-level functionality and architecture (visible to its users) of a data integration system, the last requirement R8) is related to the internal implementation of such a system.

To fulfill requirements R1 - R7 we propose a distributed mediator architecture based on the peer-to-peer (P2P) paradigm. The architecture is described in detail in Sect. 3. As proposed in [55], here mediators are relatively simple software modules that encode domain-specific knowledge about data and share abstractions of that data with other mediators or applications. Each mediator is a database system with its own storage manager, query processor and multi-mediator query language which can reference database objects in other
mediators. More complex mediators are defined through these primitive mediators by logically composing new mediators in terms of other mediators and data sources. Logical composability is realized through multi-mediator views defined in terms of views and other database objects in other mediators and data sources.

Many architectures are possible that fulfill the general requirements in one or another degree. It is hardly possible to show that one architecture is superior to another from the users’ perspective. Most likely in the future there will be many P2P data integration systems that will differ in various aspects of their architecture. Only time and active usage in real-life problems can tell what is the right combination of features for a usable system. However, we believe that no matter what is the exact architecture, any such system will have to deal with the same fundamental problems with respect to the translation of logical mediator compositions into executable plans, that is query processing, which is the focus of this dissertation. No matter what is the particular architecture one of the most important issues for its usefulness is that of scalable performance. That is why our main goal is not the design of a complete architecture for mediation, but instead is the investigation of query processing techniques that are generic for such systems. The presented architecture provides the framework for the design and implementation of query processing techniques that will provide scalable performance and make the architecture useful in practice.

Therefore, at high-level, the research question we will address in this dissertation is: given an architecture that fulfills the general requirements R1 - R7, is it possible to design query processing techniques that will achieve high overall scalable performance in that architecture? This is a very general question that can be decomposed into many related sub-problems each having different answers depending on the particular architecture chosen and the requirements we put on a mediator system.

Our thesis is that logical composability in a P2P mediation architecture is an important requirement and that composable mediators can be implemented efficiently through query processing techniques.

In the rest of the dissertation we i) present a specific mediation architecture based on the P2P paradigm, ii) describe composability as the main requirement for the components in this architecture, iii) analyze several important problems related to processing queries in such an architecture, and iv) describe and evaluate experimentally the corresponding solutions which show that, indeed, it is possible to realize composability in a P2P mediator system with low overhead. The results are verified experimentally through an implementation of composable peer mediators in the AMOS II mediator database system.
Background

The title of this dissertation combines three independent concepts: *mediators*, *peer-to-peer systems* and *query processing*, that provide the foundation for our work. All three concepts have been extensively (re)defined and used in the literature in various senses. To provide a basis for the rest of our discussion, in this section we provide definitions of these concepts. As with most high-level architectural concepts, our definitions are necessarily informal.

To provide better understanding, we position the three concepts in a wider context. That is why first we discuss the area of data integration and the main approaches to implement data integration systems. Next we focus on the mediator approach to data integration as it is the basis for our work. Then we discuss peer-to-peer systems. Finally, we turn our attention to the area of query processing. Along with our main exposition we introduce several more related concepts that are used in the rest of our work.

2.1 Data Integration

The area of data integration is concerned with the problem of combining data residing in different autonomous sources and providing users with unified and possibly enriched views\(^1\) of these sources, and the means to specify information requests that correlate data from many such sources. A *data integration system* provides the means to define such integrated views and to process information requests against these views. The purpose of data integration systems is to hide the complexity of many diverse sources and present to the users a single interface to the data in all sources. As illustrated with the cloud on Fig. 2.1, there is no specific architecture for data integration systems, nor is there one standard technology to implement such systems. However, for reasons we will describe below, the most common research approach is to use techniques from the database and knowledge management areas. General concepts and architectures related to data integration from the perspective of the database systems area can be found in [44], [53]. A recent overview of the theoretical aspects of data integration from a formal logical perspective can be found in [31].

\(^1\)Here we use the term *view* in a general sense as the logical organization of the data the user sees.
Data integration has long been important for decision support in large enterprises because of the benefits it can bring due to improved decision making. However, recently many more areas of human activity rely on information technology to create, store and search information, such as engineering, health care, scientific research, libraries, and personal uses. These application domains lead to several important characteristics of the data integration problem:

- The information needs of the users of an integrated system can be diverse and dynamic, and cannot be predicted in advance. For example a genetics researcher or a mechanical engineer would hardly know in advance the kind of information and the sources they need to access in order to solve some problem. This requires that data integration systems provide flexible means for the specification of information requests.
- Typically, the sources cannot be changed and may not be even aware of their participation in a data integration system. To take into account and integrate existing sources, data integration requires a bottom-up design approach that starts from the sources and incrementally constructs a unified view in terms of the sources’ data.
- So far, the most common use of data integration systems is for information requests. There are several reasons for that. Typically, data integration is needed for decision-making which necessarily begins with request(s) for information and may (or may not) result in need for changes in the initial data. Many sources, such as most Web sources, provide read-only access. Finally, propagating updates to autonomous sources poses many hard problems related to their consistency.
2.1.1 Data sources

Since data sources are important in data integration, let us first look at what they are and what their properties are. Data sources are uniquely identifiable (in some scope) collection(s) of stored or computed data, called data sets\(^2\), for which there exist programmatic access, and for which it is possible to retrieve or infer a description of the structure of this data, called schema and possibly additional information about the source. All the information about the contents of a source (its schema, data size, etc.), the computational capabilities of a source (the interface to access the data), and possibly other information about a source as reliability, information quality, etc., are collectively called source meta-data. Data sources may contain very large or even infinite amounts of data such as data streams from sensors or financial data, or results from computer simulations.

A data source can be anything from a file that is accessible via the file system API of an operating system, a Web page accessible through a Web server via the HTTP protocol, a CAD simulation accessible through a CORBA interface, to a complex database managed by an RDBMS accessible through an ODBC driver. From our definition it follows that in general a data source cannot be identified neither with one single software component, nor with a single storage element. Therefore a data source is defined by the combination of a software component and the data (stored or computed) that it provides access to. Given the practically unlimited number of ways to combine various technologies to access data, describe and store data, the concept of a data source is a loose term and in some cases it can be hard to decide precisely what constitutes one data source.

An important aspect of data sources is that there is no single generic method to retrieve data source schemas, and to associate a schema with a source. Some sources such as RDBMS may store and provide the source schema as part of the data source itself but separately from the actual data. In other cases, such as XML and RDF documents, the data sets in a source (in this case called documents) may be self-descriptive and schema information may be embedded inside the data sets. Finally, some sources as Web pages may not provide any schema at all, but methods can be developed to analyze the data and extract its structure.

Data sources may share the same type of interface and/or system to access their data but differ in terms of their contents. Thus it is important to distinguish between types of data sources and data source instances. For example all Oracle DBMSs are the same type of data source, however each particular installation of the DBMS is a different instance of the Oracle DBMS. Due to

\(^2\)Here we use the term set in an informal sense. Formally speaking data sets can have either set or multi-set semantics.
the many possible combinations of common and different features of all potential data sources, it is not always possible to clearly separate between data source types and instances. For example a feature that is common only for a small group of data source instances may be considered as that group’s characteristic and used to distinguish this group of sources as a new kind of sources. On the other hand, such an approach may result in an unmanageable number of source types. As in other modeling problems, it is up to the designer of a data integration system to decide which sources constitute a type of their own.

From this discussion we can derive several important properties of data sources - heterogeneity, autonomy and distribution.

- **Heterogeneity.** Data sources may be heterogeneous at many levels. Based on [43] we distinguish three general levels of heterogeneity:
  - **Platform heterogeneity.** At this level sources differ in the operating system and hardware they use, physical representation of data, methods to invoke the functions that provide programmatic access to the source’s data, network protocols, etc.
  - **System heterogeneity.** At this level data sources differ mainly in two aspects. Data sources may use different sets of concepts, called *data models* to model real world entities. A variety of methods may be used for data access and manipulation. The collection of methods to access and manipulate data in a source is called *source capabilities*. Source capabilities may vary from a query language like SQL to a sequential file scan. Corresponding to our description of data sources, system heterogeneity is related to types of data sources.
  - **Information heterogeneity.** This level of heterogeneity relates to the data itself, that is to the data source instances. Their contents can differ at a logical level, because there exist many ways to model the real world. The resolution of this type of heterogeneity is called *schema integration*. Various taxonomies have been proposed to classify the differences between source instances at the logical level [58, 28, 43, 27, 22, 47]. Most works agree on two main types of information heterogeneity: semantic and structural heterogeneity. Different real-world concepts can be related to different concepts at the data source level which leads to semantic heterogeneity. Semantic heterogeneity manifests itself, for example, in different names for the same thing or the same name for different things, or using different units and precision. *Structural heterogeneity* (also called *schematic*) is related to the use of different concepts at the data model level, such as: different data types, objects vs. types, or types vs. attributes to model the same real-world entities.

- **Autonomy.** Because of organizational or technical reasons, data sources are usually independent and even not aware of each other. This independence is referred to as autonomy, which is related to the distribution of control (and
not data) [44]. In the organizational sense autonomy means that sources are controlled by independent persons or groups. In its technical sense autonomy is related to distribution of control [45]. Various overlapping definitions of autonomy are given in the literature that reflect its different aspects. In [41] node autonomy is classified in several types: *naming autonomy* relates to how nodes can create, select and register names of system objects, *foreign request autonomy* reflects the freedom a node has if and how to serve external requests and with what priority, *transaction autonomy* describes the ability of a node to choose transaction types and to choose when and how to execute transactions. In addition [41] recognizes heterogeneity as a type of autonomy - that is the autonomy in the choice of data model, schema, interfaces, etc. In [11] autonomy is defined as *design autonomy* - the freedom to choose data model and transaction management algorithms, *communication autonomy* - the ability to make independent decisions on what information to provide to external systems and when, and *execution autonomy* - any system can execute local transactions in any way it chooses. Another important facet of autonomy is the independent lifetimes of data sources, called *lifetime autonomy*.

- **Distribution.** Typically data sources reside on different computer nodes and thus are naturally distributed. As in [44] we use the term distribution with respect to data. However data sources may not only store but may also compute data. Thus, the distribution aspect of data sources is related to both data and function distribution, rather than just distribution of stored data.

Thus, data integration has to solve a wide variety of problems ranging from access to the data, unification of the data at various levels of abstraction, extraction of meta-data, and correlation of data items from disparate sources, to name a few. Naturally, all these operations have to be performed within reasonable time and resource limits, and therefore a major issue for any data integration solution is performance and scalability both in the data size and number of sources.

### 2.1.2 General approaches to data integration

Computer networks and network protocols allow to bridge the distribution gap between many data sources. However, networks only allow to bring data together and possibly unify it at the lowest physical level of representation (such as byte order). Thus we consider networks as an enabler for other technologies that can solve the problems brought by the heterogeneity, autonomy of data sources and the performance requirements for their integration.
Standards.

Standards are only a partial solution for heterogeneity. They can be applied only in well-defined domains where consensus can be reached about data representation and programming interfaces to data. It is hardly possible to foresee and standardize all possible ways in which data sources may be combined, thus even in a single domain, if standards are achieved, there are many aspects that cannot be fully standardized, for example the way people understand and model the world. Also standards often evolve and even compete, thus there is often the need to align different standards and to update systems with support for new standards which may be very costly and difficult. That is why even if standards can be enforced, there still will be heterogeneity of data sources at many levels.

Middleware.

One possible solution to the data integration problem is to migrate all disparate systems to one homogeneous, possibly distributed system. This is hardly a viable alternative as it may require all software at the data sources and all their applications to be rewritten and all data source owners to reach consensus about data representation and system interfaces.

Because of these mainly organizational reasons, data integration problems require solutions that do not interfere with the data sources and do not require changes of the data sources. To address this requirement, many data integration solutions introduce a unifying software layer called middleware [5]. Middleware is a very broad term used for a very wide spectrum of software systems and technologies. The goal of middleware technologies is to provide a degree of abstraction that hides various aspects of system heterogeneity and distribution. While many middleware technologies are not designed specifically to solve data integration problems, they can be applied for data integration either directly or as parts of more complex solutions.

Distributed object technologies.

One type of widely used middleware are distributed object frameworks such as CORBA [45], DCOM [33], Java RMI, and Web services [50]. All distributed object technologies have several features in common. They provide a general purpose way to specify procedural interfaces to some computation services and transparent access to remote objects. These technologies are concerned with the ability of distributed heterogeneous systems to transparently invoke each other’s services and exchange data (often in the form of objects) across heterogeneous platforms and languages with different type systems. However, distributed object technologies are not concerned with how to efficiently compose distributed services and leave this task to the programmer. Since distributed object technologies are based on general-purpose procedural lan-
guages (typically object-oriented), their direct application to data integration has the following problems: i) they do not provide high-level constructs for the integration of many data sources and require “manual” programming to encode the transformation and combination of data from many sources, ii) every time when new information need arises or a new data source has to be added the middle object layer has to be changed, which may require a lot of (re)programming, iii) they do not expose the implementation of the services which prevents global optimization of composed services (e.g. a Web service that uses other Web services), and iv) it is infeasible to perform such global optimizations of composite services even if their implementation is available. This makes the direct application of general purpose distributed object management technologies unsuitable for the integration of many data sources especially in cases when the sources contain large amounts of dynamic data, and changing user information needs. Thus, distributed objects are enabling technologies on top of which more advanced solutions can be built.

Database technology.

A natural choice of technology for data integration are database management systems (DBMS) [17]. Database technology presents a high level of abstraction of large data sets and the operations to manage and query such data sets through declarative interfaces. Query languages and standardized data models allow the implementation of scalable and flexible systems that can manage and access very large data sets with very little programming effort compared to procedural frameworks.

However, database technology has been developed to manage homogeneous data sets (using the same physical and logical organization) that are fully controlled by a DBMS and therefore are not autonomous. For reliability and performance reasons DBMS technology has been extended to manage distributed data. Still, distributed DBMSs (DDBMS) are homogeneous systems that consist of the same type of nodes that operate as one system, and therefore neither the nodes of a DDBMS nor the data it manages are autonomous.

In order to be applicable to data integration problems, database technology has been extended and modified in various ways to support heterogeneity and autonomy. An exhaustive discussion of the architectural alternatives for database systems depending on the degree of autonomy, distribution and heterogeneity is given in [44]. Reference [9] provides an overview and classification of approaches to querying heterogeneous data sources along several other architectural dimensions. Here we overview in the following two sections the two most popular approaches to data integration middleware based on database technology - data warehouse and mediator systems.
2.2 Data Warehouses.

One possibility to integrate data from many sources is to extract data of interest from the sources, transform that data into a uniform representation and then load it into a central repository, a data warehouse, that provides uniform access to the integrated data. This approach is often called materialized because it physically materializes the integrated view by copying transformed data from the sources. Due to the maturity and wide use of relational database technology, it has been the primary choice to implement data warehouse systems. Data warehouses are built as a subject-oriented databases that are specialized in answering specific decision-support queries. This approach allows for avoiding the replication of all data from all sources which often may be infeasible or even impossible, and allows for the fine-tuning of a database for complex ad-hoc decision-support queries. A simplified architecture of a data warehouse is shown in Fig 2.2.

Figure 2.2: Simplified data warehouse architecture

A data warehouse integrated schema is first designed that logically integrates the data sources. The most common type of data sources are operational databases, that is, relational DBMS used for the day-to-day operation of an enterprise, tuned for on-line transaction processing (OLTP). Other types of data sources can be used as well, such as Web pages, specialized biological and engineering databases. To populate a data warehouse, the data is first extracted from multiple data sources. Then the data has to be cleaned, that is, anomalies such as missing and incorrect values are resolved, and transformed into uniform format. After extraction and cleaning the data is loaded into the warehouse. During loading data can be further processed by checking integrity constraints, sorting, summarization and aggregation. Thus data loading materializes the integrated views defined during the design phase of a data warehouse. To support decision-making data warehouses are designed to store historical data that is, organized in predefined dimensions that correspond to subjects of interest. Periodically the data warehouse is refreshed by propagating changes in the sources to the warehouse database. The process of loading and/or updating a data warehouse often may take many hours or even
days. That is why data warehouses are refreshed from time to time (once a
day, or even once a week) and the users do not have access to the most recent
data. Since a data warehouse has to accommodate all data of interest from
the sources for long periods of time, its design requires very careful planning
in advance both of its logical and physical organization, which can be a very
time-consuming and complex process. A detailed overview of data warehouse
technology can be found in [8].

2.3 Mediator Database Systems

An alternative to the data warehouse approach is to keep all data at the sources
and access the sources on per-need basis to retrieve and combine only the
data that is relevant to a request. For that, an intermediate software layer is
introduced that presents to the users a logically integrated view of the data
sources. Since this integrated view is not materialized explicitly by the user,
this approach to data integration is often called virtual.

The requirements for the functionality, interfaces, and architecture of a vir-
tual integration layer are analyzed in [55], and based on this analysis an ar-
chitecture for a mediation layer is specified, illustrated on Fig. 2.3. A medi-
atver layer is a virtual middle layer that separates the functions related to data
integration from the data management functions of the data sources and the
presentation functions of the applications. The goal of this layer is to simplify,
abstract, reduce, merge, and explain data. It consists of mediator modules,
deﬁned in [55] as “a software module that exploits encoded knowledge about
certain sets or subsets of data to create information for a higher layer of appli-
cations”.

![Figure 2.3: Mediation architecture](image-url)
The mediation architecture is targeted at the integration of large number of autonomous and dynamic data sources that are typically available on the Internet or other wide-area networks. In this environment, maintainability is of uttermost importance. For better maintainability, a mediation layer is designed in a modular way and consists of a network of small and simple mediator modules specialized in some domain. Thus every mediator can be maintained by one domain expert or a small group of experts. Mediators share their abstractions with higher levels of mediators and applications which can use the domain knowledge encoded in lower-level mediators. Applications and mediators that require information from different domains use one or more other specialized mediators. Each mediator presents its own integrated view of some sources and mediators and thus adds more knowledge to the mediator network. An important consequence is that there is no single global view of all sources. There may be a large number of mediators to choose from. To facilitate knowledge reuse and discovery, mediators should be inspectable and provide data about themselves. A logical application of mediators is to use some of them as meta-mediators that facilitate the access to mediator and data source metadata. According to [56] the main tasks of a mediation layer, called mediation services, are:

- accessing and retrieving relevant data from multiple data sources,
- abstraction and transformation of the retrieved data into a common representation and semantics,
- integration and matching of the homogenized data,
- reduction of the integrated data by abstraction

Since mediators do not store the source data themselves, all functions related to data access, integration and delivery have to be performed dynamically “on-the-fly”.

The concept of a mediator does not prescribe a particular implementation technology. However, as indicated in [55], a declarative approach to mediator design can bring the necessary maintainability and flexibility required for the integration of large number of dynamic sources. In particular mediators should support declarative interfaces to the applications and other mediators. Most practical implementations of mediator systems are based on database technology. For such mediator systems we use the term mediator database systems (MDS). Below we will focus on mediator database systems and will use common database terminology to describe the structure and operation of mediator database systems.

Data integration in an MDS is performed in two main stages. The first stage, data model mapping, specifies how to retrieve data from each of the sources and how to convert the source data to the data model of the mediator system. This step deals with system heterogeneity, and provides a uniform representation of all data sources in terms of the mediator data model, called
the common data model (CDM). The second stage, schema integration, deals with the information heterogeneity of sources’ data on a logical level. During this stage identical objects in different sources are matched and schema and data instance conflicts are resolved. Since all sources’ data is mapped to the mediator CDM, at this stage the CDM and the mediator query language can be used to define database views that logically unify the data sources.

Thus the data model and the query language of the mediator serve as the single interface to all integrated sources. User’s information requests are then expressed in terms of the mediator query language. The actual retrieval and transformation of data from the sources is typically performed on demand when users pose queries to the integrated schema of the MDS. Other modes of data delivery are possible such as publish/subscribe, push, and broadcast [36].

These two integration stages often require very different approaches. As pointed out in Sect. 2.1 the data sources may present extremely diverse interfaces to their data and use very different data representations. This often requires a Turing-complete programming language to be used to specify the access to the sources and the required low-level data transformations. On the other hand, once the data has been transformed into a CDM and can be manipulated by a query language, semantic transformations can be specified declaratively.

Based on this two-phase integration approach, mediator systems are usually organized into two architectural tiers, each responsible for some of the tasks specific for the mediation layer. The first tier is typically responsible for the data model mapping phase. It is usually implemented as software components, called wrappers, that implement a uniform programming interface which hides all access details to the sources. Typical wrapper functions are retrieval of source data and its translation into the mediator CDM, access to (or inference of) source meta-data and statistics. The second tier, usually called the mediator tier, provides conflict resolution primitives across multiple sources. These primitives can be expressed in the query language of the mediator system, because the data from all sources is translated in the data model of the mediator by the wrappers. This two-tiered architecture is often referred to as the mediator-wrapper approach.

Notice, the term “mediator” was used in two senses - denoting the general mediator concept as presented in [55], and denoting only the mediator tier of an MDS. In addition projects such as TSIMMIS [13] and AURORA [58] use the term mediator in the sense of the integration views defined in a mediator, while they use correspondingly the term mediator template or mediator skeleton to denote the mediator system itself. Other works do not specify the exact meaning of the term “mediator” and often use it in all three senses. We provide a precise definition of the mediator concept as we use it in this work in Sect. 3.4.
At the semantic level of data integration there are two distinguished approaches to logically specify the relationship between a mediated schema and the schemas of the data sources. In the first approach the integrated (also called “global”) schema is described as views in terms of the local schemata of the sources. This is the approach known as global-as-view (GAV). As opposed to GAV, the second approach first defines a global integrated schema. Then the contents of the sources is defined as views over this global schema. This approach is known as local-as-view (LAV), since the source schemata is expressed as views in terms of the global view. An overview and comparison of the two approaches can be found in [32, 52].

Very few systems fully implement the general mediator architecture described here. Most such systems are either centralized or have a fixed 2- or 3-tier architecture. Furthermore, most such systems provide read-only access to the data sources.

One of the advantages of using database technology as a basis for the implementation of mediator systems is that much of the research and practice in the database area can be reused. Since both the integrated views and the user information requests are expressed in terms of a query language, the area most important to mediation is query processing. We discuss the general and the mediation specific concepts related to query processing in Sect. 2.5.

2.4 Peer-to-peer Systems

According to the Oxford English Dictionary the primary meanings of the word peer are “1. An equal in civil standing or rank; one’s equal before the law. 2. One who takes rank with another in point of natural gifts or other qualifications; an equal in any respect”. The concept of peer-to-peer (P2P) is a general software architecture paradigm at the same level of abstraction as client-server computing. Systems with P2P architecture consist of software components, called peers, that share and use each other’s resources to perform a common task. The shared resources can be computing power, storage space, bandwidth, and even human presence. Two recent overviews of the general aspects of P2P and of the most popular P2P systems can be found in [3, 40].

Due to its general nature, the concept of P2P systems has been understood and defined in various ways. Here we provide several recent definitions. The Intel P2P Working Group defines P2P computing as “the sharing of computer resources and services, including the exchange of information, processing cycles, cache storage, and disk storage for files, by direct exchange between systems. P2P computing approach offers various advantages: (1) it takes advantage of existing desktop computing power and networking connectivity, (2)
computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network, and (3) it can reduce the need for IT organizations to grow parts of its infrastructure in order to support certain services, such as backup storage.” According to [49], “P2P is a class of applications that takes advantage of resources - storage, cycles, content, human presence - available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers”.

P2P systems are based on three fundamental principles [3]:

- **Resource sharing** requires that peers (some or all) share some of their resources with other peers.
- **Decentralization** means that a system consisting of many peers is not controlled centrally.
- **Self-organization** is required in view of decentralization so that autonomous peers can coordinate to perform global activities based on local shared resources.

Initially the term P2P has been used for distributed file sharing and simple keyword search, made popular by the Napster system. While P2P is often considered equivalent to distributed file sharing applications used in systems such as Gnutella, Kazaa.

However many other systems targeted at different application areas fall into the P2P category. Distributed computing systems as SETI@home, Entropia use P2P technology to share processing power resources. Such systems are useful for complex computational tasks that can be split into smaller ones and then distributed among available peers. Another application area is collaboration. Such systems allow users to collaborate, often in real time to perform a common task without relying on a central infrastructure. Popular applications are Jabber for messaging, Groove for combined messaging and document sharing, project management, etc. Another type of systems are P2P platforms such as JXTA and FastTrack that provide generic APIs to build P2P systems.

Technically, two general types of P2P architectures are distinguished: **pure** P2P systems do not have any centralized server or repository of any kind and all nodes are equal, while **hybrid** P2P systems employ one or more central

---

4 [www.napster.com](http://www.napster.com)
5 [setiathome.ssl.berkeley.edu](http://setiathome.ssl.berkeley.edu)
6 [www.entropia.com](http://www.entropia.com)
7 [www.jabber.org](http://www.jabber.org)
8 [www.groove.net](http://www.groove.net)
9 [www.jxta.org](http://www.jxta.org)
10 [www.fasttrack.nu](http://www.fasttrack.nu)
servers, e.g. to obtain meta-data such as network addresses of peers, and/or have some nodes with special functionality. *Super-peer* architectures [59] are a kind of hybrid architectures with hierarchical organization, where groups of peers communicate with all other peers through super-peers.

### 2.5 Query Processing and Optimization

The main purpose of a mediator system is to retrieve, combine and enrich existing data through queries in a declarative language. Therefore one of the most important functionalities of a mediator is the ability to efficiently process queries. *Query processing* is a collective term that stands for all techniques used to compute the result of a query expressed in a declarative language. Usually query processing is performed in two distinct steps. *Query optimization* transforms declarative queries into an efficient executable representation called *query evaluation plan* or *query execution plan* (*QEP*). *Query evaluation* takes a QEP and interprets it against a database to produce query results.

![Simplified DBMS query processor](image)

*Figure 2.4: Simplified DBMS query processor*

A simplified diagram of a DBMS query compiler is shown on Fig. 2.4. The *parser* checks input queries for syntactic correctness and translates them into an in-memory representation called *parse tree*. The parse tree is analyzed for semantic correctness by the *preprocessor*. The semantic analysis includes
checks such as whether relation and attribute names actually correspond to existing relations with corresponding attributes, whether all attributes and constants are type-compatible with their usage, etc. Semantically correct parse trees are translated into an internal representation. The actual compilation of the query begins with this internal representation.

In most modern database compilers [53, 44] query optimization is performed in two main stages each using a different internal representation of the query. The first, called \textit{query rewriting}, is based on equivalent logical transformations of some kind of a calculus form of the query. A calculus is a non-procedural representation of the query where the desired result is expressed via a logical formula equivalent to some variant of predicate calculus. This phase is performed by the \textit{rewriter}. The goal of the calculus-based rewrites is to simplify the query and to transform it into some normalized form suitable for subsequent optimization.

The next compilation phase, called \textit{query optimization}, accepts a calculus query representation and transforms it into an equivalent algebraic form. This phase is performed by the \textit{optimizer} which applies algebraic laws to produce a more efficient algebraic representation. An algebra is a formal structure consisting of sets and operations on the elements of those sets. For example relational algebra is a formal system for manipulating relations. The operands of relational algebra are relations. Its operations include the usual set operations (since relations are sets of tuples), and special operations defined for relations: selection, projection and join. Since algebraic operators have precedence and order of application of the operators, a query algebra is procedural in the sense that it prescribes how to construct the result of a query. Abstract algebraic operations may be implemented by various algorithms, each with different execution cost. To produce an optimal QEP, the query optimization phase usually searches the space of all logically equivalent algebraic expressions that compute a query, assigns to the logical operators all applicable algorithms and computes the cost of executing all the operators in the plan according to their order and chosen implementation. An algebraic representation of a query where the operators are associated with evaluation algorithms and cost functions for those algorithms is called \textit{physical algebra}. In order to choose the best possible QEP the optimizer uses a cost model to evaluate the quality of each candidate plan. For this various measures can be used such as resource consumption or total execution time.

An optimal physical algebra expression where all algebraic operations are assigned an implementing algorithm can serve directly as a QEP of a query, and can be directly interpreted. Optionally there may be a final phase, performed by the \textit{code generator}, that transforms the algebraic expression into some lower-level representation, e.g. CPU instructions.

Finally the resulting QEP can be executed by the \textit{execution engine} or may
be stored for future use.

Query processing for mediator systems introduces several more phases and components shown on Fig. 2.5. In the figure the additional components are denoted with continuous lines. A decomposer component identifies which portion of the original query can be computed by each of the sources. Decomposition is performed either by the rewriter or by the optimizer or both, depending on the particular mediator architecture. Decomposition is also present in query processors for distributed DBMS [44]. However, there the query processor needs not take into account the heterogeneous computing capabilities of the data sources. As a result of decomposition the original plan is split into sub-queries, each executable by one particular data source. The subqueries are then submitted to the corresponding wrappers, which in turn translate them into requests specific for the type of source accessed. The source return results in their format which are mapped to the mediator CDM by the wrappers and assembled by the execution engine as final results.
A P2P Architecture for Mediation

In this section we present a software architecture for the integration of many distributed and autonomous data sources that fulfills the need for scalable data integration in a distributed and autonomous environment. As in [14] by software architecture we mean “The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.” Software architectures provide a high-level view of software systems that allows to define and reason about their general properties. We define our architecture in a top-down fashion. To fulfill requirements $R1$-$R7$ defined in Sect. 1 we base our architecture on two fundamental ideas - mediators and peer-to-peer systems. These two design choices are justified in Sect. 3.1. For system architectures based on these two ideas we use the term peer mediator system (PMS) architecture. Next we identify some important requirements that stem from these design choices in Sect. 3.2. Then we identify and describe the components of a PMS and how they form together a PMS. Finally we discuss the interactions between the components that meet the architecture specific requirements in Sect. 3.2. The presented PMS architecture is partially implemented in the framework of the AMOS II mediator system and is described in Paper B.

3.1 Design Motivation

Mediators.

We base our architecture on the mediator approach because the materialized (data warehouse) approach is not suitable for our target environment for several reasons. With a large number of sources it may not always be practical or possible to materialize all data from all sources due to its volume or source limitations. On the other hand if only a subset of the data in the sources is to be materialized in a data warehouse, it may not be possible to know in advance what queries will be issued in the future and thus what data to materialize. With autonomous data sources it may not be always possible to access all their data due to security or request size restrictions. For sources that compute their data dynamically instead of storing it explicitly (such as simulations), the source contents may be infinite and therefore impossible to pre-compute and store. Even more, for sources that perform specialized computations it may not be
possible to pre-compute and store all possible results. Such sources require that results are computed on the fly depending on their input. Finally due to potentially very large amounts of data in the data sources, it may not be possible to update a data warehouse in a timely fashion, and to keep up data warehouse updates with the change rate of the sources, which would result in materialized views that are gradually getting older than the source data.

The mediator approach presents a solution to these problems. Since mediators are virtual databases that provide only a logically unified view of many data sources, all data access and transformations are performed on the fly during query execution. This allows to:

- avoid the storage and update overhead of the materialized approach,
- answer any query over a virtual database with the most current data from all sources without prior knowledge of a query workload,
- request only small portions of the sources’ data that relevant to a query,
- optimize query execution plans so that the least possible amount of data is retrieved from the sources and transferred over the network,
- reuse the computational capabilities of the sources, such as feature extraction or pattern matching operations, and thus compute wider classes of queries,
- take into account source security and access limitations.

In addition, mediators may transparently materialize and cache intermediate results and some source data for improved performance.

Peer-to-peer architecture.

The mediator approach does not prescribe any particular architecture apart from that there is a distinct middle layer between the user applications and the data sources that separates the application logic from the data management logic at the data source layer.

The simplest possible approach to design a mediator layer is to encapsulate all mediation functionality in a single centralized mediator system. This is the approach typically used in most existing mediator systems. However, with a centralized approach only few of the general requirements $R1$-$R7$ can be fulfilled. The main reason for that is the assumption that all integration can be performed by a single entity (individual or organization), which is not true in our target environment. On the contrary, when large numbers of distributed data sources are available, it will be most likely that many independent domain experts will have the knowledge of how to integrate only portions of the data in a subset of all sources. Therefore in a centralized mediation architecture:

- only the autonomy of the data source owners can be preserved, but not that of the domain experts that integrate the sources, who will have to reveal all their specialized knowledge to the mediator owner,
- data integration will require the coordinated access to and modification of a
one global virtual database schema by many independent entities, a process that may not scale over many sources and many participants,

• central administration is required and therefore coordination between many independent parties, which may not always be possible,

• it may be expected that a single mediator that covers many knowledge domains and accesses large number of sources of many different types will be extremely complex and hard to maintain,

• finally, a centralized mediator presents a single point of failure.

In summary a centralized mediator architecture requires central management and concentration of all knowledge required to integrate all sources in one place, something that is hardly possible on a large scale across geographic and organizational boundaries.

However, as envisioned originally in [55], a mediation system may in general consist of many distributed mediators specialized in some knowledge domain where each mediator integrates only a small subset of all available sources and shares its data abstractions with a higher level of mediators and applications, a vision that naturally maps into a P2P architecture. Autonomy and decentralization are inherent properties of the P2P paradigm for distributed computing, therefore one natural way to design a distributed mediation system that fulfills the organizational requirements for autonomy (R1) and decentralization (R2) is to design this system as a peer mediator system (PMS) consisting of autonomous mediator peers that interact with each other and with the data sources and user applications. Next we discuss some important requirements for a PMS that follow from the design choice of peer-to-peer architecture.

3.2 Requirements

Before describing our architecture for peer mediators, we first discuss the important requirements that peers participating in a PMS should meet. We divide these requirements into two groups. First are the ones that we address through the contributions presented in this dissertation. For completeness, we present a non-exhaustive list of additional requirements that are important for a successful implementation of a PMS, but are outside of the scope of this work. We consider the next three requirements to be fundamental for the realization of a PMS architecture which is why we chose to focus our work at their study and fulfillment.

Logical composability.

The main value of a PMS is in its ability to not only distribute the integration effort among many autonomous participants, but in that it provides the means to assemble integrated views of both data sources and other integrated views
and thus reuse human efforts and knowledge encoded in the mediators.

Two main approaches exist to realize compositions of distributed software components. One is through distributed technologies such as RPC, CORBA, or Web services. These approaches are procedural, require a lot of programming effort, are rather static, and result in more or less tightly coupled distributed systems that are hard to evolve. Therefore we do not consider these approaches to be directly suitable for dynamic systems such as PMSs. We term distributed systems that can interoperate through such procedural approaches as \textit{physically composable}.

A much more flexible and scalable approach is to specify mediator compositions logically in terms of a declarative language. This requires that the peers in a PMS \textit{i) have} a query language and a view definition mechanism that provides constructs to refer to both views and stored data in other mediators, that is define and access data in \textit{global views (queries) and ii) are} able to share their views and stored data with other peers, that is define some schema objects as \textit{public} and provide the means to access them. Having these two properties allows to transitively define arbitrary logical compositions of peers in terms of each other, a property we name \textit{logical composability}.

Logical composability extends the concept of logical data independence in traditional databases across many distributed peers and allows peers to evolve without affecting each other as long as the view interfaces are kept intact. Another advantage of logical composability is that mediators can reuse indirectly abstractions exported by other peers without even knowing their existence which promotes reuse and autonomy.

\textit{Physical composability.}

To realize logical composability it is necessary that peers are able to generate executable plans to compute the extensions of many transitively composed global views. That is, peers must be able to translate logical view compositions into physically composed access plans across many mediators and sources. In order for such plans to be executed peers must support programmatic interfaces to communicate over a network. These programmatic interfaces can be implemented via one or more of many available technologies for distributed interoperability [33], such as RPC, CORBA [45], DCOM, and more recently SOAP [2].

\textit{Location transparency.}

Large number of computer nodes, typically used as “dumb” Internet clients, connect to the Internet via temporary connections and identify themselves through dynamic physical (IP) addresses (such as computers connected over a modem, LANs with DHCP, subnetworks behind NAT) that may change over time. Many of these nodes may host mediator peers managed by the node
owner(s) and possibly used by other such nodes. Due to the mobility of many computing devices, peer owners may migrate their peers from one node to another (e.g., when a peer has been moved from an office workstation to a portable node). To support such scenarios, peers should not be bound to physical addresses or to physical nodes. This requires that peers are somehow uniquely logically identified within a PMS in a way that allows to dynamically map logical peer identifiers to physical locations.

Logical identification of peers allows both users and peers to abstract from the physical network details. In order to be able to refer to remote peers by their logical identifiers, peers have to be able to perform name resolution, that is, map logical identifiers to physical addresses. For a PMS to scale in number of peers and users, name resolution must be performed in a fully automated and transparent manner that scales over large numbers of peers.

Requirements outside of the dissertation scope.

An implementation of the PMS architecture that would be useful in practice raises a number of additional problems that will not be addressed by this work. Below we discuss some of these problems that we consider to be important for a successful implementation of a PMS.

Information discovery:
The task of identifying relevant sources of information is information discovery. These sources can be both other mediators that provide already existing abstractions of data sources and other mediators, or directly data sources. The result of information discovery consists of logical identifiers of peers and optionally additional meta-data about peer contents such as relation names and attributes, file names, functions, etc. Information discovery requires that mediator peers are able to store, exchange and query meta-data about other mediators and data sources, a feature described as inspectable mediators in Sect. 2. In a P2P architecture, information discovery poses additional performance problems since there is no central meta-data repository and thus large number of global meta-data requests may need to be processed. A related problem is that of bootstrapping a PMS with initial meta-data so that a set of disconnected peers can “learn” about each other and form a PMS together.

Schema integration: One of the most important problems in data integration in general is how to describe mappings between an integrated schema and the sources’ schemas. In a PMS this problem is exacerbated by the potentially very large number of views distributed among autonomous mediators. Thus, a PMS requires information modeling concepts at the query language level that will provide the users with scalable tools to
easily integrate large number of sources. In addition tools and methods are necessary to perform schema integration in an (semi-)automated way.

**Dynamic availability:** Due to their autonomy, the peers in a P2P system may control their own availability independent of other peers. At the same time, on a global network some peers may become unreachable due to network problems or simply because the nodes they reside on were disconnected from the network. That is why peers should be able to join and leave a PMS at any time without disrupting the overall operation of the system. This requires a mechanism for the peers to detect each others’ availability and gracefully react when some peers are not available. The most challenging problem here is to define the semantics of integrated views when some of the views’ sources are unavailable and to process queries against such views in a way most suitable for the user.

**Security:** In a PMS system users may have conflicting interests and even malicious intentions. Thus, care should be taken in a PMS that users cannot access restricted information, tamper with information that travels through many peers, and disrupt the operation of the system as a whole. Two problems specific for a PMS architecture are, e.g.: i) a highly decentralized system catalog with security related information such as users, groups, passwords, keys and permissions may lead to performance problems, and ii) when integrated views are defined, it may happen that some global execution plans are non-executable due to local security restrictions which requires the query processor of a PMS to be able to take security restrictions into account.

### 3.3 External System Components

To describe our P2P mediation architecture we first analyze the types of software components that participate in a PMS. According to the conceptual mediation architecture described in Sect. 2.3, an integrated information system consists of three types of software components, each with specific purpose and functionality, divided into three layers: data sources, mediators and user applications. The data source components and their interfaces are given a priori. Many applications exist that derive their data from various information systems over standard interfaces. It is desirable that a data integration system provides the means to reuse such applications. Therefore the design of a data integration system can be viewed as a two step process. Existing sources and applications require a bottom-up step that puts some requirements on the mediation layer. After these requirements are stated, the design of the mediation layer itself can
be done in a top-down fashion. That is why we first observe and analyze the
main properties of the components in the data source and the application lay-
ers which are external from the view point of the mediators. Then in Sect. 3.4
we define the internal architecture of the mediation layer so that it fits best our
observations and requirements.

3.3.1 Data sources.

Section 2.1 defines a data source as a uniquely identifiable couple of a software
component and its data where a method exists to acquire some source meta-
data that contains at least the source schema and possibly other information
about the source. In this section we investigate in more details the properties
important for data integration of the data source components.

Low-level interfaces.

Data sources provide access primitives that allow external components to in-
voke some computation at the data sources, and to send and receive data. The
collection of all access primitives of a data source comprises its low-level inter-
face. We distinguish two kinds of such interfaces. Global data sources support
network-based interface(s) and are globally identifiable and globally accessi-
ble by remote systems over a network. Examples of global sources are Web
sites, Internet search engines, Web services, LDAP and DNS servers, etc. Lo-
cal data sources do not have globally unique identifiers and there is no method
to access them by external components over a computer network. Typically
local interfaces are provided in the form of call-level APIs. Examples of local
sources are ODBC and JDBC sources, local files, and software components
accessible via an API (e.g. a B-tree index library). To make local data sources
globally accessible to all peers in a PMS, one or more mediator peers must
serve as intermediary between the local source and the rest of the PMS.

The large number and diversity of the low-level interfaces to existing and
future data sources, requires that a mediator system is easily extensible with
new functionality for the access to a variety of sources.

Computational capabilities.

A higher level of abstraction above low-level interfaces are the data sources’
capabilities which are related to, but often not equivalent to the low-level in-
terface(s) supported by the sources. In fact the same capabilities may be ac-
cessible via different low-level interfaces, e.g. for RDBMS typically these are
ODBC, JDBC, and a call-level API all providing access to the same functional-
ity. Thus capabilities are not equivalent to interfaces. By capabilities we mean
the abstract computations that a source can perform over some optional input
data. Based on similarities and differences in their supported capabilities the
data source components can be subdivided into four levels of abstraction.
• Type of source. This is the most general classification of data sources according to which all sources with the same set of capabilities are of the same kind. Some examples are all relational DBMSs that support the SQL'92 standard, or all installations of a particular DBMS like Oracle 9i or DB2 v7.2, or all installations of the Google search engine. All sources of these kinds have their own specific capabilities either by virtue of being instances of a particular software system or by fully implementing some standard. Typically such data source kinds will be defined by standards or by some well-known systems.

• Source instance. Many kinds of sources are customizable and extensible. Thus, particular source instances (typically represented by a system of some type being installed on a computer node) may differ in the functionality they provide. For example a relational database may contain special user-defined functions, created by its local administrator. Of course capabilities present in one or few source instances may gradually become adopted by a vendor, and then such group of capabilities may form a separate kind of sources.

• Schema instance. The above two classifications look at a source as a whole. It is possible that a source can perform certain computations over some of its data sets, but not over others. A typical example are Web forms where scans can be performed over some data sets (e.g. get all countries), other data sets may allow only selections (e.g. retrieve all cars of a specific make), while third ones may allow only joins (e.g. get all parts supplied by suppliers in Sweden). Thus, the capabilities of a source may change with respect to its current schema and are not inherent for the source instance. Such limitations may be due to only few queries being publicly accessible through a Web interface, or because the data access is hard-coded in some procedural language.

• Data instance. Finally at the lowest level of abstraction a source instance with particular schema may have varying capabilities depending on its current data contents. For example, if a Web form presents a choice of cities where users can look for housing, this page can be viewed as a source with two data sets - that of cities and of properties. However, the housing information that can be retrieved depends on the contents of the cities data set.

Given the wide variety of interfaces and capabilities of the data sources, one of the major problems for mediator systems is how to utilize existing capabilities over the available low-level interfaces, how to compensate for missing capabilities, and finally how to find sources with some specific set of capabilities, e.g. a matrix multiplication source or an image matching source. Solving this problem requires that mediators are able to represent in some way the capabilities of the sources they access. Ideally such a representation of capabilities should be easy to specify, query and manipulate both “manually” by humans.
and automatically by the mediators so that both new kinds of sources and new source instances can be easily added, existing ones modified and queried for their capabilities.

Relationships between data sources.

Data sources and/or the data items in the sources can be interrelated in a variety of ways, the most common of which we discuss below.

- **Data ↔ meta-data.** One possible way to acquire source meta-data is to retrieve it from another source. An example of such sources are XML files with external DTD or XSchema descriptions, and Web services described in UDDI registries. Thus data sources may be related by a data - meta-data relationship. This relationship may be “known” to some of the involved sources (e.g. as a URI in an XML document that points to its DTD), to third source(s) or mediators, or to humans. To facilitate source discovery and automated integration meta-data sources should be accessible in the same way as other sources. To allow for uniform treatment of data and meta-data at any level, we do not distinguish meta-data sources from data sources, but we require that a mediator system can model this relationship in terms of its CDM.

- **Data ↔ index-data.** Sources may also serve as indexes to other source’s data. One example are text document indexes that provide fast access to external documents either in a file system or on the Web. According to our definition of a data source, indexes can be considered as data sources of their own. In such case a relationship exists between the index source(s) and the data source(s) it indexes. For example the Google and AltaVista Internet search engines can be considered as indexes of most Web documents on the Internet. Knowledge of the relationship between index and data sources can be very important for the overall performance of a mediator system and can provide alternative more efficient access paths to external data. In the cases when a data source does not provide a “scan” interface, an index may be the only way to access the data in the source. Utilizing the index - data relationship is the only way to retrieve data from such limited sources.

- **Data ↔ nested data.** Certain data sources may have nested structure, that is, access and combine data from other data sub-sources. Due to the diversity of all possible types of sources it is very hard to automatically detect and model the structure of arbitrarily composed data sources. This may not be possible either because the sources do not contain information about their own structure or do not provide access to this information, or because of security and privacy restrictions. Therefore in most cases data sources can be considered to be atomic from the view point of an external system, that is their nested structure is “invisible” to a mediation system. However, such compound data sources may provide the means for external
systems to inspect their internal structure. Typically such sources would use a language to describe the composition of many sub-sources and would provide some way of retrieving definitions of source compositions. Examples of such sources are DBMS products with support for external sources in their data definition and query languages (e.g. the SQL/MED standard [39]). In other cases the source structure may be specified manually by a human. Either way a mediator system may benefit from the knowledge of the relationship between sources and sub-source(s) in two ways. If the sub-sources are directly accessible by an external system, then a mediator system may generate more efficient source access plans that bypass the container source and access the sub-sources directly. If the container source provides a language interface, then the mediator may generate more efficient requests in terms of the container source language, e.g. by combining multiple requests.

- **Inter-source semantic constraints.** The contents of data sources may be semantically related in various ways. A source may be a replica of another source, or there may be functional dependencies between sources. A mediator may utilize this knowledge to provide integrated views with richer semantics, to generate more efficient access plans to the sources and to generate integrated data with better quality.

### 3.3.2 Applications.

User applications send requests to the mediation layer on behalf of a user, and deal with the presentation of mediator replies to the user. By definition applications are not capable of processing requests by themselves.

Many applications or application development frameworks have been developed that provide advanced data analysis and visualization functionality, and support standard interfaces for data access. To utilize such legacy applications and frameworks a mediator system must be able to support some data access standards (such as ODBC/JDBC, EJB, etc.) and provide the means to be easily extensible with new interfaces.

Since these standard interfaces are not developed with any particular system in mind, they may not be suitable for future applications that would access integrated data through a mediation system. Standard interfaces suffer from several deficiencies: i) they already assume a predefined set of functionalities that may not be sufficient to express all capabilities of a mediator system, ii) they are based on data models that may not be expressive enough to translate all concepts at the mediator CDM, and iii) they may not provide the necessary level of performance. Therefore a mediator system should provide rich specialized interfaces for more effective and efficient access to the mediation layer by new applications. To support the needs of future applications, the
mediators should provide at least two types of specialized interfaces.

To allow arbitrary applications to access arbitrary mediators across the network in a flexible manner, the mediators may provide a low-level network interface directly based on some transport protocol as TCP/IP. Typically such interface would be implemented by advanced applications that support a mediator network protocol, need data processing functionality not present in the mediators and need to access more than one mediator. The advantages of a network interface are that it allows for loose coupling between the application(s) and the mediator(s) that is independent from programming languages, operating systems and hardware. However, such global applications require more intelligence built in them so that they can discover and communicate effectively and efficiently with many distributed mediators and combine the retrieved data. Thus, low-level application-to-mediator network interfaces would result in very complex applications that implement much of the functionality already present in the mediators.

In order to avoid such complex applications, all functions related to the retrieval and combination of data from many mediators can be delegated to a single, specially designed mediator that serves as the application’s gateway to all other mediators. This approach allows applications to stay relatively simple and delegate all tasks related to the efficient access to many remote mediators to the gateway mediator. For this, a high-level function call interface is needed to provide future and existing applications with the ability for simple and distribution transparent access to mediators. Such an interface would provide the means for applications to be easily mediator-enabled either by directly embedding a mediator system in the application through an API or providing a high-level client-server interface. Applications that access a gateway mediator are called local because they are not aware of the distribution of the mediators and they typically access only one gateway mediator.

### 3.4 Mediator components and their functionality

In this section we describe the functionality and architecture of the mediator components which are the focus of our work. For the design of the mediator components of a PMS we follow the mediator-wrapper approach described in Sect. 2.3. We design each mediator as an autonomous extensible DBMS with a query language interface, a view definition facility, local persistent storage, its own catalog and query processor. All mediators share the same query language and data model, and are capable of processing queries in terms of this query language. A very important feature of the mediators is that they treat each other as data sources, which serves as the basis for mediator compositability.

To define what is a mediator, we first distinguish a mediator system and a mediator instance. A mediator system is a software system represented by
program code and initial data necessary for the system to operate. A mediator instance is either a mediator system instantiated as a process on a computer node, or a mediator system that was executing on a computer node and which state was persistently stored so that the mediator instance can be fully restored. Thus a mediator instance would normally be a mediator system that is being used to integrate data sources, and contains integration views, and possibly other stored data and meta-data defined by the user(s). For short, we will use the term “mediator” in the sense of “mediator system”.

Figure 3.1: Distribution of mediator functionality across components.

At a high level, the mediators are divided into two architectural tiers: a mediator DBMS (MEDBMS) tier that is responsible for information integration and processing of user queries, and a wrapper tier responsible for source access\(^1\). A mediator consists of one MEDBMS, one wrapper for external mediators and any number of optional wrappers for other types of sources. Wrappers are designed in a generic way so that one wrapper can access multiple instances of the same data source type. For reusability, simplicity and flexibility of the mediators, the mediation functionality described in Sect. 2.3, p. 2.3, is distributed between the wrapper and the MEDBMS tiers as illustrated on Fig. 3.1. In the next two sections we describe the functionality of the two mediator tiers.

3.4.1 Wrappers

The wrapper components are responsible for data model mapping from the source’s data model to the mediator CDM. Unlike other mediator architectures \[58, 51\] wrappers are internal, non-autonomous components of a mediator, that are tightly connected to and controlled by the mediator. Thus wrappers are not components of a PMS by themselves and are “invisible” outside their mediators. Each wrapper component consists of two main sub-components - a source interface and an optional translator.

\(^1\)Thus we resolve the first naming problem mentioned in Sect. 2.3, by naming the part of the mediator complementary to the wrapper as “MEDBMS tier” instead of using the overloaded term “mediator”.

32
The source interface provides functions to connect to sources of some type, access data and meta-data in the sources, manage session information, and, when possible, retrieve source statistics. The data access functions of a wrapper are responsible for sending input data to the data source, the invocation of some functionality at the data source, the retrieval of the resulting data, and transformation of that data into the mediator CDM. Source interface functions return to the MEDBMS data objects in terms of its own CDM. In addition, during data transformation the source interface component may perform various data cleaning and semantic enrichment tasks, such as replacing missing values with defaults, or inferring the type of retrieved data (e.g. recognizing strings as dates or numbers). Source interfaces hide only some of the system heterogeneity of the sources - that of their low-level interfaces.

As already mentioned in Sect. 3.3.1 many types of sources can still be heterogeneous in their computational capabilities. For such sources wrappers need a translator component that “knows” how to translate operations expressed in terms of the mediator query language into operations that can be computed by the corresponding type of sources. A translator consists of source capabilities descriptions and rewrite rules. Source capabilities roughly describe the operations that a source supports, while rewrite rules provide detailed translation of expressions in the mediator query language into requests or language expressions in executable the sources. Examples of data sources for which only a source interface is sufficient are storage managers such as BerkleyDB\(^2\) which provides simple data access operations that can be easily mapped directly to operations in the MEDBMS.

An example of simple sources that require translation is a source that provides only range access via non-strict inequalities only. If queries to such sources require strict inequalities, the strict inequality in the query has to be translated into a combination of a non-strict inequalities that can be computed by the source and additional inequality tests that to be performed by the MEDBMS. It is possible to access some types of data sources both only through a source interface or with an additional translator for better performance. As an example we point to RDBMS sources. They can be treated simply as storage managers with a simple interface to scan tables and get tuples by key. Then all other operations must be performed by the MEDBMS. For better efficiency a translator may be added that would push whole query sub-expressions to the relational source. This approach to wrapper building provides the means to construct wrappers incrementally - first provide a minimal wrapper only with data and meta-data access functionality, and then gradually add functionality for source statistics, and a translator with source capabilities and rewrites.

\(^2\)www.sleepycat.com
3.4.2 Mediator DBMS

The MEDBMS component provides functionality to perform schema integration of many data source instances and to query integrated schemas. This functionality is available through constructs of the mediator query language that are suitable for the resolution of various types of information heterogeneity. Unlike wrappers which are created for each data source type, the integration constructs deal with the semantics of the data in the sources and therefore are used at the data source instance level.

To fulfill requirement R7, Sect 1, our mediators provide a functional and object-oriented (OO) common data model and a relationally complete query language based on the Daplex functional data model [48]. The mediator data model and query language are described in detail in Paper B. The functional OO data model provides powerful modeling capabilities that allow to represent the data in most most existing kinds of sources starting from flat files, to relational databases [12], object databases and even product models of engineering artifacts [29]. In particular, the concept of function in the query language presents a perfect match to the view of data sources as sets of computations that possibly require input data.

More specifically schema integration in our architecture is decomposed into the following tasks.

- **Data transformation.** While the wrapper tier performs various data transformations, this is done automatically for all data sources of the same type. Often these automatic transformations may not be sufficient and additional transformations may be necessary that are related to the data semantics and thus depend on the source instance. For example strings in a Web document may be converted by a wrapper to numbers, but the application domain may require these numbers to be rounded to some precision. Data transformations may also be necessary to extract individual items from complex values, e.g. to extract the first and last names of persons from a string, or to merge individual items into one value. Data transformations are seldom used alone. Typically they are used as parts of the more complex transformations described next.

- **Schema restructuring** is used to map both semantically and structurally heterogeneous sources into uniform representations which can be further integrated. Schema restructuring involves operations like: renaming of attributes and data sets, using data transformation to align attribute data types, addition of new (possibly computed) attributes or merge of several attributes into one, changing the schema concept used to represent a concept in a data source, and restricting a data set to some subset. Schema restructuring is performed over the schema elements of a data source instance. The result of schema restructuring are schema elements that represent real-world entities from the same domain in the same way.
• **Unification of overlapping data.** When integrating data sources that model the same or related application domains, the sources may contain data items that represent real world objects of the same kind. There are two general cases: either some real-world entities are represented in more than one data source *overlapping sources*, or there is no overlap between the sources. The latter case is the simpler one. For non-overlapping sources it is sufficient to restructure their schemas so that they have compatible structure, after which the sources can be merged by a union operation.

The case when sources overlap poses two problems. First, it requires that data objects which represent the same real-world entity are matched. This requires object identity to be defined in some way and (possibly) different representations of object identities to be mapped. This can be solved either by applying schema restructuring, or by directly using data transformation. Second, once object identity can be established, matching data items may not agree on the values of some attributes. In some cases such data conflicts can be resolved automatically by default operations for each attribute data type, e.g. always take attribute values from one of the sources, or always compute average of numeric values. However, in many cases the data semantics may be more complex and may require a human to explicitly specify data conflict resolution rules.

When source overlap, the user may want to define a view that contains various subsets of all objects in the sources. The most common case is a view that contains all real world objects from all sources without the duplicates. Another case is a view that contains only the objects present in all sources. Finally a user may be interested in the real-world objects present only in some sources.

• **Reduction and summarization.** The integration of many sources may result in views that contain very large amounts of data while a user may be interested only in some general properties of data sets as a whole like trends, averages, etc. Data reduction and summarization tasks can be performed as part of any of the previous two stages or separately over the integrated views.

To support these schema integration tasks, our mediators’ query language has several features that interact with each other: i) support for extensibility through foreign functions, ii) a view definition facility, iii) reflectiveness, by which schema objects are treated as other data items and can be queried, and iv) *global query facilities* that allow for a mediator to specify queries in terms of database objects in other data sources, including mediators. Next we introduce our mediator data model and query language in terms of which these features are realized and point out how the mediator language constructs realize the data integration functions listed above.
Data model and query language.

The basic modeling concept in the mediator data model is the object. Objects are classified in types. Attributes of objects and relationships between types of objects are expressed through functions. While objects model real-world entities, in general functions represent computations. Depending on how a computation is implemented we distinguish several kinds of functions - stored functions store explicitly the result of a computation, derived functions specify the result of a computation as a declarative query defined in terms of other functions, database procedures describe computations in a procedural language that uses the mediator data model, and foreign functions represent computations specified in an external language(s) and/or module(s). To model arbitrary computations, functions are annotated with binding patterns [34] that specify inputs and outputs. Each binding pattern may have its own implementation that computes the foreign function in the most efficient way. To allow the MEDBMS query processor to pick the best foreign function implementation when several are applicable, each binding pattern also has a cost function associated with it. Functions that have more than one binding pattern associated with them are called multi-directional. All kinds of functions can be used anywhere in the query language where a function can be used.

All objects of a type constitute the extent of that type. Thus types can be viewed as named sets of objects with the same structure. All types are organized in a multiple inheritance hierarchy where the extent of a subtype is a subset of the extents of its super-types (extent-subset semantics).

The mediator data model is reflective [38] in the sense that all data model concepts are represented in terms of the data model by meta-objects classified in meta-types. Types and functions are objects themselves and are instances correspondingly of the meta-types Type and Function. Other metatypes describe various aspects of the schema of a mediator, its knowledge about other mediators, data sources, applications and even its internal state. Since all meta-type objects are no different from the user objects, mediators are inspectable via their query language through queries that can freely mix user types and meta-types. This approach provides flexibility when inspecting mediators combined with the simplicity of using the same query language for data and meta-data retrieval.

Data integration functionality.

Data transformation and data reduction and summarization are supported directly through foreign functions and database procedures. Since foreign functions can be implemented in external languages as C and Java, the mediator user may add new functions that perform arbitrary specialized computations to transform data in an application domain-specific manner (e.g. to apply an image filter to image data) or to summarize domain-specific data (e.g. to com-
pute the average lightness of images). Foreign functions[34] are similar to, but simpler and yet more expressive, than user-defined functions (UDFs) in object-relational DBMS.

The mediator query language has a view\textsuperscript{3} definition capability through derived functions which are named and parameterized queries specified in terms of an SQL-like \texttt{select-from-where} statements and derived types, which are types with their extents specified as queries. Database views address different aspects of schema restructuring and unification of overlapping data. For the schema restructuring tasks it is sufficient to use derived functions. For schema unification a more suitable construct are derived types which provide simple to use syntax to specify rules to match data items from different data sources, and rules to reconcile conflicting attribute values.

Views by themselves are not sufficient to integrate many data sources. For that the mediator query language has the ability to refer to schema elements and objects in other data sources and use them transparently in all language constructs as if they are local. This allows free mixing of local and remote functions, types and objects both in derived types and derived functions. We call this feature \textit{global query facilities} because the query language allows to refer to any globally accessible object in a mediator or a data source. Logical compositions of mediators and other data sources are defined declaratively in terms of each mediator’s global query facilities when views in one mediator are defined in terms of other data sources and views in other mediators.

The reflective nature of the mediator data model, combined with its global query facilities and meta-model of data sources (described in \textit{Paper B}), allows queries to be issued over the meta-data of any mediator peer and/or data source. This allows to perform information discovery in a network of mediators and data sources through regular queries. The resolution of structural heterogeneity can be approached by parameterizing schema elements in the integration views (e.g. parameterized relation names) and mixing data and meta-data in the same query or view.

Integrated schemas in terms of the mediator query language are constructed from the sources’ schemas in a bottom-up fashion using the global-as-view approach. First, storage elements and computations in the sources are mapped by the corresponding source wrappers to mediator schema objects. After this initial step, data sources logically become part of the mediator database, but there still are semantic differences between the data in different objects. These differences are reconciled through the definition of views (derived types and functions) defined in terms of the source types and functions. These integrated views are then available to other mediators for further integration according to their needs and application domain.

\textsuperscript{3}Here we use the general term \textit{view} to denote any declarative specification of derived data from other stored or derived data in terms of a query language.
3.5 Systems of Peer Mediators.

In this section we describe how the three types of software components, described in Sect. 3.3, form together a peer mediator system.

Originally mediators are defined [55] as a middle layer that is distinct both from the underlying data source layer and the higher application layer. However, many software components that belong either to the data source or to the application layer exhibit some functionality characteristic for mediator systems. Since mediators may serve both as intelligent data sources themselves for other mediators and as applications for others, we will consider a Peer Mediator System (PMS) to consist of not only the mediators themselves but also of global data sources and global applications all of which communicate over a network interface. This approach allows uniform treatment of all issues regarding the joint operation of applications, mediators and data sources. In particular it allows to design a system that is capable of introspection and through that to facilitate (semi-) automatic integration and adaptive behavior.

From our general mediation architecture it follows that a PMS has at least one mediator, and any number of global applications that access any number of data sources through the mediator(s). Collectively the mediators, the global applications and global data sources are the peers in a PMS.

Every peer has a globally unique logical identifier called peer name independent of its physical network address. Since data source peers use global naming schemes different from that of the PMS, there is at least one mediator peer that provides a mapping between logical names in the PMS to network-dependent locations of the data sources. Global applications by definition support the inter-mediator interface and thus support the same naming scheme as the mediators. The set of peers whose names are known to a peer are called the peer neighbors and are related by a neighbor relationship.

We define a PMS instance as follows. Given a nonempty set of peers \( N \) with at least one mediator peer, where the peers can physically reach each other over a computer network, a PMS instance is the set of all peers \( P, P \subset N \), formed by the transitive closure of the relationship neighbor over the set of peers \( N \) starting from one of the mediator peers in \( N \).

Mediator peers can form named groups, called communities, which would be typically formed because of common interests of the mediator owners. Groups can be nested arbitrarily and mediators can participate in more than one groups, as in [20, 7]. Some mediators act as meta-mediators that know about other mediators and groups. Each group has at least one meta-mediator that stores at least the logical identifiers of its members, their corresponding physical addresses and the name of the group. Meta-mediators may store additional information both about mediators and mediator groups such as content descriptions, statistics, etc.
A PMS is completely decentralized - there is no global meta-data repository (catalog) with information about all peers, and there is no central controller that coordinates all peers. As a consequence:

- No peer has global knowledge about all other peers.
- Since the mediators are completely autonomous, there may be several mediators that define different (possibly conflicting) virtual databases over the same set of sources.
- The only way that data and meta-data can be acquired by peers is by sending requests (usually as queries) to known peers (which may trigger queries to other peers).

Mediators cooperate directly with each other and all control, data and meta-data are distributed among the mediators. Each mediator peer chooses the peers it wants to cooperate with (both as its clients and/or servers) among its neighbors, and has only limited knowledge about a subset of all available peers. Each mediator locally plans its actions based on its local knowledge. Global computations that involve many mediators are planned as a result of many local cooperative decisions. Applications and mediators may recursively initiate cooperation between peers on behalf of other peers or applications.

Most importantly, there is no global integrated view as in federated database systems and centralized mediators. Each mediator defines its own integrated view over a subset of all data sources and mediators and makes some part of its integrated view available to other mediators and applications for further integration or querying. Each mediator peer has total control of its own schema. Finally, mediators may join and leave a PMS at any time.

Our definition of a PMS allows PMS instances to have a very wide range of logical topologies from a client-server with many applications, 1 mediator, and many data sources, to a “pure” P2P system where all applications have gateway mediators and all data sources are embedded in mediators, and there is a network of mediators between the application and the data source layer.

An example of a PMS is shown on Fig. 3.2 where several mediators are defined in terms of other mediators and data sources. In the example, applications access data in several data sources of different kinds (two RDBMS, one Internet search engine and a Web site) through a collection of composed mediator servers. The directed arcs connecting the mediator nodes and data sources correspond to the relationship “defined in terms of” between them, that is, the mediators that point to other mediators or sources contain views that are defined in terms of views or data in the pointed to mediators and sources. We illustrate this in the upper-left mediator in the figure, where a global view is defined in terms of other two mediators. It is important to point out that mediator compositions are not defined as static networks of mediators but are dynamically generated through the definition of queries or views. Each query or a view uses only a subset of all logical links, defined by the transitive closure.
of the logical relationship “defined in terms of” between all views referred by that particular query or view. Thus Fig. 3.2 is a simplified view of the union of several superimposed logical mediator compositions.

The advantages of the P2P approach for mediation are that it allows the domain experts to own and control independently their mediators in the same way as data source owners have total control over the data sources. Each mediator may evolve at its own pace as long as it preserves its public interface. In the foreseeable future it may be expected that data integration will remain a predominantly “manual” task that requires a lot of domain knowledge and human participation. A P2P architecture allows to distribute the integration effort between many autonomous domain experts and thus scale the integration process. The domain knowledge encoded in the mediators is shared so that other more complex mediators can be composed in terms of simpler ones and thus integrate data across many data sources and knowledge domains in a scalable manner. Finally, a P2P architecture promotes reuse of computation resources such as storage, CPU cycles, and specialized software and hardware.

Figure 3.2: An example of a peer mediator system

![Diagram of a peer mediator system]
A successful implementation of the PMS architecture presented in Sect. 3 must fulfill a wide range of requirements, some of which are discussed in Sect. 3.2. Our focus is on the most fundamental requirement for the PMS architecture, that of composability of mediators in terms of other mediators and data sources. As discussed in Sect. 3.2 the fulfillment of this requirement results in a data integration architecture that meets the general requirements \( R1-R7 \) for large-scale data integration stated in Sect. 1.

The problem we address in this dissertation at a high-level is how to implement mediator composability effectively and efficiently in a PMS architecture so that a PMS system can scale over the number of composed mediators. This general problem can be decomposed into two sub-problems described below.

- **Scalable integration.** The first aspect of mediator composability is how mediator compositions are defined so that many views from many mediators are integrated into higher-level reconciled views. Compared to centralized mediation architectures, in a PMS this problem has the additional complication that the views are defined in many mediators and there is no central repository that keeps track of all existing views in all mediators. We address this problem by providing a query language with global query capabilities. However, the problems remain \( i) \) how to discover the relevant views to a problem domain, and \( ii) \) how to specify in a scalable manner integrated views over large number of views. While these two problems are very important, in our work we assume that method(s) exist to specify integrated global views over many mediators. This can be done manually directly in terms of the global query capabilities of the mediator query language\([23, 24]\), or (semi-) automatically through the use of visual tools and inference mechanisms\([57]\). In addition the mediator query language may be extended with more expressive constructs for data integration. Given the high expressive power of the mediator query language and data model, we believe that future tools or language constructs can be expressed in terms of the existing language features. Thus in the rest of our discussion we will assume that integrated global views are preexisting and are specified in terms of the mediator query language as presented in Paper B.
Query processing. Assuming that the means exist to specify mediator compositions, the next important problem is how to provide scalable performance for the computation of queries against composed mediators so that a PMS is usable. Composability of mediators has two dimensions. Logical composability is related to the means of specifying compositions of mediators in terms of a declarative query language. Physical composability is related to the means by which mediators physically interact with each other and with external sources as one distributed system. In order to compute answers of queries in a PMS, logical mediator compositions must be represented as physical ones.

Our mediator compositions are described in terms of a query language, therefore the process of translating logical view compositions into physical ones is in fact query compilation, while the computation of query results according to a physical composition of mediators is query execution.

The two problems are tightly interrelated. On one side various approaches can be envisioned to integrate many mediators and views such as tools and language constructs. On the other side only some of these approaches may be viable because of limits on their performance. Based on our analysis of related work in the area of data integration, we conclude that while a considerable amount of work has been done in the area of data models and query languages for data integration that can be applied to a PMS architecture, the problems specific to query processing in peer-to-peer architectures for data integration have not been adequately addressed.

Thus, query processing in peer mediators itself poses a wide variety of challenges. In the remaining of this section we discuss several interrelated sub-problems that we address in this dissertation.

Capabilities of inter-mediator interfaces.

One of the most fundamental issues for a distributed system is how to design the public interfaces of the components in this system so that they can interoperate, are easy to evolve, and are efficient. In large scale P2P systems it is also important that the peer interfaces provide enough expressive power so that the distributed system as a whole can self-organize itself to perform efficiently as a whole. In particular the interfaces of the PMS components should be sufficient for them to cooperatively process global queries in an efficient way.

As noted in our discussion on data sources in Sect. 3.3, low-level interfaces provide the communication infrastructure for distributed systems, but they do not solve the problems of the semantics and granularity of the interfaces, that is, what functionality is exposed through an interface and what is the granularity of the interface. By functionality we mean what computations does one system expose through its interfaces, and by granularity we mean at what granularity does a system provide a view of its internal state through an interface.
Thus, independent of the low-level infrastructure used for interoperability between the components in a PMS, there is a large space of design choices related to the functionality that PMS components should expose to enable efficient cooperation between them. *Paper B* investigates what computational capabilities a software component should provide in order for that component to participate as a peer in a PMS.

*Overhead of logical mediator compositions.*

Logical composability and autonomy of mediators poses several challenges to the computation of queries over integrated global views. Since there is no global control in a PMS, every mediator owner has the freedom to compose arbitrary global views defined in terms of any of the known and accessible mediators and data sources. This ability to compose new mediators in a globally uncontrolled manner may result in enormous redundancy in large mediator compositions. Typically a mediator will be aware of and will integrate a relatively small number of sources and neighbor mediators that provide information of interest. However, the neighbor mediators may derive their information from any number of other mediators and sources not known directly to the first one. In this way it may be common that data from the same mediator(s) and/or source(s) is indirectly integrated by a mediator through many levels of other mediators, where each one eventually adds some value by restructuring and enriching the information from the lower levels.

If queries over such composed mediators are executed naively by following the logical links between the mediators, this may result in many redundant computations performed by each of the underlying mediators, as well as in many redundant network accesses and data transfers, which may result in an unusable PMS.

Therefore methods need to be developed that remove these redundancies and generate efficient query execution plans (physical compositions of mediators). Since logical mediator compositions are essentially views defined in terms of other views, these views can be expanded (unfolded) as in traditional DBMS. However, in a P2P setting there is no central catalog and typically no mediator “knows” the definitions of external views. Another issue is that in traditional database design the database schema is designed in a top-down fashion and one may expect it to be relatively well designed and have relatively small number of levels in the view definitions. However, due to the uncontrolled bottom-up design of data integration solutions, it may be expected that very large number of views will be nested very deeply. Finally, due to mediator autonomy, some mediators may refuse to make their view definitions available to others, e.g. because they want to hide their information sources. To respect each other’s autonomy, mediators should be able to negotiate if and which views can be expanded, and be able to compile and execute queries in...
all cases. Therefore view expansion in a P2P setting may not be as “simple” as in a traditional DBMS setting. Paper C studies the problem of view expansion in the presented PMS architecture.

Decentralized query processing.
A decentralized architecture of many autonomous, but equal in capabilities peers, such as the PMS architecture, presents new opportunities and problems for the processing of global queries. In a centralized distributed DBMS system, there is one controlling peer, typically the peer where a query is issued, that is responsible for the compilation and execution of its queries. This is possible because there is a central catalog with all meta-data necessary to produce optimal QEPs, and because the component DBMSs give up their autonomy and leave the control to one peer. However, in a decentralized system, no peer has global knowledge, or global control over the other peers. One alternative to approach the lack of meta-data is to request it from the other peers involved in a query. Another possibility is to use the fact that the other mediator peers have their own query processors and local meta-data and thus may take better decisions regarding local queries. Thus, instead of exchanging meta-data, an alternative is to submit queries for remote compilation. In addition such distributed compilation provides the means for load balancing during query compilation. Another side of the problem is query execution in a centralized system. There, one peer controls other peers during query execution. As a result all data flows through the central peer. In a P2P mediator system, where peers are distributed across a wide-area network with highly varying link parameters, centralized data flow may be far from optimal. Instead, it may be much more efficient to exchange data directly between peers that are connected with fast links and and let them cooperate to compute intermediate results which can be shipped to the query peer or some other intermediate peer. As with cooperative compilation, such cooperative execution provides the additional possibility for utilization of the resources of all peers. In Paper D we study one particular method for optimizing global queries through distributed compilation that produces decentralized QEPs.

Distributed join methods for mediation.
Data integration problems often require cross- source or mediator join operations because of overlapping information in the sources and/or mediators. Join is known to be the most expensive operation in database systems. The presented PMS architecture is different from centralized and distributed but homogeneous DBMS architectures in that joins have to be made between mediators and sources with limited capabilities or computational sources often over slow network connections. With such sources, data produced by one of the join operands is required by the other operand as input, and therefore this interme-
mediate result data has to be shipped from one operand to the other. Such joins are often called dependent because the execution of one of the join operands depends on the execution of the other. Thus mediator systems need specialized methods for the execution of dependent joins that take into account and reduce data shipping costs together with the cost of join computation. The focus of Paper E is the design and study of three mediation-specific join strategies.

Access to diverse sources.
A mediation system would typically access a wide variety of data sources. It is hard to predict in advance even what will be the future kinds of sources that need to be integrated as a data integration system evolves. Therefore the mediator components must be designed in a way that allows new sources to be added easily and dynamically. Since sources are accessed through wrapper components, this amounts to the question how to design a generic mediator-to-wrapper interface and meta-model of data sources that allows the addition of new wrappers for new kinds of sources. Another, more specific question is, given the presented mediation architecture, is it flexible enough to easily accommodate new kinds of sources? We address this problem in Paper F, where we design and investigate a wrapper for several Internet search engines as an example of non-database-like data sources.
Related Work

In this section we overview works related to the PMS architecture which serves as the basis for our work, we point out the similarities and differences between our architecture and other projects, and summarize how these projects relate to the query processing problems described in Sect. 4.

5.1 Distributed Database Systems

There is a large body of knowledge on query processing in distributed databases [30] that may provide partial solutions for problems related to query processing in a PMS. However, the autonomy, distribution and extensible object-oriented data model of the PMS architecture proposed here, poses new problems different from the ones related to distributed databases [37]. In distributed database systems (DDBMS) the peers are homogeneous, there is a single site that controls query processing and a centralized catalog, usually replicated in all databases. In contrast to that, in a P2P system there is no central controlling site and all meta-data is distributed among the peers. Various new problems arise from that. Here we mention only some of them: to produce a global QEP all peers involved in a query have to cooperate in the compilation process because no peer has complete knowledge of execution cost; peers have to request cost information from other peers over the network which incurs high cost of getting the cost; due to autonomy, cost information may not be available at all peers; peers cannot assume that every other peer is capable of the execution of arbitrary query fragments; therefore predicates might not be freely pushed in the QEP from one mediator to another.

5.2 Mediator Systems

A considerable number of mediator systems [10, 13, 18, 25, 51, 46, 35, 58] have been proposed with varying architectures in terms of their degree of distribution, autonomy and data model.

Many of the mediator proposals and systems have a centralized architecture, that is they consist of a single mediator component, interacting with the data sources via wrapper components. The wrappers themselves may or may not
be distributed with respect to the mediators. However, even in mediator architectures with distributed wrappers, all meta-data and control in the mediator-wrapper system as a whole are concentrated in the mediator, and wrappers in such architectures are not autonomous. Therefore, we will consider mediator systems with distributed wrappers to have still centralized architectures, where composability is not an issue. Typical example of centralized mediator systems are Garlic [13] and DB2 Federated DBMS [18]. The TSIMMIS [13] and Pegasus [10] projects mention that distributed mediators may access other mediators, but no results are reported in this area.

Some mediation prototype systems have distributed architectures where mediators access other mediators. Next we compare these mediator systems with our PMS architecture.

The AURORA prototype [58] follows a fixed two-tier mediation model consisting of three types of distributed components. The first tier consists of homogenization mediators that deal with schematic mismatches on per-source level, and distributed wrappers that provide access to the data sources. The second tier consists of integration mediators responsible for integrating multiple homogenized sources through their respective homogenization mediators. This distributed architecture is similar to our PMS architecture in that there is no single monolithic integrated view, instead the integration process is distributed among many mediators. The main focus of the AURORA project is a methodology for source homogenization and conflict resolution. Various algebraic rewrite methods are proposed for pushing integration and standard relational operations to the sources, but neither distribution of sources and/or mediators is considered in any way, nor the effects of logically composing the integration mediators in terms of many homogenization mediators. Query processing in AURORA is considered only from the viewpoint of a single homogenization mediator, and while not explicitly said, it seems to be performed in a centralized manner. Our PMS architecture generalizes that of AURORA because our mediators can be specialized to perform different roles as homogenization and/or integration, while there is no restriction on the number of mediator tiers.

The DIOM project [46, 35] is one of the few mediator projects that points out composability as an important property for scalable integration of many sources. The project presents the implementation of a distributed mediator architecture where mediators can access other distributed mediators and/or wrappers. One feature that distinguishes DIOM from other mediator projects is that it does not require conflicts to be resolved statically in an integrated schema. Semantic conflict resolution is deferred to query result assembly time instead before or at query compilation time. Thus users can dynamically specify the information they are interested in and their preferences, and the mediator performs automatically source selection and conflict resolution based on user pref-
erences. The DIOM prototype features a query processor aware of the distribution of the sources and capable of dynamic query routing and scheduling, but all query processing is performed in a centralized manner, such that query compilation and execution are controlled by only one mediator. As a result QEPs in DIOM are centralized and always follow the logical composition of the mediators. To the best of our knowledge there are no reports of the actual performance improvements achieved through the query processing approach in DIOM. In particular all reported results describe processing of queries by one mediator against several wrappers and do not address issues specific to processing queries in mediator compositions.

The DISCO mediator system [51] also has a typical mediator-wrapper architecture with distributed wrappers accessible by distributed mediators. Since every mediator is a wrapper, mediators can call other mediators as if they were wrappers. One of the prominent features of DISCO is graceful handling of unavailable sources through partial evaluation semantics that returns partial answers to queries by processing as much of a query as possible and returning the remaining non-processed part of the query to the caller. This approach to handling source unavailability can be applied to our architecture to fulfill the requirement for dynamic source availability described in Sect. 3.2. As in the DIOM system, query processing in DISCO is described only for flat two-tier cases and does not take into account problems (and optimization opportunities) resulting from mediator compositions.

Our conclusion is that, while several projects mention mediator composable as an important feature, none of these projects address issues related to query processing in many composed mediators, which is the main focus of our work. However, the described projects address other issues, important for scalable data integration of many data sources, that are complementary to our work.

5.3 Peer Data Management Systems

Several recent works propose P2P architectures for data integration and for the management of distributed and autonomous databases. The ideas presented in these works are the closest to our PMS architecture, and therefore we discuss them here in more detail.

Data management systems based on P2P computing paradigm are discussed in [16, 19, 20, 6] where new problems and opportunities arising from the usage of a P2P paradigm are identified. However, there is little work on implementation issues of such systems, especially related to large number of cooperating query processors. Even more, these works point out problems specific to P2P architectures some of which we address in this dissertation. In the vision paper [16] it is indicated that two fundamental problems in most P2P systems are
the placement and retrieval of data and therefore DBMS technology can and should be applied to P2P systems. At the same time P2P architectures can be useful in DBMS systems to provide system robustness and scalability, eliminate proprietary interests, reduce administration effort and provide anonymity.

Of the two main problems mentioned, the paper describes in more detail the problem of data placement. Solutions to this problem can be applied in our PMS architecture, e.g. for efficient caching and replication of data at the mediator peers. One of the problems related to a P2P architecture is that of the extent of knowledge sharing between peers. We analyze and provide some answers to this problem in Paper A with respect to an architecture with no centralized catalog.

Another vision work [6], addresses the problem of semantic inter-dependencies in between autonomous peer databases in the absence of a global schema. The paper introduces the Local Relational Model (LRM) as a data model specific for P2P data management systems. Inter-peer semantic dependencies are described through coordination formulas that allow the synchronization of many peer databases. The LRM can be used to mediate between multiple peers and to propagate updates between peers so that consistency is preserved. The architecture proposed for the LRM is described at a very high-level of detail, and at that level of detail it is similar to our PMS architecture. In terms of query processing in the proposed LRM model, the paper lists several P2P-specific problems, but no solutions are proposed.

At the architectural level, the works closest to ours are [20, 19]. Based on the assumption that data integration systems have one global mediated schema that integrates all sources, the two papers advocate the concept of peer data management systems (PDMS), as systems that replace the single logical schema of data integration systems with an interlinked collection of semantic mappings between the peers’ schemas. The ideas described in the two papers are implemented in the Piazza peer data management system. The main problem addressed in the two papers is that of schema mediation in a PDMS. To specify schema mappings between peer databases the authors propose a language PPL that allows to express both GAV and LAV style mappings between peer schemas. In [20] the PPL language is an extension of Datalog, and thus suitable for peers supporting the relational data model. In [19] the mapping language is modified to support RDF and XML sources. With respect to query processing, both works deal with the problem of query answering (reformulation) in the presence of mixed GAV and LAV transitive mappings between peers. The goal of query answering is to reformulate an initial query in terms of schema mappings to a query in terms of the base relations. As the authors notice in [19] they do not address the problem of efficient processing of queries which is essential for the overall performance of a PDMS.

From an architectural perspective, at the level of detail presented in [16, 19,
all these proposals including ours are related. The main differences are in the data models proposed, which is functional and object-oriented in our case, and relational and RDF/XML in the other cases; the schema mapping approaches used; and the query processing issues addressed in these works.

Regarding the problems related to query processing in a P2P architecture which are our primary interest, our work and that of [20, 19] are complimentary in several ways. The query reformulation algorithms presented there fully expand all views and rewrite all queries in terms of the base relations. As shown in Paper C selective view expansion may often lead to better results with substantially less compilation cost. Thus query reformulation in Piazza can be simplified by not expanding all views (mappings), while our PMS architecture can benefit from a more general method of mapping peer schemas and its query reformulation algorithm. Since the current work on the Piazza system is focused on query reformulation, all our solutions related to query processing in a PMS can be directly applied in Piazza and similar PDMSs.

In [42] a P2P distributed data sharing system, PeerDB, is presented and some of its aspects are experimentally evaluated. PeerDB consists of an arbitrary number of autonomous peers each of which consists of a relational DBMS (MySQL), an agent system DBAgent, and a cache manager. Peers find each other through one or more global names lookup servers that provide each node with a unique identity. PeerDB uses an information retrieval approach to the discovery of relevant information. Each relation and attribute in the peers’ databases is tagged with keywords. Relevant relations are discovered through keyword matching and ranking. Compared to our PMS architecture, PeerDB does not provide global query facilities and does allow for the definition of integration views across multiple peers. Since there is no global view definition capability, PeerDB does not provide logical composability and the peers constitute a logically “flat” system. PeerDB naturally handles peer unavailability because there is no predefined integration schema. Query processing in PeerDB is performed through “agents” that are dispatched to other peers by the DBAgent component, but the paper neither defines what is an agent, nor it describes by what algorithm(s) agents are dispatched to other peers. Finally, PeerDB does not address issues concerning access to external sources with varying capabilities. Our conclusion is that PeerDB is suitable for the sharing of structured data in a P2P fashion, but it cannot be applied to real data integration problems.

A distributed relational query processor is proposed in [7], where the focus is on dynamic extensibility and security. Advances in this project are complimentary to our work and can be applied in the presented PMS architecture. The project does not specifically address the integration of heterogeneous data sources, neither problems related to redundancy in compositions of many autonomous database.
Summary of Contributions

The hypothesis underlying this work is that a peer-to-peer mediator architecture is more suitable for many real-world data integration problems than a centralized one. It is shown to be possible to design a mediator system with a peer architecture that can process queries efficiently and can scale in terms of the number of peers. The main contributions described in this dissertation are:

- Analysis of the components of a PMS - applications, data sources and mediators. (Sect 3.3)
- Design and implementation of a P2P system for distributed data integration. In the architecture autonomous peers share data and services with other peers without a global coordinator. Mediator peers provide a unified and knowledge-enriched view of many autonomous and heterogeneous sources in terms of a functional and object-oriented common data model and query language. The integrated views can be either queried directly or can be used by other mediators to compose higher-level integration views in terms of views in other peers. (Sect 3 and Paper A)
- Analysis of the inter-peer interfaces and corresponding computational capabilities of the peers, the meta-data that needs to be exchanged between the peers, and the query processing techniques that can be used in the presence of some capabilities and meta-data in order to implement a PMS. (Paper B)
- Technique, called distributed selective view expansion (DSVE), to efficiently process queries against many composed mediator views. DSVE has been implemented in practice in the AMOS II mediator system and based on this implementation it has been experimentally evaluated. The experimental analysis of this technique shows that it is possible to provide good query performance with low compilation cost in a peer mediator system. (Paper C)
- A distributed compilation technique to re-balance left-deep QEPs which due to the autonomy of each peer, not only describe access to distributed sources, but are distributed themselves. The QEP rebalancing technique improves the quality of the QEPs in a peer mediator system by enabling direct decentralized communication between the peers involved in the computation of a query result. The QEP rebalancing technique was implemented in the AMOS II mediator system and studied experimentally. (Paper D)
- Design, and experimental study of three join algorithms for a peer mediator
Two of the algorithms, called *ship-out*, ship bindings from one of the join operands (local or remote) to another remote operand and thus are suitable for the computation of joins involving sources with limited capabilities. The third *ship-in* algorithm, ships all data to the join site, where the join is computed. Ship-in joins are suitable for sources with a scan interface accessible over a fast network. (*Paper E*)

- **Application of mediation for Internet search engines (ISEs).** Various ISEs are integrated through a flexible wrapper manager sub-system, called object-relational wrapper for ISEs (*ORWISE*), that utilizes external web wrapper toolkits and allows for flexible and dynamic addition of new ISE wrappers. The design of ORWISE shows that the basic facilities for extensibility in the AMOS II system described in *Paper A* are powerful enough to support such non-database-like sources with ease. (*Paper F*)

In addition, during my work various components of a peer mediator system have been implemented as part of the AMOS II mediator system.

- **Design and implementation of a meta-schema that models data sources.** The meta-schema allows for declarative manipulation of information related to all data sources through the mediator query language. This allows mediator users to query data source meta-data for discovery of relevant sources. In addition the mediator kernel itself has been changed to reflectively utilize the data source meta-data during query optimization. The meta-schema is described in *Paper A* and *Paper F*.

- **Experimental studies of a PMS require that large number of measurements are performed and dependencies on many parameters are investigated. This results in large volumes of distributed measurement data with complex structure. This requires that both the execution of experiments and experimental data collection are performed in an automated way. A natural approach is to use the mediator system itself to manage and collect the experimental data. To enable the performance of large-scale computation experiments in a PMS, I designed and implemented a declarative framework for automated computational experiments built on top of the AMOS II system. The framework allows to configure and execute an experiment, collect all experimental data and plot various dependencies only through the query and stored procedure language of the AMOS II system. The framework was used to perform all experiments in *Paper C*.

- **One of the most important types of data sources are RDBMSs.** The most wide-spread and standardized way to access RDBMS sources is ODBC. To make our experimental studies more relevant, a wrapper for ODBC data sources was implemented in AMOS II. The wrapper was used in all experiments in *Paper C, Paper D* and *Paper E*.

- **Many improvements in most components of AMOS II were necessary to implement the query processing techniques and to perform the experiments**
described in this dissertation. Some of the improvements led to orders of magnitude less memory consumption and smaller compilation times.
Summary of Appended Papers

The papers included in this dissertation and summarized in this section are inter-related in the following ways. Paper A describes an implementation of a PMS that uses some of the results of Paper B to process global queries. Paper B investigates inter-peer interfaces and capabilities required for the interoperability between mediator peers and/or data sources in a PMS, and the applicable query processing techniques in the presence of these interfaces. Paper C studies in detail how to process queries over mediator compositions specified in the query language described in Paper A using the view shipping approach described in Paper B. Paper D investigates query optimization techniques based on the query shipping approach described in Paper B. Paper E describes distributed join methods for the PMS described in Paper A. Finally, Paper F describes how to add new wrappers for Internet search engines to the PMS presented in Paper A as a test case for mediator extensibility.

The overall structure of the dissertation is depicted on Fig. 7.1 where the thin lines represent the relationship “uses results from”.

Figure 7.1: Logical organization of the dissertation.

7.1 Paper A: Functional Data Integration in a Distributed Mediator System

This paper describes an implementation of the PMS architecture presented in Sect. 3 in the framework of the AMOS II (Active Mediator Object System) mediator system. In this section we will summarize the main features of the
AMOS II mediator system from Paper A and will point out their relationship to the other works that constitute this dissertation. We will also point out the limitations of the current implementation with respect to the general PMS architecture.

With respect to the requirements for a PMS listed in Sect. 3.2, AMOS II mediators are fully composable at the logical and physical level. Location transparency is supported through unique names assigned to each mediator and a name resolution service described below. Peer discovery is supported through the data source meta-schema that models data sources and can be queried via the AmosQL query language. The issues that are not addressed in the current implementation of the PMS architecture are dynamic availability of sources, security, replication and caching.

With respect to the general mediator architecture, the AMOS II mediator system described in this paper is different in that there is only one meta-mediator, called nameserver that stores the mapping between logical mediator names and physical network addresses. The second difference is that all data sources are treated as local by the mediators, and thus even if the same global data source is accessed by several mediators, such information cannot be discovered and utilized by the mediators.

Further details about the data model and query language of AMOS II, its reconciliation primitives, and its query processing can be found in Paper A and its references.

Comments
The main contribution I made to this paper is in the design and description of the data source meta-model and description and clarification of the multi-database query processing. I wrote the respective sections and participated in other parts of the paper clarifying various aspects of the functional approach to distributed mediation.

7.2 Paper B: Interface Capabilities for Query Processing in Peer Mediator Systems

Each peer in a PMS must provide an interface to its data and meta-data sufficient to allow the cooperative processing of queries in a PMS. In Paper B we analyze the computational capabilities and meta-data that a software system has to export through its interfaces in order to participate as a peer in a PMS, and the corresponding query processing techniques that can be applied in the presence of some meta-data and capabilities of the peers. Our analysis is based on the functional data model and query language for data integration, presented in detail in Paper A. We model data collections in remote peers as proxy functions that describe the data types and relationship of the data items.
Proxy functions may be implemented in different ways, and queries over such functions may be also processed in different ways depending on the computational capabilities of the peers in a PMS. Based on the concept of proxy functions, we identify and compare six classes of peer capabilities with increasing complexity, summarized below.

- **Single-directional proxy functions (SDPF).** The simplest possible way to interface peers in a PMS, so that mediators can compute inter-peer queries, is to assume that remote peers provide some interface to directly access their data and to associate each proxy function with an implementation that computes the results of the function by using the data access interfaces at the remote peers. Proxy functions implemented in this way are computable in RPC-like manner only in one direction - from their input parameters to their results, and thus are termed as single-directional proxy functions (SDPF).

- **Multi-directional proxy functions (MDPF).** Often it may happen that several SDPFs represent different directions of the same abstract relationship stored at a remote source. As a result users have many alternative ways to specify the same query, each with potentially very different performance. To offload the user from performance considerations, we introduce multi-directional proxy functions (MDPFs) that tie together several SDPFs into one multi-directional proxy function. MDPFs provide higher degree of abstraction for the mediator users than SDPFs, and better query performance through query optimization.

- **Multi-peer proxy functions.** Some of the functions referenced in a global query may be computable at more than one peer. With SDPFs or MDPFs it is up to the user to choose which alternative to use which requires users to deal with performance issues and may result in sub-optimal query performance. To alleviate this, we introduce an additional level of abstraction through multi-peer proxy functions (MPPFs), where one MPPF relates together all MDPFs (or SDPFs) that represent the same computation in different peers. Thus MPPFs shift the task of choosing the optimal peer for a function from the user to the mediator query processor.

- **Plan shipping.** All previous three interface classes assume simple peers that provide only direct access to their data through some interface, and all query operators, such as join, must be performed at the query peers. This requires that all data during query execution is shipped through the query mediator in a centralized manner. However, remote peers may be capable of computing groups of database operations in the form of query plans. Such capability can be utilized by the mediator peers through plan shipping where the mediators compile an inter-peer QEP, identify the portions of this QEP, called sub-plans, that can be computed by other peers, and ship these sub-plans to the remote peers for execution.

- **Query shipping.** Many data sources, such as relational DBMS, the medi-
ators themselves, provide a declarative query interface to their data. Thus, an alternative to plan shipping is to group together through query decomposition the query operations computable at the same remote peer into sub-queries, and to ship them for compilation and execution to the corresponding peer.

- **View definition shipping.** In a PMS, mediator peers can be freely composed logically in terms of other mediators and data source peers through database views. This may result in a network of logically composed peers with redundancy, where many peers integrate the same source peers and even the same remote views through many different logical paths. This logical redundancy may result in many redundant computations and network data transfers. To discover and remove such redundancy, the peers must be capable of exchanging view definitions, so that the query peers can analyze and optimize together the expanded view definitions.

The analysis of inter-peer interface capabilities and the related query processing techniques presented above is based on our experiences from the implementation of the AMOS II peer mediator system. We describe the implementation of a PMS in the AMOS II mediator system with peer capabilities within each group. The description of our PMS implementation relates together the results of most of the papers in this dissertation. Since queries over many peers are always reduced to SDPFs, and join is one the most common and expensive database operation, in **Paper E** we design and study the performance of three algorithms for computing inter-peer joins over SDPFs. In **Paper D** we study an application of query shipping for rebalancing left-deep global query execution plans to produce decentralized inter-peer QEPs. Finally, **Paper C** investigates techniques to implement view definition shipping that improve the quality of QEPs with low compilation cost.

**Comments**
The work described in this paper was done by me with discussions with Tore Risch.

### 7.3 Paper C: Scalable View Expansion in a Peer Mediator System

Views are the central concept for data integration in the PMS architecture. This paper studies in detail the view shipping query processing for peers with view shipping capabilities described in **Paper B** as a promising technique for efficient processing of global queries over views defined in many peers.

There are two well-studied approaches to implement distributed information systems. The first treats each of the distributed modules of an information system as black boxes. The modules communicate with each other through
some protocol without revealing the implementation of the services they export. This is the approach used in CORBA based systems [1]. On the other end are distributed database systems where database views are fully expanded [44] independent of the location of the base tables and views that are used in a view definition. We term the first approach as the black-box and the second as transparent box approach.

The black-box approach provides full autonomy of the mediators, while at the same time compiling queries without expanding all view definitions may result in sub-optimal execution plans due to missed optimization opportunities and many redundancies in mediator compositions. Without view definitions being expanded, client mediators cannot ‘see’ that their sub-mediators have views implemented in terms of the same common sub-mediator. As the experiments in this paper show, such redundancies often lead to very inefficient QEPs.

To solve the problems of the black-box approach, DBMS query compilers expand view definitions. This ‘reveals’ to the query compiler the information ‘hidden’ in the view definitions which allows for better quality execution plans by optimizing together queries with all directly and indirectly referenced view definitions. In a PMS, view expansion may also allow to combine the view definitions from various mediators, discover and remove redundant accesses to intermediate mediators and push the resulting merged query down to the mediator(s) that actually contain/produce the data of interest. As one may expect, such compilation techniques lead to several orders improvement in the quality of a QEP. However, expanding all participating mediator definitions may result in high compilation cost as many more mediators may become ‘visible’ to the mediator that compiles a query and many more predicates are added to the initial query. In large mediator compositions this may lead to prohibitively high compilation cost because of very large queries and large number of mediators.

A natural idea is to combine both approaches and treat the mediators as grey boxes with varying level of transparency. This paper presents and studies experimentally an implementation of the grey-box approach in a new query compilation technique for P2P mediators - distributed selective view expansion (DSVE). In DSVE for better performance mediators can control the level of transparency by selectively expanding only some multi-mediator views. To preserve their autonomy, mediator peers can decide whether to fulfill or not view definition requests. The performance improvements with DSVE are due to more selective queries, smaller data flows between the servers, fewer servers involved in the query execution while spending relatively little effort in query compilation.

DSVE is implemented in the AMOS II mediator system described in Paper A. The implementation is studied experimentally in two scenarios with up to 20 mediator peers to determine the effects of selective expansion of multi-
mediator views on the quality of QEPs. The study shows that one of the most
important factors for the overall performance of a P2P mediator system is the
topology of the logical mediator composition (i.e. of the graph defined by the
mediator peers as graph nodes and the relationship ‘defined in terms of’ as
graph arcs). Our experiments show that in mediator compositions with 10 and
more peers DSVE reduces query compilation time with orders of magnitude
with minor losses in the QEP quality and thus DSVE allows for efficient query
processing in logically composed mediators.

Comments
I am the main author of the paper. My main contribution to the paper was in
the scenario description, view expansion algorithm description and the exper-
imental section. I proposed the idea to selectively expand views as a genera-
ization of traditional full view expansion. An initial implementation of full
view expansion was done by Vanja Josifovski. I modified and extended this
implementation to support both full and partial view expansion. I also pro-
posed and implemented the mediation scenario and designed and performed
all experiments.

7.4 Paper D: Optimizing Queries in Distributed and Com-
posable Mediators
One of the challenges in processing queries against many composed mediators
is how to determine an optimal data flow between the mediators during query
execution and where to compute intermediate join results. The approach typ-
ically taken in mediator architectures is that the mediator to which a query is
posed, called client mediator, compiles locally and executes by itself a global
QEP for that query. As a result all data and control flow pass through that
mediator in a centralized manner, where all inter-mediator joins between its
sub-mediators are computed. Depending on the quality of the physical links
between all participating mediators and their processing resources, such cen-
tralized plans may not be always the most efficient. For example, when the
links between the sub-mediators are faster than the links between them and
the client mediator it may be more efficient to let those mediators directly
exchange data and compute intermediate results without involving the client
mediator. As pointed out in Paper B, for this mediators must support either
a plan shipping or a query shipping functionality, so that the client mediator
can instruct its sub-mediators to process global sub-queries that directly access
other sub-mediators.

For the optimization of global queries over many levels of composed me-
diators we take a two-phase approach which we have implemented in the
AMOS II PMS described in Paper A. In the first phase, the client mediator
decomposes global queries into sub-queries, each of them local with respect to some remote sub-mediator. To reduce the cost of query optimization, at this phase the optimizer searches only the space of left-deep QEPs. The resulting left-deep query plan tree, stored at the client mediator, specifies the order in which the client mediator will perform joins between the remote sub-queries in the sub-mediators, and may contain additional local operations. Since all joins are performed at the client mediator, this results in centralized plans.

In order to produce decentralized plans where sub-mediators communicate directly and perform some of the joins themselves, the initial QEP has to be decentralized into one or more global sub-plans that are computed by the sub-mediators. This is performed by a second query optimization step, called query plan tree distribution, or tree distribution for short. The tree distribution optimization phase is the focus of this paper.

One way to decentralize an initial QEP is to let the client mediator’s optimizer explore all possible allocations of joins to sub-mediators and then to send join sub-plans (via plan shipping) to those sub-mediators using the plan shipping approach described in Paper B. In a PMS architecture this approach has several problems: i) it would require centralized decision making for which the client mediator would need to know the cost of executing joins by other mediators, ii) since there is no global catalog this would incur many costing requests, and iii) it does not respect the autonomy of the sub-mediators.

A second possibility is to reuse the mediator’s capability to accept sub-queries and locally compile them for further execution. Since each mediator provides a global query language, requests for the execution of global sub-plans can be submitted to remote sub-mediators in a declarative form using the query shipping approach and let the sub-mediators decide on the exact execution plan. In this way a global query is compiled cooperatively by the participating mediators, where each mediator both compiles and executes a piece of the global QEP. The advantages of this approach are that: i) mediators can make cost estimates without performing remote cost requests, ii) mediator’s autonomy is respected since each one can decide whether and how to execute a sub-query, and iii) better load distribution is achieved not only during query execution, but also at query compilation time.

The main contribution of this paper is a tree distribution algorithm based on the query shipping approach. The algorithm starts with a centralized left-deep global QEP. This initial tree is transformed by a series of plan node merge operations. A node merge operation takes two randomly chosen neighbor nodes, generates a sub-query that describes the join of the two nodes and replaces the two original nodes with one that accesses the merged sub-query. A node merge operation is performed only when it reduces the total QEP cost. For that the merged sub-query is compiled at both sub-mediators, the current plan cost is compared with the costs of the two new plans, and the cheapest of the three
is chosen. The process continues until no beneficial merge operations can be performed. The node merge operations replace the inner relations of the initial QEP with composite joins, therefore the algorithm essentially re-balances the initial centralized left-deep global QEP into a set of interacting distributed QEPs, which if looked at one plan distributed among many mediators would have a bushy instead of linear topology.

Comments

The general idea to re-balance distributed QEPs was suggested by Vanja Josifovski. I designed and implemented a distributed QEP rebalancing algorithm on top of the multi-mediator query compiler of AMOS II. I also designed, implemented and performed the experiments that evaluate the performance improvements resulting from the algorithm and wrote the experimental section of Paper D that describes the experimental results. The rest of the paper was written jointly by Vanja Josifovski, Tore Risch and myself.

7.5 Paper E: Evaluation of Join Strategies for Distributed Mediation

The distributed mediation architecture described in Sect. 3 and Paper A requires that mediators are able to cooperate at the physical level to compute answers of queries over integrated views. One of the most common tasks in data integration is to match overlapping entities in different sources. Since the mediators in the PMS architecture are essentially DBMS, matching of overlapping entities is logically expressed through a join. Join is one of the most expensive operations in a DBMS and therefore much attention has to be paid to its physical implementation. While many join variants have been proposed for centralized and distributed DBMS, a PMS system requires new algorithms that support inter-peer joins between mediators and sources with varying capabilities. Thus the design of join methods for a PMS have to take into account two aspects - efficiency and applicability. This paper proposes and evaluates three distributed join algorithms suitable for the computation of inter-mediator and mediator-source joins in a PMS.

Two ship-out algorithms ship data from a joining mediator towards the sources. In these algorithms, intermediate result tuples are shipped to the sources where they are used as parameters to remote subqueries or function calls. The first algorithm is an order-preserving semi-join, PCA which is suitable when there are no duplicates in the outer collection. The second algorithm, SJMA, uses a temporary hash index of possibly limited size to reduce the number of accesses to the data sources. It is suitable when there are duplicates in the outer collection. Both ship-out algorithms are streamed and the data is shipped between the mediator servers in bulks that contain several
tuples to avoid the message set-up overhead. The third algorithm is a ship-in join, where the data for the inner join operand is shipped from the remote source into the joining mediator.

The ship-out algorithms are applicable to joins with remote sources that need input data to execute local parameterized computations. If these computations are viewed as relations, then the sources are said to have limited capabilities because elements in these relations can not be retrieved by arbitrary attribute(s). To fully implement the algorithms the remote sources must be also able to accept and store locally whole bulks of data and then locally compute over them. The ship-in join algorithm is applicable to joins with remote sources that can ship to a mediator upon request the whole extent of a query or a computation. Such sources may or may not accept parameters. If they accept parameters, then both ship-out and ship-in join algorithms are applicable.

To analyze the performance of the three join algorithms we have fully implemented them in the PMS architecture presented in Paper A. Our performance study shows that the ship-out joins perform better that the ship-in join when: i) early first results are important, ii) joins are performed over slow lines, iii) mediator memory is limited. In particular, the PCA algorithm is simpler to implement, while the SJMA algorithm performs considerably better for outer collections with duplicates. The ship-in join generally performs better when the communication is over a fast network. Finally the ship-out algorithms shift the CPU load to the sources, while the ship-in join puts more of the CPU load on the join mediator.

Comments

The join algorithms described in this paper were proposed and implemented by Vanja Josifovski. I designed and performed the experiments and wrote the experimental section of the paper. Parts of the supporting code for the implementation was done by me together with various improvements necessary to make the implementation complete.

The published version of the paper contains a technical error - in Table 1 and Table 2 the resulting temporary relation tmp has to be inverted together with the final result of the example join.

7.6 Paper F: Object-Oriented Mediator Queries to Internet Search Engines

An important issue in design of a mediation system is its ability to easily incorporate new types of sources. In the mediation architecture presented in Sect. 3 and Paper A, mediators access data sources through wrapper components which interact with the mediator system through its facilities for extensibility - foreign functions, user-defined types and a call-level interface. The
work presented in this paper investigates the flexibility of the extensibility fa-
cilities related to the design and addition of new wrappers. For that, an “exotic”
(from database view point) type of global sources is chosen - Internet search
engines (ISEs). Internet search engines differ from typical database-like data
sources in several ways:

- Their data access interfaces are non-standard, typically requiring program-
natic access to HTML forms.
- Their contents is represented as semi-structured documents without an ex-
plicitly defined schema. The structure of the ISEs’ content differs in struc-
ture among ISEs and even often changes over time for each ISE.
- ISEs do not have a standardized query language.

This requires that a system that accesses ISEs is very flexible. Due to the dy-
namic nature of the ISEs, it should be possible to easily modify and update
existing ISE wrappers, preferably in a dynamic “on-the-fly” manner. Since
the data delivered by ISEs have varying structures the mediator system has
to be able to model the schemata of the ISEs and to reconcile the semantic
differences between them. A large body of work exists that targets the prob-
lem of automatic schema extraction from semi-structured data. That is why a
desirable feature of a wrapper solution for Web sources (as ISEs) is to easily
incorporate new and existing wrapper toolkits that perform automatic schema
extraction.

The paper describes a component of the AMOS II mediator system de-
scribed in Paper A, called ORWISE (Object-Relational Wrapper of Internet
Search Engines) that allows to easily add new ISE wrappers or update existing
ones. Each kind of search engines is modeled as a subtype of the type ISE
under DataSource, described in Paper A. New ISE wrappers are added to a
mediator through the foreign function orwise that is overloaded for each ISE
sub-type. Each implementation of orwise takes a query string in the language
of the particular kind of ISE (e.g. Google) and invokes the wrapper specific for
that kind of ISE through the ORWISE component. The ISE wrapper submits
the ISE query through a low-level wrapper generated by a wrapper toolkit to
the ISE. The data returned by the ISE is then parsed by the low-level wrapper
typically into strings. Finally the ORWISE component semantically enriches
the resulting ISE data by translating it into objects of type DocumentView that
describe Web documents. This enrichment uses routines built-in ORWISE that
map strings into AMOS II types.

In summary, the ORWISE component provides i) the ISE schema for de-
scribing and querying data from any ISE in terms of subtypes of type Data-
Source and the overloaded function orwise, ii) a mechanism to specify search
engine specific translators by redefining orwise and adding new ISE subtypes,
and iii) facilities to allow different wrapper toolkits to be easily plugged into
the system.
The design of ORWISE shows how to include a global data source in the PMS framework. In addition, it shows that the approach to use foreign functions, overloading, and user-defined types to develop new wrappers is indeed very flexible and can easily accommodate even non-database-like global data sources as ISEs.

Comments
The initial idea to wrap ISEs was proposed by myself. I also designed the ORWISE component with discussions with Simon Zürcher. Simon Zürcher implemented and tested ORWISE. The paper was written jointly by me and Tore Risch using as a basis a technical report from Simon Zürcher.
Future Work

The presented mediation architecture poses a wide range of problems to be solved as shown by our analysis of requirements in Sect. 3.2. The fulfillment of each of these requirements is a research area of its own. Here we focus on some future directions that follow directly from the main focus of this work - scalable performance in composable mediators.

Topology-aware heuristics for view expansion
A direct continuation of the work presented in Paper C is to design an efficient heuristic for selective view expansion that utilizes the knowledge of the topology of the logical composition of mediators and targets the view expansion process towards those mediator views that will produce highest increase in QEP quality with the least compilation effort. In our ongoing work we evaluate several such heuristics.

Adaptivity in mediator compositions
Ideally query processing in a PMS should scale up to hundreds and even thousands of mediator peers. In most cases it is impossible to perform precise cost and selectivity estimates when integrating many mediators and diverse data sources over a global network. This may lead to sub-optimal query execution plans. Even if all necessary statistics information is available it is also infeasible to perform full cost-based query optimization in the traditional System R style due the potentially very large number of mediators, sources and views. Our current experience from experiments with mediator compositions of over 20 mediators show that incorrect cost and selectivity estimates can lead to orders of magnitude worse query execution plans (QEP). Several factors specific to peer mediators contribute to the incorrect cost estimates. In most cases it is not possible to acquire statistics about the data stored in the data sources. This is even harder when the data in a source is actually computed and not stored. Imprecise cost modeling may result in that the errors in cost and selectivity estimates increase by orders when propagated through many mediators. Finally data sources, network conditions and mediator load can all change in an unpredictable manner. Therefore it is essential for a mediator system to adapt to an unpredictable and changing environment.

Adaptive query processing for single-site query processors has been addressed by various works [54, 26, 4], to name a few. A good overview of
adaptive query processing can be found in [21, 15]. Many of the proposed approaches can be integrated with the solution proposed here to implement adaptive behavior of each of the mediator peers. However these approaches do not address all the complexity of the problem of adaptivity in a P2P mediator architecture. A centralized query processor usually has direct access to the data structures of a QEP and therefore it has the full power to modify the QEP at any time and adapt its execution accordingly. In a P2P mediator system a QEP is distributed among all peers participating in the evaluation of a query. Because of autonomy, no peer has direct access to the fragments of a global QEP in the other peers. Instead, the query processors of autonomous peers have to cooperate through network protocols in order to change a global QEP and adapt during query processing. Thus adaptivity in P2P mediators requires not only single-site adaptation, but also cooperative adaptation by all participating peers, so that sub-optimal global execution plans can gradually converge to more efficient ones.

Integrated self-profiling

As a basis for adaptivity, mediator systems should be able to measure various parameters of their environment and their own operation and that of neighbor sources and mediators, store this measurements and use them to detect sub-optimality and to adapt by recomputing the affected QEPs.

One approach to measure system performance and manage measurement data is to integrate a database-based profiling system with the query processor of each mediator peer. This will enable the query processor of a mediator to measure parameters related to its own operation, the sources it accesses and the network, and then use the accumulated information for better future decisions. The main idea behind such an integrated profiling approach is to use the mediator system itself in a reflective manner to store all measurement data in the database itself. The benefits of this approach are that the full power of the mediator query language will be available to update, retrieve and analyze the distributed measurement data. Potentially there may be large amount of profile data with dynamically changing distribution across many mediator peers. Using the global query capabilities of the mediator system in a reflective manner to access the profile data would allow to let the system automatically compute the best access path to the data without the need to hard-code it and to easily modify the decision-making procedures inside the optimizer.

With a main-memory mediator database system, such as AMOS II, we can expect very fast updates and retrievals of the measurement data. This will allow to minimize the performance penalty of profiling during normal system usage. The extensibility of AMOS II allows to define custom data structures and functions to store and update profiling data in the most efficient manner while still preserving a query interface to that data. Finally the architecture
of the AMOS II mediator system allows any system component to be profiled in a generic manner. An interesting direction is to profile the operation of all critical components of the query engine and to introduce adaptivity not only at the level of the query execution plans but other system components as well, e.g. the query compiler itself.

The major challenges are how to minimize the performance penalty of profiling, to ensure that the necessary profiling data can be accessed very fast as this will be done from inside the query engine and finally the ability to dynamically control what parameters are being measured.

Adaptive rebalancing of global QEPs

One potentially useful application of the integrated self-profiling is to adapt the distributed data flow of global QEPs. In Paper D we investigated rebalancing of global QEPs that allows the query compiler to generate decentralized plans at each mediator. QEP rebalancing takes a centralized plan where all communication between one mediator and all its direct sub-mediators passes through the controlling mediator and transforms it whenever favorable into a plan with side-wise information passing, where some of the communication is performed directly between the sub-mediators. For this sub-plans of the centralized QEP are sent to the nearest mediators (in terms of logical composition) and further compilation of the sub-plans is delegated to neighbor peers. The peers in turn may further decide to apply rebalancing to the sub-plans received for compilation.

While Paper D shows that distributed QEP rebalancing removes some of the overhead of logical mediator composition, this is done in a static manner. Future work for this project is to extend QEP tree rebalancing to allow mediators to automatically adapt the data flow of distributed QEPs to changes that may occur in a P2P mediator system.

Important research issues related to adaptive QEP rebalancing, and to adaptivity in general are: detecting sub-optimal performance and adapting to it; reuse parts of a QEP when re-adapting to save compilation work; reuse of the intermediate query execution results - if only some of mediators’ plans are reoptimized only the execution of a sub-plan could be restarted instead of recomputing the whole result from scratch.
References


A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series *Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology*. (Prior to October, 1993, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science”.)