

PhD Dissertation



**International Doctorate School in Information and
Communication Technologies**

DIT - University of Trento

**TOWARDS PEER-TO-PEER INFORMATION
MANAGEMENT SYSTEMS**

Ilya Zaihrayeu

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

March 2006

Abstract

Peer-to-Peer (P2P) is a fully distributed communication model in which parties (aka peers) have equivalent functional capabilities in providing each other with data and services. Because they are totally *autonomous*, peers are independent of other peers in which data they store, in which other peers they interoperate with, and in how they process peers' service requests. While the cost of participation for a single peer is limited to seldom answering of peers' requests, the benefit of a peer grows rapidly as more peers take part. In order to reach this effect, peers *coordinate* the processing of the requests by transitively passing data to one another.

The benefits of using P2P as the underlying architecture for building e-commerce and Semantic Web applications has been realized. However, there are still some essential challenges to be addressed. We summarize them in three groups:

- the decentralized open-ended nature of P2P makes it very hard to locate peers which have necessary data, as well as to control the scope of peers' interaction;
- the autonomy of peers implies that no global agreement on data representation languages, on the contents and the location of the data can be made; and,
- partly as a consequence of the first two items, the quality of answers in a P2P network is unpredictable, and it cannot be measured by

standard data quality metrics such as correctness and completeness.

The present thesis is one of the few first dissertation works in the area of P2P information management systems. It provides a detailed account of what P2P information system is and how it relates to the previously developed systems for data and information management. Particularly, we consider conventional database- and ontology-based information systems and show in what respect they are different from P2P ones. We also show how application space is divided among the conventional and P2P information systems. Apart from this, we propose a *novel* information management model, which addresses the first two challenges described above. To address the third challenge we introduce the notion of *good-enough answers* and show in what respect it is different from the standard quality metrics, well studied in the data quality literature. Finally, based on the proposed information management model, we developed a novel query answering algorithm, which supports cyclic topologies and a dynamic system behavior.

Keywords

Peer-to-Peer Information Systems, Databases, Query Answering, Quality of Query Answers

Acknowledgements

I would like to thank my scientific advisor, Prof. Fausto Giunchiglia, for all the efforts he has made to teach me how to go about my research. Particularly, I am thankful for his teaching me how to be deep and precise, and how to learn to think clearly. He has always been patient with me, when I have needed more time, and he has always been supportive and encouraging, when I have needed more inspiration. Without his help, I definitely would not have come this far.

I also thank all my friends who have contributed to this thesis with their support and advice. They have created a friendly and joyful atmosphere, where it has been a great pleasure for me to work.

I am very grateful to two external thesis committee members, Prof. Wolfgang Nejdl and Prof. Paolo Traverso, for the time they have spent in reviewing and nitpicking my thesis.

I eternally thank all my family for their interminable support and understanding. I am particularly thankful to my beloved wife, Olga, without whom it would have been impossible to complete this thesis.

Contents

1	Introduction	1
1.1	The Context	1
1.2	The Problem	3
1.3	The Solution	4
1.4	Contributions	5
1.5	Structure of the Thesis	7
2	Information Management Systems	9
2.1	Basic Terminology	9
2.2	Dimensions for Analysis	11
2.2.1	Autonomy	11
2.2.2	Heterogeneity	12
2.2.3	Distribution	15
2.2.4	Dynamics	16
2.2.5	Scalability	17
2.2.6	Causality among the dimensions	18
2.3	Database Systems	19
2.3.1	Centralized DBSs	20
2.3.2	Distributed DBSs	21
2.3.3	Tightly coupled federated DBSs	22
2.3.4	Loosely coupled federated DBSs	26

2.3.5	Multidatabase language systems	27
2.3.6	Commercial state-of-the-art database systems	28
2.4	Ontology-based Systems	31
2.5	Where and Why	
	Existing Approaches Fail	35
3	P2P Information	
	Management Systems	37
3.1	Definition	38
3.2	A P2P Information Management Model	43
3.2.1	Interest groups	43
3.2.2	Acquaintances	48
3.2.3	Coordination rules	51
3.2.4	Correspondence rules	52
3.3	Principles of a Query Answering Algorithm	53
3.3.1	Query propagation	54
3.3.2	Query result propagation	63
3.3.3	Query answering termination	71
3.4	Good-Enough Answers	74
3.5	System Requirements and Logical Architecture	80
4	Architecture and Implementation	85
4.1	P2P Platform selection: JXTA	86
4.2	Technology Selection: Databases	89
4.3	System Architecture	91
4.4	Implementation	94

4.4.1	Information management model	94
4.4.2	Query answering algorithm	95
4.4.3	Prototype Screenshots and Description	100
4.5	Properties of the Solution	103
5	Evaluation	111
5.1	Experimental Setup	111
5.2	Experimental Results	112
5.3	Conclusions	118
6	Related work	123
6.1	Foundations of P2P Data Management Systems	124
6.2	P2P vs. Conventional Database Systems	126
6.3	Information Management Model	128
6.3.1	Interest groups	128
6.3.2	Acquaintances	131
6.3.3	Coordination rules	134
6.3.4	Correspondence rules	135
6.4	Query Answering Algorithm	136
6.5	Good Enough Answers	139
7	Conclusion	143
	Bibliography	145

List of Tables

2.1	Causalities among the analysis dimensions. Legend: C=“causes”, F=“favours”, R=“requires”. ¹ Becomes a requirement in the presence of dynamics. ² Becomes a requirement in large-scale systems.	18
2.2	A classification of DBSs. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; H = hybrid; NA = not applicable.	20
2.3	A typical profile of an ontology-based IS. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; L = limited.	32
4.1	A classification of information systems. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; H = hybrid; L=limited; NA = not applicable.	109
5.1	P2P IS generator parameters.	113
6.1	A classification of acquaintance links.	132

List of Figures

2.1	Logical architecture of a centralized DBS.	21
2.2	Logical architecture of a DDBS.	22
2.3	Logical architecture of a tightly coupled FDBS. Source: [142].	24
2.4	Five level schema architecture of a (tightly coupled) FDBS. Source: [142].	25
2.5	Schema architecture of a loosely coupled FDBS.	27
2.6	Limitation facet of traditional IMSs.	36
3.1	Principles of locality and transitivity.	40
3.2	P2P IMSs and the limitation facet of traditional IMSs. . .	42
3.3	Interest group hierarchy.	45
3.4	Acquaintance query.	50
3.5	Coordination rule.	51
3.6	Transitivity property of the coordination rules.	52
3.7	Query propagation.	55
3.8	Repeated queries in query propagation.	59
3.9	Query result propagation.	64
3.10	Cycles in a query result propagation graph.	68
3.11	Dynamics in query results propagation.	70
3.12	Query answering termination.	72
3.13	A logical architecture of a P2P IS.	82
4.1	The JXTA software architecture. Source: [111].	87

4.2	A logical architecture of a P2P IS instantiated with JXTA and a DB wrapper.	92
4.3	A logical architecture of the JXTA and P2P IM components.	93
4.4	Query propagation and correlated queries in DB-based P2P ISs.	96
4.5	Multiple repeated loop queries.	97
4.6	Results caching and computation of correlated queries in DB-based ISs.	98
4.7	Running peers.	101
4.8	Discovery of peers on the network.	102
4.9	Setting up acquaintance queries and pipes.	103
4.10	Wrapper configuration.	104
4.11	User query submission.	105
4.12	Query results (at an intermediate peer).	106
4.13	Global update processing.	107
4.14	Topology discovery for a user query.	108
4.15	Network statistics.	109
5.1	Query answering time.	114
5.2	Communication complexity of query answering.	115
5.3	Total new query results produced in the P2P IS by all peers.	116
5.4	Repeated loop queries in the query propagation graphs. . .	117
5.5	Involvement of peers in query answering in the P2P IS consisting of 400 peers.	118
5.6	Average and maximum cache sizes in the P2P IS consisting of 400 peers.	119
5.7	The effects of the result accumulation and of the correlated query computation in the P2P IS consisting of 400 peers. .	120

5.8 Pending messages with and without result accumulation in the P2P IS consisting of 400 peers. 121

5.9 New user query results received from the P2P IS consisting of 400 peers. 121

5.10 Performance evaluation of the query result propagation algorithm. 122

6.1 Schema architectures of a loosely coupled FDBS and a P2P DBS, compared. 127

Chapter 1

Introduction

1.1 The Context

Peer-to-Peer (P2P) is a distributed communication model in which parties (also called peers) have equivalent functional capabilities in providing each other with data and services. Peers come and go, create connections with other peers and, eventually, drop them. Peers are largely autonomous in how to describe data they share and/or services they provide, as well as in which other peers they interoperate with. Compared to the client-server model, P2P offers prospects of higher *autonomy*, *scalability*, and *anonymity*; lower *cost of ownership*; better *fault resilience* and *aggregate performance* [112].

P2P became a buzzword with the appearance and afterwards growing popularity of SETI@Home [7] in 1996 (distributed computing), ICQ [3] in 1996 (instant messaging), Groove [2] in 1997 (collaborative networking) and then Napster [6] in 1999 (music file sharing). Nowadays the range of domains has grown and it also includes Internet telephony [8], and full-fledged file sharing [4, 5]. It is notable that all these applications, to a significant extent, are built on top of P2P architectures. Most successful domains at the moment are file sharing, and instant messaging, whereas the area of Internet telephony is rapidly growing. Hundreds of millions of

users are using the above mentioned and other P2P applications, and this number is constantly growing¹.

Although P2P has proved to be successful in simple but essential end-user applications such as file sharing, its wide adoption in the business settings and government institutions has still to come. However, the potential of new business opportunities brought by the P2P paradigm has been already realized. For instance, the world leading consulting group Gartner reports that “*Beginning in late 2006, the application architecture used to deploy shared business processes for multienterprise supply chain management (SCM) and c-commerce applications will begin to focus on exploiting a peer-to-peer (P2P) computing platform*” [163]. In the same report Gartner makes another statement: “*By 2010, the majority of business processes that cross trading-partner boundaries and provide competitive advantage will exploit shared business processes and leverage P2P computing models (0.7 probability)*”. In fact, in recent years we have been witnessing an increasing bias towards P2P solutions in the areas of knowledge management [34, 109, 52], e-commerce [44], virtual enterprizes [41], enterprise computing [26], and some others.

In c-commerce, SCM and other distributed business applications, parties will not only have to orchestrate workflows and business processes, but also to *share* and *coordinate* data and information, residing locally to the parties *information sources*. This, in turn, will require a special middleware, namely, an *information management system*, which would support the parties in the sharing and coordination of their data.

While it looks as if P2P information management systems (P2P IMSs) have favourable conditions to take its niche on the market of industrially proposed technological solutions, some essential challenges still need to be addressed by the research community.

¹The estimate is based on the number of products’ downloads as reported on the projects’ websites.

1.2 The Problem

P2P IMSs must be able to cope with at least three factors, conditioned by the open-ended autonomous nature of P2P:

Heterogeneity: Even if they model overlapping concepts, data sources will almost always be mutually heterogeneous in the way they describe, store and interpret data. Apart from this, data sources can use different formats to store the data, including databases, XML files, web pages, and structured flat files;

Scalability: The number of data sources can be arbitrarily large. Even in the scope of a single enterprise it can include up to hundreds of sources. If considered in the context of the web, it can count to thousands and thousands of sources;

Dynamics: New sources may become available, existing ones may change their structure, data, or may just become unavailable.

Existing information management systems (IMSs) are viewed mainly as *integration* solutions, whose underlying idea is to represent a set of sources in terms of a single integrated conceptual model, and provide centralized access and management facilities to these sources. Particularly, we refer here to the well studied field of database systems [154], and to relatively new approaches based on the use of ontologies [145].

The main drawback of the integration approaches is that they become impractical, and even inapplicable in the P2P settings. Namely, the factor of heterogeneity leads to the need of finding a consensus in information representation at the *global* level, which is far from a trivial task as such. The factor of scalability dramatically multiplies the integration effort. Finally, the factor of dynamics leads to the need of reintegrating everything almost from scratch each time a change is introduced into the system [28].

There are some other approaches that do not rely on the notion of a

global conceptual model, but which still possess a significant level of centralization. For instance, loosely coupled federated database systems are operated without a global schema, but require central database administration [142, 104]. The ontology-based systems described in [110, 127] interoperate without being integrated through a global ontology, but rely on the notion of a shared ontology, which ensures interoperability. These approaches are not scalable for similar reasons as discussed above.

Thus, existing integration-based solutions are inappropriate in the P2P settings. Therefore, *new methodologies, theories, mechanisms and technologies need to be developed that would enable data sources to effectively interoperate in the P2P settings.* Despite the factors of heterogeneity and dynamics, P2P IMSs must ensure high quality of answers, robustness and reliability in the presence of transient large-scale peer federations.

1.3 The Solution

In this thesis we propose a solution to the problem of the management of data sources in the P2P settings. Namely, we propose a new approach, which we call *data coordination*. Its fundamental assumption and the main difference from the existing approaches is that peers interoperate by means of *coordination* of their data sources with each other, and not by means of *integration* as it is done in most of the existing approaches. Particularly, peers define *pairwise* mappings, which relate corresponding elements in representations of their data sources. They then use these mappings in order to *transitively* propagate data and information to one another.

There is no central point of access to the information system, where users would submit their queries. There is no central authority, which would define mappings between peers, or which would have a global view of the whole system. Finally, there is no central control, which would define

how queries are propagated and answered in the system. Instead, queries are submitted locally to peers; each peer locally decides which other peers to build mappings with, and which other peers to propagate a query to; finally, peers have only a partial view of the whole system.

We define a novel P2P information management model, which is built on top of four fundamental notions. They are: *peer groups*, *acquaintances*, *coordination rules*, and *correspondence rules*. In short, peer groups comprise peers, which share a common interest. Acquaintances are peers, a peer has established mappings with. Coordination rules define mechanisms for the propagation of requests and data between peers. Finally, correspondence rules define how elements, belonging to the domain of one peer, are translated into elements of the domain of an acquaintance peer.

We introduce a novel notion of *good-enough answers*, which reflects the fact that in the P2P settings, answers to user requests in most cases are not complete and correct, as they usually are in the integration approaches. This notion is crucial for measuring the quality of service provided by P2P IMSs.

Apart from this, we propose a novel query answering algorithm that allows it to define a query at a peer, and to answer it in a completely decentralized fashion through transitive propagation of queries and query results in the system. In this process, multiple peers contribute to the answer of a local query. The algorithm supports topologies with cycles, and it supports dynamic network behavior at the runtime of a query. It allows the querying peer to determine when query answering is actually complete.

1.4 Contributions

The present thesis makes the following contributions:

- it gives a definition of a Peer-to-Peer Information Management System. The definition has a scientific value because, differently from the previous definitions, it unambiguously discriminates P2P IMSs from other types of the existing data and information management systems;
- it compares P2P IMSs with the closest ancestors in the family of database management systems and explicitly shows in what respect they are different;
- it explicitly identifies the principles of locality and transitivity and shows their principal role in the architectures of P2P information management systems. The two principles are fundamental as they allow it to unambiguously differentiate between P2P and conventional IMSs;
- it proposes a novel query answering algorithm for P2P ISs, which is capable of processing multiple cycles in the topology of connections of the system components of a P2P IS;
- it shows why standard data quality metrics cannot be used to measure the quality of query answers in a P2P IS. To address this problem, it presents and extends the notion of good-enough answers in P2P ISs;
- it presents a developed prototype of a P2P IMS and its first evaluation results, which demonstrate the proof-of-concept of P2P IMSs.

Part of the material of the thesis has already appeared in the following co-authored papers (in order of appearance): [31, 68, 55, 57, 71, 58, 56, 69, 70, 34, 61, 166, 62]. Whenever results of any of these papers are reported, proper citations are made in the body of the thesis.

1.5 Structure of the Thesis

The thesis is organized as follows: in Chapter 2 we discuss the state-of-the-art conventional information management systems. In Section 2.1 we introduce some basic terminological terms related to the IMSs. In Section 2.2 we introduce five dimensions for our analysis of the IMSs. In Section 2.3 we discuss conventional database systems, and in Section 2.4 we discuss ontology-based systems. In both cases we consider the IMSs in the light of the proposed dimensions. In Section 2.5 we show where the existing IMSs have certain limitations, critical to a new class of emerging applications.

In Chapter 3 we introduce and discuss P2P IMSs. In Section 3.1 we provide a definition of P2P IMSs, which discriminates them from the conventional IMSs. In Section 3.2 we introduce a P2P information management model and discuss its core notions. In Section 3.3 we propose a generic query answering algorithm that can be implemented in database-based P2P ISs and, potentially, in ontology-based P2P ISs. In Section 3.4 we discuss the novel notion of good-enough answers. Finally, in Section 3.5 we discuss the system requirements and a logical architecture of a P2P IMS.

In Chapter 4 we provide the details of the implemented prototype of a P2P IMS. In Section 4.1 we motivate our choice of a P2P platform, on top of which we build our IMS. In Section 4.2 we motivate our choice of the technology, used to implement the data management in the P2P IMS. In Section 4.3 we discuss the system architecture of the implemented P2P IMS. In Section 4.4 we discuss the details of the implementation, provide some screenshots and a description of the implemented prototype. In Section 4.5 we discuss the properties of the proposed solution in the light of the analysis dimensions introduced in Section 2.2.

In Chapter 5 we provide an evaluation of the proposed approach. In Section 5.1 we discuss the settings in which we ran our experiments. In Section 5.2 we report the results of the conducted experiments. In Section 5.3 we summarize the results of the experiments and make our conclusions.

In Chapter 6 we provide a detailed analysis of the related work. In Section 6.1 we discuss how the proposed definition of a P2P IMS is related to the ones proposed in the literature. In Section 6.2 we show in what respect P2P IMSs are different from their closest ancestors from the family of database systems. In Section 6.3 we discuss the related work for each of the core notions of the proposed P2P information management model. In Section 6.4 we relate the proposed query answering algorithm to other similar algorithms proposed in the literature. Finally, in Section 6.5 we discuss what has been done in the field of data quality for P2P systems and we relate the notion of good-enough answers to the state-of-the-art in the field.

In Chapter 7 we provide concluding remarks and highlight the directions of future work.

Chapter 2

Information Management Systems

In this chapter we define the notion of information system (IS) and one of information management system (IMS). We then discuss some of the state-of-the-art conventional IMSs, their advantages and limitations. This chapter serves as the basis for our further comparative analysis of these systems with the new kind of IMSs proposed in this thesis.

The chapter structure is as follows. In Section 2.1 we introduce some basic terminological terms related to the IMSs. In Section 2.2 we introduce five dimensions for our analysis of the IMSs. In Section 2.3 we discuss conventional database systems, and in Section 2.4 we discuss ontology-based systems. In both cases we consider the IMSs in the light of the proposed dimensions. In Section 2.5 we show where the existing IMSs have certain limitations, critical to a new class of emerging applications.

2.1 Basic Terminology

The evolutionary line of information systems shows a clear trend from raw data- to information-, and knowledge-centric systems [141]. In other words, and from the IT perspective, *raw data* are symbols with no meaning.

2.1. BASIC TERMINOLOGY

Information is interpreted raw data, i.e. it is raw data with a meaning given. The notion of *knowledge* is connected to the ability of relating pieces of information, such that it forms the basis for new information and knowledge to be produced [146]. For example, `cobia` is a raw data instance since, as a set of symbols, it carries no meaning; whereas `cobia is a fish` becomes information; The ability to combine this piece of information with another piece, e.g. `fish can swim`, is an example of knowledge, based on which one can deduct a new piece of information, e.g. `cobia can swim`. Hereafter we will write *data* to refer to a collection of raw data, information and knowledge.

We define an *information system* (IS) as a combination of hardware, software and telecommunication networks that is used to collect, store, and transmit data. An *information management system* (IMS) is the software that permits the management of an IS, and particularly provides for the acquisition, manipulation, control, and display of data. In an IS, data are stored in the form of files, database records, and other data repositories, which we call *local information sources* (LISs). From the operational perspective, the main function of a IMS is the management of a LIS or a set of LISs.

Usually, a LIS consists of two conceptual parts: data and *metadata*, which describe the data. Metadata may take different forms: in a database system it may include information about base relations and integrity constraints; in a file system it may include such information as file names, timestamps and directory paths; in a knowledge management system it may be represented in the form of *ontology* [75]. When structured, metadata are often called a *schema*. In the following we assume that metadata of a LIS is a schema, and we call it, the *local source schema* (LSS). Since LSSs are schematic, queries, submitted to the LISs of an IS, are also schematic (as opposed to queries represented, for example, as a set of keywords).

We refer to Quality of Service (QoS) of an IS as to the ability of the system to locate and deliver necessary pieces of data, transmit data preserving their semantics, and perform other operations requested by the user, in an adequately timeled manner.

2.2 Dimensions for Analysis

Among the variety of IMSs we are interested in those, which permit the management of multiple LISs. These systems can be characterized along several dimensions, such as *autonomy*, *heterogeneity*, *distribution*, *dynamics*, and *scalability*. In this section we provide a detailed account of what these dimensions are, and how they characterize an IMS and its LISs.

2.2.1 Autonomy

Autonomy became an important issue with the emergence of distributed systems [114]. It obviously has different shades of interpretation depending on the context. For the domain of information systems, following the considerations made in [141, 51], we define the following five forms of autonomy:

(i) Design autonomy: Parties are free to choose their own data model (e.g. Entity-Relationship model, relational model), query language (e.g. SQL, XQuery [53]), functionality (i.e. the operations supported by the system), and implementation (e.g. record and file structures, concurrency control algorithms). They are free to decide what data to store, how to describe the data, what constraints to use on the data, and what interpretation to associate with the data;

(ii) Communication autonomy: Parties decide which other parties to communicate with, when and how to respond to requests coming from them;

(iii) Execution autonomy: Parties are solely in charge of how to execute operations on local data; they do not need to inform any other parties of what operations and in what order are executed. Execution autonomy implies that a party can outright abort any request coming from outside and which, for instance, violates a local constraint;

(iv) Association autonomy: Parties decide how much of their functionality and data to share with other parties;

(v) Participation autonomy: Parties decide when to join and when to leave the system, which logically means that they decide when to communicate and share their resources with other parties. Thus, for instance, no participation for a party means no communication with other parties (as it may be the case with full communication autonomy) and no sharing of its resources (as it may be the case with full association autonomy). Apart from this, participation autonomy also implies that parties are free to connect to the system from any location.

Though autonomy is considered to be a desirable property, it does not come free. It may have negative consequences effecting the QoS, such as: query answers may be incomplete and mutually inconsistent; the same data may be duplicated in the system with excessive or insufficient redundancy; and, queries may be answered in an unsatisfactorily timed manner. Readers, interested in a comprehensive overview of the subject of the cost of autonomy in distributed systems are referred to [114].

Hereafter we refer to *total* or *full* autonomy as meaning that the parties in the context possess all the forms of autonomy discussed above.

2.2.2 Heterogeneity

Heterogeneity refers to the possibility of representing the same concept in many different ways due to differences in perspectives, expressiveness of data models, and so on [28]. When parties are given significant or full

design autonomy, a natural consequence of this is that they will almost always be mutually heterogeneous in the way they organize, represent, and store data. Heterogeneity takes several forms, which we classify (based on an aggregated analysis of [72, 99, 35]) in the following levels:

(i) Data model level: It refers to differences in data models, query languages, and constraints. For instance, one LIS may use the relational data model and SQL as the data manipulation language, another may have RDF [9] as the data model and SPARQL [11] as the query language. Even two LISs exploiting the same data model may differ in the query languages (e.g. different versions of SQL). Apart from this, LISs can differ in what mechanism they have for the specification of constraints on data (e.g. triggers vs. referential integrity constraints);

(ii) Schema level: It refers to differences in modelling (even with the same data model) of the same or related concepts. It comprises the following cases:

- *Labelling conflicts*, which refer to the situation when the same concept is referenced using different labels in distinct LISs (synonyms); or when the same label denotes semantically different concepts (homonyms);
- *Data type conflicts*, which refer to the situation when the same concept is associated with different data types in distinct LISs (e.g. `Date` vs. `String`);
- *Schema structure conflicts*, which refer to the situation when the same entity is modelled using different (sets of) constructs of the data model. In fact, there are many different ways to represent the same concept [96]. For instance, in the relational model, the same concept can be represented as a relation, as an attribute, or even as an attribute value; or the same concept can be represented as relations

with different (but overlapping) sets of attributes; or it can be represented at different levels of abstraction [144];

- *Integrity constraint conflicts*, which refer to disparities in the definition of integrity constraints on data. For instance, in the relational data model, two tables in one database can be linked by means of a foreign key constraint, and a similar constraint may not be specified for the identical tables in another database.

(iii) Data level: It refers to differences in representing identical or related raw data values. It comprises the following situations:

- *Naming conflicts*, which refer to synonyms and homonyms among *raw data values*. For instance, “*UNITN*” and “*University of Trento*” are synonymic values; and data value “2000” for concept *year* has a different meaning if it appears as a data value for concept *salary*;
- *Scaling conflicts*, which refer to the situation when the same raw data value is stored using different units of measure. For instance, values for concept *cost* may be stored in US dollars in one LIS, and in Euros in another;
- *Precision conflicts*, which refer to the situation when a value of a concept in one LIS corresponds to a range (or a set) of values for the same concept in another LIS. For instance, product delivery may be measured in days in one LIS, and in weeks in another.

(iv) System level: It refers to differences in hardware, system software, and communication protocols. Although not directly related to the notion of data as such, this kind of heterogeneity is growing in importance due to the emergence of pervasive computing [137].

In the literature item (i) is often called *schematic heterogeneity*, and the union of items (ii) and (iii) is usually titled as *semantic heterogeneity* [72].

In most cases, the difference is made between structure (“how are the data logically organized?”) and interpretation (“what do the data mean?”).

Semantic heterogeneity can be dealt with by finding and analyzing heterogeneity conflicts using the structure of and semantics encoded in LSSs (see [72, 94, 85, 64] for various approaches to the problem of semantic heterogeneity). However, there may be heterogeneity conflicts not caused by disparities in the schema structures. Namely, the way the schema is used by the application defines its precise meaning, which can be different in different applications [81]. For instance, concept **address** can be treated as the street address in one application, and as the complete address in another application. We refer to this kind of heterogeneity as to *application level heterogeneity*.

For the purpose of completeness, we must say that there are other forms of heterogeneity discussed in the literature, which are outside the scope of this thesis and, therefore, are not considered in detail. There is the so-called intentional heterogeneity concerned with domain conflicts [72], heterogeneity concerned with the fact that data may be incorrect or may become obsolete [99, 72], heterogeneity concerned with concurrency control implementation [89], and some others.

2.2.3 Distribution

The third dimension we consider important for ISs is *distribution*. This term has been used equivocally in the literature – it has been given different definitions in different sources. We define distribution along three levels:

Physical distribution: An IS is *physically distributed* if data are distributed among multiple LISs, and it is *physically centralized* if data are stored in a single LIS. Under the term *distributed* we mean that the LISs reside on different computational sites, i.e. which are geographically distributed, but interconnected through a network;

Logical distribution: An IS is *logically centralized* if data residing in its LIS(s) are described by means of a single global schema, and the data are accessed through this schema; and it is *logically distributed* if each LIS has its own LSS, and there is no global schema. If both the global schema and LSSs are present, then system has a *hybrid logical distribution*;

Operational distribution: An IS is *operationally centralized* if it maintains a shared global register or index of its resources, such as IP addresses of LISs, their LSSs, mappings between LSSs; and uses this information to enable the operation of the system (e.g. it uses it for the computation of a query plan). Apart from this, an IS is operationally centralized if there exists a central authority (e.g. a system administrator), which has a global view of the system and which defines how system components need to be interconnected. An IS is *operationally distributed* if LISs store *locally* a (partial) index of the resources in the system, and there is no central authority.

Note that an IS must be physically distributed to allow for logical distribution, and it must be logically distributed to allow for operational distribution. We say that an IS is *fully distributed*, if it is distributed in all the three levels.

2.2.4 Dynamics

If an IS possesses design, association, and participation autonomy, then it is a subject of *dynamics*, which may become apparent to a greater or lesser extent depending on a particular application.

Design autonomy forces us to assume that LISs may change over time their LSSs, data model, functional capabilities, etc. Apart from this, LISs may change treatment of domains in LSSs, even without changing the actual schema [157], thus contributing to the application level heterogeneity. Association autonomy forces us to assume that data and/or functions

shared by the LISs may change over time. Finally, participation autonomy implies that new LISs may join the system, and existing ones may eventually leave it.

Dynamics may have a negative effect on the QoS provided by the IS. Thus, whenever a change is introduced, the system must take an action (either centrally or locally) in order to compensate for this change and maintain the QoS. For instance, if a LIS stops sharing (a part of) its data, then those LISs which used these data have to find other source(s) which would compensate the loss. Analogously, if a LIS changes its LSS then other parties have to adjust their import settings with this LIS, or find other data providers.

In the above and many other examples of dynamics, in order to maintain the QoS, the system will have to deal with two major problems: *resource discovery* and *heterogeneity resolution*. Depending on the distribution characteristic of the IS, these two problems can represent different levels of difficulty. For example, in an operationally centralized system, resource discovery is likely to be easier than it is in an operationally distributed system. In a logically centralized system, the problem of heterogeneity is something harder (as the global schema may need to be reconsidered) if compared to the logically distributed case (as only some local mappings may need to be reconsidered).

We say that an IS is *dynamic* if it provides proper mechanisms and tools to support dynamics without having a significant negative affect on QoS; and it is *static* otherwise.

2.2.5 Scalability

Participation autonomy implies that an IS can include an arbitrarily large number of LISs. A larger IS can potentially provide its users with more data, but it also leads to new problems. Namely, a larger system means a

2.2. DIMENSIONS FOR ANALYSIS

Autonomy	Heterogeneity	Distribution			Dynamics	Scalability
		physical	logical	operat.		
design	C	C	F ¹		R	
comm.		C	F	F		
exec.		C	F			
assos.		C			R	
partic.		C	F ¹	F ²	R	R

Table 2.1: Causalities among the analysis dimensions. Legend: C=“causes”, F=“favours”, R=“requires”. ¹Becomes a requirement in the presence of dynamics. ²Becomes a requirement in large-scale systems.

more challenging resource discovery if the system is operationally distributed. It leads to higher computational and storage requirements for the central component if the system is logically centralized, and to a harder global integration if, in addition, it is heterogeneous.

We say, that an IS is *scalable* if, with the growth of the number of LISs, it is capable of maintaining or improving the level of QoS; and it is *unscalable* otherwise.

2.2.6 Causality among the dimensions

As it can be seen, almost all the analysis dimensions depend to a large extent on various forms of autonomy. Here, we summarize the causalities among autonomy and the other dimensions (see Table 2.1). The table should be read as follows. In the first column we list the different forms of autonomy, and in the other columns we show how they affect the other dimensions. For instance, design autonomy *causes* heterogeneity. Below, we discuss this and other interrelationships in detail.

Design autonomy stipulates that different LISs are developed differently, and it therefore causes heterogeneity. The notion of autonomy, as such, conditions physical distribution, as it assumes the presence of multiple

LISs. Design and participation forms of autonomy make the task of global integration arbitrarily hard, and, therefore, they favour logical distribution. Finally, in the presence of dynamics these two factors make logical distribution a requirement. Apart from this, logical centralization stipulates that parties are committed to communicate through a single component, and execute requests coming from it, which violates the communication and execution forms of autonomy. Therefore, these two forms of autonomy also favour logical distribution.

Participation autonomy makes it impractical to maintain a global index of resources as it needs to be updated each time a party enters or leaves the system. Therefore, participation autonomy favours operational distribution, and it makes it a requirement in large-scale systems. Communication autonomy diminishes the role of a central authority and, therefore, it also favours operational distribution.

Design, association, and participation forms of autonomy require the IS to be dynamic. Finally, participation autonomy requires it to be scalable.

2.3 Database Systems

A *database system* (DBS) consists of software, called a *database management system* (DBMS), and a database (DB) that it manages. A *multi-database system* (multi-DBS) consists of several logically interconnected and/or integrated database systems. These DBSs are called, in the context of a multi-database system, *component DBSs*.

DBSs are likely to be the most widely used technology for the implementation of LISs. There is a variety of DBSs, which differ in whether they support a single or multiple databases, in whether the databases can be managed by different software, and so on. There are several types of data models supported by DBMSs, such as relational, object-oriented, or

2.3. DATABASE SYSTEMS

DBS	Autonomy					Heterogeneity			Distribution			Dyn.	Sc.
	D.	C.	Ex.	As.	P.	D. M.	S. L.	D. L.	P.	L.	Op.		
C. DBS	Y	NA	Y	NA	NA	NA	NA	NA	C	C	C	Y	NA
DDBS	N	N	N	N	N	N	NA	NA	D	C	C	N	N
T.C. FDBS	Y	N	N	Y	N	Y	Y	Y	D	H	C	N	N
L.C. FDBS	Y	Y	N	Y	N	Y	Y	Y	D	D	C	Y	N
MDBS	Y	Y	N	Y	N	Y	Y	Y	D	D	D	Y	N

Table 2.2: A classification of DBSs. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; H = hybrid; NA = not applicable.

hierarchical.

In the rest of this section we discuss the family of relational database systems. Namely, we first discuss a single DBS, called *centralized DBS* (Section 2.3.1), and four multi-DBSs, namely *distributed DBSs* (Section 2.3.2), *tightly-coupled federated DBSs* (Section 2.3.3), *loosely-coupled federated DBSs* (Section 2.3.4), and *multidatabase language systems* (Section 2.3.5). We discuss these DBSs in the light of the dimensions presented in the previous section, and we summarize our findings in Table 2.2. Finally, we provide a brief discussion of a typical state-of-the-art commercial database system (Section 2.3.6).

2.3.1 Centralized DBSs

A centralized DBS consists of a single DBMS and a database that it manages (see Figure 2.1). As it is a sole DBS, it possess total design and execution autonomy; it is logically, physically and operationally centralized; significant dynamics can be afforded since there are no other parties with which there might be restricting agreements, for instance, on the database schema. The notion of heterogeneity is not applicable, as there are no other DBS with respect to which the centralized DBS would be heterogeneous.

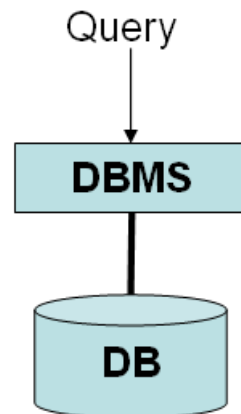


Figure 2.1: Logical architecture of a centralized DBS.

2.3.2 Distributed DBSs

A *Distributed Database System* (DDBS) is a collection of multiple, logically integrated databases distributed over a computer network [126]. As from [126], a distributed database management system (DDBMS) is “*the software system that permits the management of the DDBS and makes the distribution transparent to the users*”. As the definition indicates, DDBSs are physically distributed.

The architecture of a DDBS (see Figure 2.2) does not address the problem of integration of *preexisted* DBSs. In fact, a DDBS is usually designed centrally by the DDBS administrator, who is responsible for schema design of and data placement in the component DBSs. Therefore, the notion of semantic heterogeneity is not applicable for DDBSs. All the component DBSs in a DDBS usually exploit the same data model [51].

The component schemata are integrated into a single conceptual schema which describes *all* the data residing in the component DBs. This schema is often called the *global schema*. Users and application programs use the global schema to access and manipulate (through the DDBMS) the data residing on the component DBs (i.e. it is logically centralized). The man-

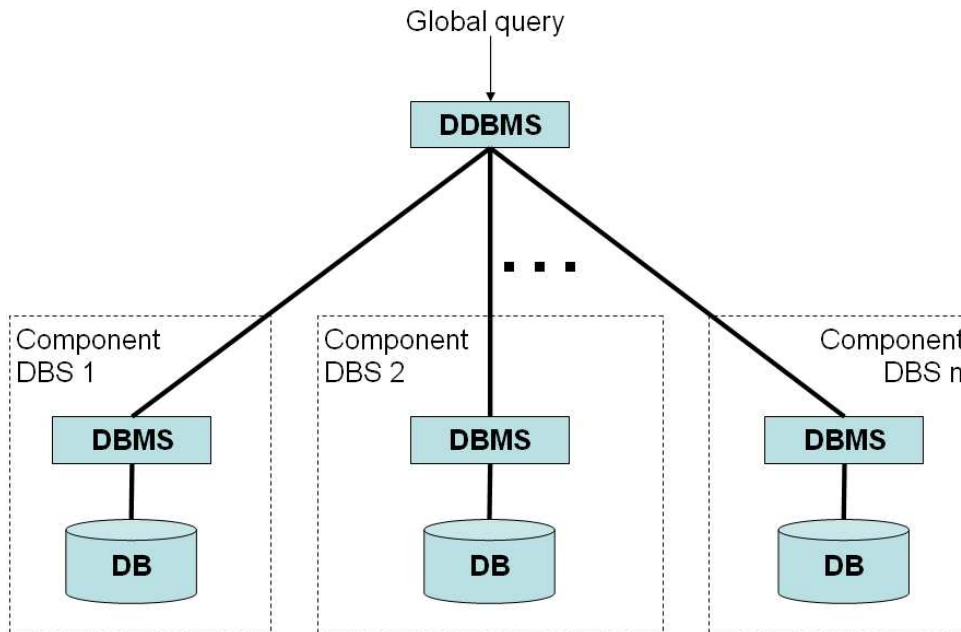


Figure 2.2: Logical architecture of a DDBS.

agement of the DDBS is performed centrally by the DDBS administrator (i.e. it is operationally centralized). As a consequence, component DBSs possess no autonomy.

DDBSs provide no support for dynamics. The DDBS constitution is fixed and changing the schema of a component database may require reconstruction of the global schema from scratch, and this can be a very expensive operation [28]. Usually, DDBSs integrate a small number of component DBSs, and, therefore, we consider DDBSs as not scalable.

2.3.3 Tightly coupled federated DBSs

A *Federated Database System* (FDBS) allows for higher autonomy of the component DBSs and for less global integration [88, 142, 104, 51]. The software that provides controlled and coordinated manipulation of the component DBSs is called a *federated database management system* (FDBMS).

In the literature, two types of FDBSs are distinguished: *loosely coupled*

and *tightly coupled* (see, for instance, [142, 51]). The difference is in the quantity of integrated schemata, which are called in the context of FDBSs, *federated schemata*; and in how the FDBS is managed. In this section we discuss tightly coupled FDBSs and in the next section we discuss loosely coupled FDBSs.

A tightly coupled FDBS usually has only one federated schema, and the federation is managed centrally by a FDBS administrator. The data in a FDBS can be accessed either globally (through the federated schema) or locally at each component DBS. Therefore, FDBSs have a hybrid logical distribution, and they are operationally centralized.

Yet another factor favours operational centralization. Namely, a FDBS maintains the so-called *federated dictionary* [88], which, in more recent publications, is called *data dictionary* [51]. The federated dictionary is a distinguished component that maintains the topology information of the federation and oversees the entry of new databases. It supports the establishment, maintenance, and termination of the federation. Apart from this, it stores database schemata, mappings among the schemata, network addresses of the federation databases, communication facilities of each database site, and so on.

The FDBS architecture does address the issue of integration of *preexisted* DBSs. Therefore, design autonomy is respected and, consequently, component DBSs can be heterogeneous at all the levels [142]¹. Apart from this, component DBSs can be of different type. For instance, a component DBS can be a centralized DBS, a DDBS, or even another FDBS (see Figure 2.3).

The amount of integration does not have to be complete as in the case of DDBSs and the DBSs decide how much to share, which means more

¹In [88, 51] the authors assume that all the DBSs in a FDBS share the same data model and query language.

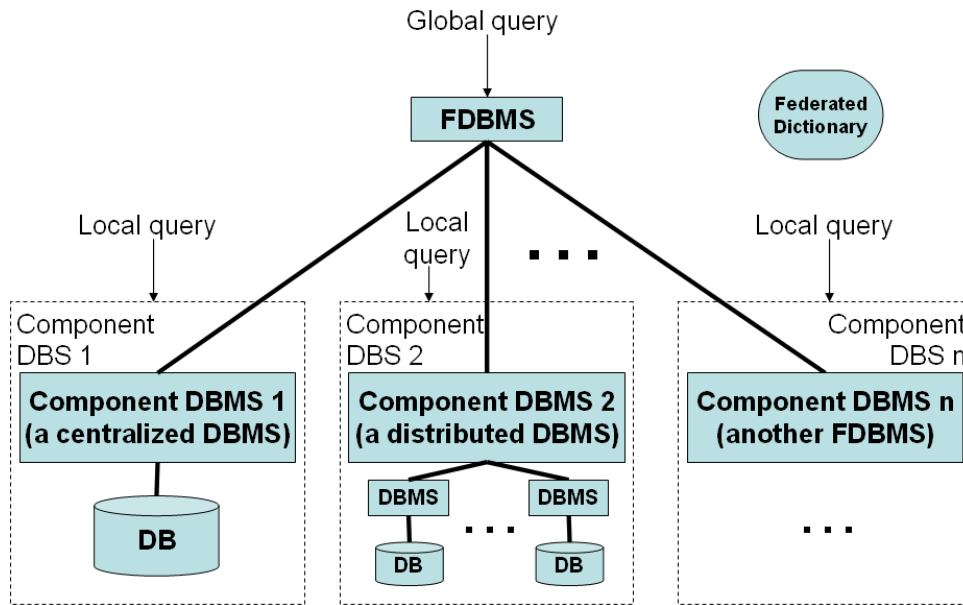


Figure 2.3: Logical architecture of a tightly coupled FDBS. Source: [142].

association autonomy. However, by participating in a FDBS, component DBSs agree to cooperate with others in executing user requests that access multiple DBSs [126]. This means no execution autonomy. FDBSs are static for the same reason as DDBSs are. Namely, the need of reconsideration of the federated schema makes any kind of changes very undesirable. For the same reason FDBSs are not scalable – with large number of preexisted component DBSs, the task of global integration becomes infeasible.

Consider Figure 2.4, where the schema level architecture of a FDBS is reported². It consists of five schemata [142]:

- *Local Schema*: the conceptual schema of the component database, expressed in the data model of the component DBMSs;
- *Component Schema*: a local schema, translated to the data model of the FDBS. It solves the data model heterogeneity problem;

²We provide these details as they are relevant to our further discussion in the thesis.

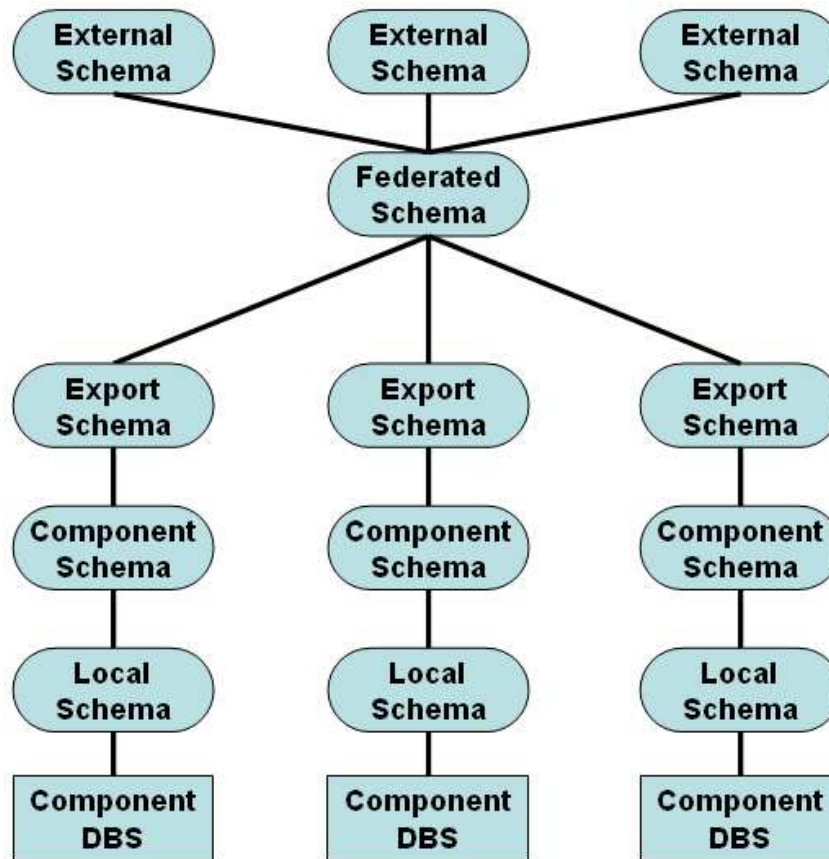


Figure 2.4: Five level schema architecture of a (tightly coupled) FDBS. Source: [142].

- *Export Schema*: a shared part of a component schema. Each database maintains access control information (i.e. it defines what other databases may access its export schemata). Thus, association autonomy is maintained;
- *Federated Schema*: a federated schema is an integrated view of multiple export schemata;
- *External Schema*: If the federated schema is large and complex, then a set of external schemata can be defined for certain class of users and/or applications.

Queries are posed against a federated (or external) schema, and are then

decomposed and rewritten into queries against the local schemata of the corresponding component databases.

2.3.4 Loosely coupled federated DBSs

In the loosely coupled case, the owners of the component DBSs are responsible for creating and maintaining their own federated schema(s), and there is no central administrator. There is no global federated schema either, i.e. the system is logically distributed.

Component DBSs independently define what other component DBSs to use for the construction of a federated schema (see Figure 2.5). This means more communication autonomy. However, users need to have a global view of the system, so that they can potentially access (through the construction of federated schemata) *all* the relevant component DBSs in the federation. To make it easier to find relevant component DBSs, the data dictionary is still maintained. This means that loosely coupled FDBSs are still operationally centralized.

Loosely coupled FDBSs are relatively dynamic as they allow for federated schemata to be created and dropped on the fly [51]. These systems are not scalable for two reasons: first, they are still operationally centralized. Second, as mentioned above, the underlying assumption of this approach is that users still possess a global view of the system, and this allows it to interconnect all the component DBSs which need to be interconnected. This assumption is fundamental to the system design of a loosely coupled FDBS. While it performs well when the number of component DBSs is relatively small, this approach fails when their number grows larger.

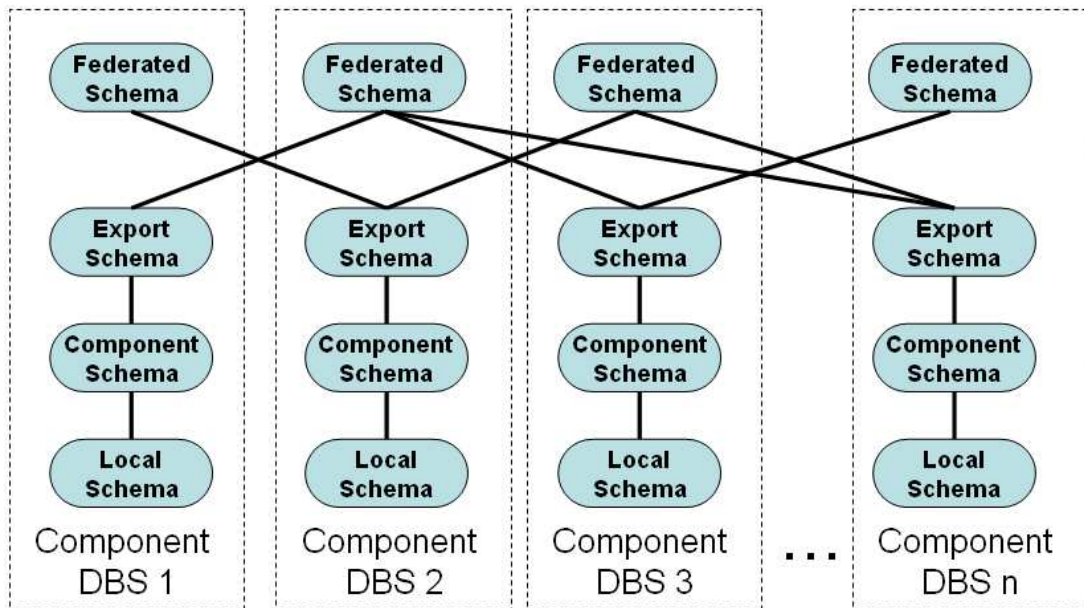


Figure 2.5: Schema architecture of a loosely coupled FDBS.

2.3.5 Multidatabase language systems

The main idea behind multidatabase language systems is to allow for data manipulation operations that can refer to more than one database at a time [103, 102, 36]. It requires a multidatabase manipulation language which goes beyond standard SQL, and whose constructs allow it to refer to multiple databases in a single language statement. One of such languages is MSQL [102], which was later extended to MSQL+ [113].

There is no global or federated schema in a multidatabase language system. Instead of posing their queries against federated schemata, users specify the databases, they are interested to retrieve (and combine) data from, in the syntax of the multidatabase language. There is no such notion as federated dictionary, what favours operational distribution of this approach. However, similarly to the case of loosely coupled FDBSs, users need to have a solid idea about what information is required and where it probably resides [36]. This fact restrains the scalability of multidatabase

language systems.

2.3.6 Commercial state-of-the-art database systems

There are at least three main goals, which largely motivated research in the database field in the past, and which determine the architecture of modern industrial database systems: *transparency*, *data consistency*, and *performance* [126]. Transparency means that the details of the different databases and their location, data distribution, schema heterogeneity, and processing logic are hidden from users and applications. Users work with a database system as if it were a single centralized homogeneous database. Intuitively, data consistency refers to the state of the database system in which data, possibly residing on multiple heterogeneous databases, are logically coherent and there is no conflicting data representation. Performance is usually measured in the number of atomic operations a database system is capable of performing in a unit of time. The more operations it can perform, the better.

All the three goals are (much) easier to achieve within a tight statically integrated architecture of the database system rather than within a loosely coupled one. In fact, the global schema hides the details of the underlying databases and data placement in these databases, and this partly guarantees transparency. The main instrument to ensure data consistency is a (distributed) transaction management mechanism, which requires tight and coordinated synchronization of the involved databases. Finally, the possibility of central planning of data allocation and, consequently, of query execution plan optimization, allows it to achieve a better performance of the database system.

In accordance with the above arguments, loosely coupled FDBS and multidatabase language systems received very little attention in the industrial arena. In fact, these systems provide very limited support for

transactions, data placement is not transparent, and they have inferior performance indicators than centralized data integration solutions. Therefore, the prevailing part of multi-database systems are built on top of the DDBS and tightly coupled FDBS architectures. In the following we provide some examples of architectural particularities of three leading multi-database systems: SQL Server 2000 [73] (Microsoft corp.), Oracle Real Application Clusters 10g [106] (Oracle corp.), and DB2 UDB v8.2 [12] (IBM corp.).

Scalability for modern multi-database systems has not been seen as the problem of adding new databases, but mainly as the problem of increasing the performance. For instance, a database system built on top of SQL Server 2000 can support “only” up to 64 nodes, and can process more than one billion transactions per day [73]. The performance can be increased in two ways: by *scaling-up* and by *scaling-out*. Scaling-up is increasing resources on a single machine (e.g. adding CPUs, disks, memory). When scaling-up becomes technically difficult and/or financially inefficient, the performance is increased by scaling-out, i.e. by adding machines, still integrating the resources into one global logical database [73]. Thus, the trade-off in the design of modern database system is in achieving the highest performance with the smallest number of databases, and at the lowest possible cost.

As a means to increase performance, SQL Server 2000 supports *vertical* and *horizontal partitioning* of database tables. Vertical partitioning is splitting one table with many columns into several tables with few columns; and horizontal partitioning is splitting one table with many rows into many tables with few rows. If these tables are distributed among several servers, then it is called a *federated database servers configuration*; and, the distributed data are then manipulated using special language statements, which are called *distributed partitioned views*. The performance gain from using the federated database servers architecture normally goes from 20% to

30% [139].

Oracle overcomes some of the shortcomings of the federated database servers architecture by implementing the *shared-disk* cluster architecture [106]. In this approach all the servers have access to all the disks in the system. It allows it to reach better performance and fault tolerance. Apart from this, adding new servers is easier than in the federated servers approach. However, due to the need to synchronize access to the shared disks from multiple servers, the scalability of this approach largely depends on correct data placement [37].

DB2 UDB v8.2 is built using the *shared-nothing* architecture, in which each server can access only its own disk. However, the DB2 database supports data partitioning among multiple servers³ [12]. Similarly to SQL Server, its scalability is largely dependent on accurate data partitioning among the servers' disks. The main difference from SQL Server is in that DB2 UDB is seen as a single database with a single data dictionary, and not as a collection of tightly integrated distributed databases as it is the case for SQL Server [37]. The main difference from the Oracle database system is in the disk sharing policy, i.e. in the *shared-nothing* vs. *shared-disk* architecture [22].

Note that even if the term “federated database” is used to characterize SQL Server, its architecture, as well as the architecture of the other two database systems, is closer to the architecture of a DDBS, rather than to the one of a tightly coupled FDBS. In these approaches, database schema design, data placement and partitioning among the servers is usually decided a priori by the system administrator.

IBM proposes a family of products, whose architecture is similar to that of a tightly coupled FDBS. The family includes DB2 Connect, and DB2 DataJoiner [78]. DB2 Connect allows it to integrate autonomous data

³If The Database Partitioning Feature is installed.

sources (e.g. relational databases, XML files) and provide their uniform integrated view for users and applications. It supports read-only SQL queries, which are formulated against the integrated federation schema. DB2 DataJoiner provides tighter integration, and it supports also modification operations on the federation data.

2.4 Ontology-based Systems

The most quoted definition of an ontology is “*an explicit specification of a conceptualization*” [76]. A *conceptualization* is an abstract model that describes objects, concepts, and other entities, peculiar to some (usually restricted) domain, as well as relationships that hold between these entities. An *explicit specification* means that the entities and relationships in the abstract model are given explicit names and semantics, expressed in some *formal* language. In practice, this usually means a logic-based language, as it allows for automated reasoning [121]. Ontologies are seen as the main technological instrument for realizing the Semantic Web [10].

Ontologies are intrinsically intended for knowledge *representation, reasoning with, and sharing* across different applications and communities. This is probably the main difference from database schemata, whose role is to structure a set of data items for querying tasks, and which are not necessarily shared among different parties. The interested reader is referred to [155] for a detailed discussion of the differences between ontologies and database schemata.

Usually an ontology has no data instances associated with its classes, and it is all-sufficient for most of the intended operations⁴. When ontology classes are equipped with respective instances, then such an ontology is called a *populated* ontology. Populated ontologies form the core of ontology-

⁴For a comprehensive list of concrete application scenarios of ontologies see [145].

2.4. ONTOLOGY-BASED SYSTEMS

IS	Autonomy					Heterogeneity			Distribution			Dyn.	Sc.
	D.	C.	Ex.	As.	P.	D. M.	S. L.	D. L.	P.	L.	Op.		
Ont.-based	L	Y	Y	Y	N	N	Y	Y	D	C	C	N	N

Table 2.3: A typical profile of an ontology-based IS. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; L = limited.

based ISs; and we define an *ontology-based information system* as an IS in which ontology represents a means to access, query, and manipulate the underlying data instances⁵. Note that the underlying data can be stored in a database.

Some examples of ontology-based information systems are SIMS [25], OBSERVER [110], BUSTER [148], KRAFT [127], COIN [72], Ontobroker [48], and DQW [39]. The interested reader is referred to [159] for a survey of existing ontology-based approaches. In the following we discuss some of the systems in the light of the analysis dimensions presented in Section 2.2. We summarize our conclusions in Table 2.3.

Due to the fact that ontologies are intended to be *shared* among *several* parties, ontology-based ISs are intrinsically centralized. Centralization can take the form of a global ontology to which local sources are mapped (e.g. SIMS); or the form of a shared vocabulary or an ontology to which local ontologies are related (e.g. COIN, BUSTER, KRAFT). These systems are logically centralized, even through they are physically distributed. OBSERVER supports multiple ontologies without a global ontology (i.e. it is logically distributed), but it relies on a centralized component (called Interontology Relationships Manager) to connect lexically related terms (e.g. synonyms, hyponyms) found in different ontologies, and, therefore, it

⁵Note that there are other uses of ontologies in ISs. For instance, ontologies can be used for supporting interoperability between IS's components; for designing of database schemata; they can be a part of the IS application, thus statically encoding domain knowledge into the application; or they can be used in another way in the IS engineering process [77].

is operationally centralized.

Development of an ontology requires domain-specific knowledge, methodological skills, and knowledge of the language in which the ontology is to be expressed [91]⁶. This is a knowledge and labour intensive task, and it requires the involvement of narrowly specialized experts, which makes it a non-cheap task as such. Apart from this, if an ontology is shared among several parties (which is the core idea behind ontologies), then changing the ontology may mean re-negotiation between the parties in order to reach a new consensus. All this makes ontologies very reluctant to any changes. In fact, as reported in [159], the existing approaches do not support ontology evolution⁷. For what regards adding a new ontology, in most cases it requires the specification of mappings, which would integrate the ontology into the system. This task may be as hard as the task of developing a new ontology. Thus, ontology-based ISs are static – ontologies are carefully tailored, and are supposed to remain unchanged. For the same reason, participation autonomy in an ontology-based IS is very limited.

Heterogeneity in ontology-based ISs has a slightly different form w.r.t. the case of database systems. First, in the typical case of a shared global ontology, the parties agree on the shared semantics and, therefore, the problem of heterogeneity is solved *a priori* (e.g. SIMS). In the case when ontologies are developed independently by domain experts, the use of terms is likely to be similar due to the use of a domain-specific vocabulary, whereas the structure of ontologies can be rather different. Ontology languages are usually expressive and allow for the definition of complex ontological structures, which may be much more tangled than (relational) database

⁶See [91] for a survey of ontology development methodologies, and [156] for a detailed discussion of one of the methodologies.

⁷According to [159], only the SHOE ontological language [87] provides adequate tools for ontology evolution. However, a SHOE implementation represents a retrieval, annotation and querying tool, rather than an IS. Another relevant reference is [122], where a tool for ontology versioning is presented.

schemata. For this reason, by participating in an ontology-based IS, parties agree to use the same knowledge representation formalism, which introduces homogeneity at the data model level, and which limits design autonomy of the parties. Summarizing, ontology-based ISs are usually rather homogeneous, however, if an IS is composed of independently developed ontologies, semantic heterogeneity may represent a serious problem.

Ontology-based ISs, relying on the notion of a global ontology or a shared vocabulary, are unscalable (e.g. SIMS, COIN, BUSTER, KRAFT). In fact, as noted in [75], from the involved parties it requires *ontological commitments*, which are “*agreements to use the shared vocabulary in a coherent and consistent manner*”. Thus, if the number of ontologies grows significantly, then reaching an ontological commitment becomes a practically unmanageable task.

Scalability of logically distributed systems (e.g. OBSERVER) is still an open question and an active field of research. For instance, one of the main research lines of the Knowledge Web network of excellence project [158] is largely devoted to this topic. One of the major challenges is in finding a tradeoff between expressivity of ontological formal languages and scalability [135].

To complete the overview we must mention some recent approaches to building ontology-based ISs, which are based on the notion of *lightweight ontology* [155]. Loosely speaking, a lightweight ontology is a formalized tree-like structure, where nodes’s labels encode concepts, and edges represent the intersection relationship between the concepts [62]. Lightweight ontologies are a way to formalize knowledge encoded in human-crafted natural language classifications (e.g. a library classification, a web directory, a personal classification of music files), such that it becomes suitable for automated reasoning [62].

One of the prominent approaches, that uses lightweight ontologies, is the

EDAMOK (*Enabling Distributed and Autonomous Management of Knowledge*) project [32]. In this approach lightweight ontologies are used to describe personal knowledge of parties; however the main power of ontology, i.e. where reasoning is used, is in automatic computation of relations that hold between ontology classes of different parties. Another relevant project is SWAP [49] (*Semantic Web and Peer-to-Peer*), which allows it to extract knowledge from tree-like structures (such as email folders, file system folders, favourite bookmarks classifications), annotate it with metadata, and share it among parties on a P2P network.

Because they are based on computationally inexpensive formal languages, lightweight-ontology-based ISs have the potential to scale to a large number of nodes. However, their ability to operate in dynamic environments and to be fully distributed has still to be studied.

2.5 Where and Why Existing Approaches Fail

The recent advances in the networking technology, as well as the constantly growing processing and storage capacity of computational devices, both stipulate higher degree of distribution and interconnectivity in computer networks, while favouring significant autonomy in computation of individual participating sites. This changes the focus from intrinsically centralized to more distributed architectures, giving rise to more collaborative, dynamic, and pervasive (business) applications, which cross enterprise boundaries [163, 151, 27, 137].

While significant autonomy and distribution become rather a fact, than a design requirement of these applications, their ability to scale, and support dynamics and heterogeneity has been recognized as the main success factors, which are tackled by both research and industry [13, 14, 81]. Thus,

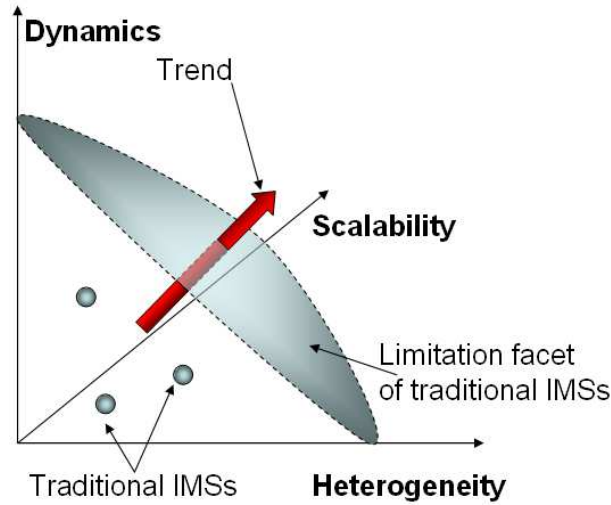


Figure 2.6: Limitation facet of traditional IMSs.

for instance, the NoE project Knowledge Web [13] names scalability, heterogeneity, and dynamics (in the context of ontology-based systems) as “*the main aspects for realizing the semantic web*”. Another EU-funded project, OpenKnowledge [14], is focused on dynamic knowledge discovery and reuse in large-scale heterogeneous peer-to-peer systems. On the industrial arena, scalability, and dynamic access to heterogeneous data sources at runtime are recognized to be the primary concerns of the relatively new approach to integration, which is called, *Enterprise Information Integration* [81].

As it can be seen, there is a clear trend towards information management systems which would expose good performance in all the three dimensions: scalability, heterogeneity, and dynamics. However, this is where existing approaches fail, as we have shown in the previous two sections. In Figure 2.6 we position the existing IMSs within the three dimensions and show their limitation facet.

Chapter 3

P2P Information Management Systems

In this chapter we introduce P2P IMSs as a new kind of information management systems. As we show, P2P IMSs have the potential to overcome the limitations of the conventional IMSs discussed in the previous chapter. However, the new way of information management poses some new serious research challenges. In this chapter we discuss these challenges and propose a solution for how to solve them.

The chapter structure is as follows. In Section 3.1 we provide a definition of P2P IMSs, which discriminates them from the conventional IMSs, discussed in the previous chapter. In Section 3.2 we introduce a P2P information management model and discuss its core notions. In Section 3.3 we propose a generic query answering algorithm that can be implemented in database-based P2P ISs and, potentially, in ontology-based P2P ISs. In Section 3.4 we discuss the novel notion of good-enough answers. Finally, in Section 3.5 we discuss the system requirements and a logical architecture of a P2P IMS.

3.1 Definition

“Peer-to-peer is a class of applications that take 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, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers”

Quote from Clay Shirkey [125]

The above definition from Clay Shirkey is probably the most cited definition of what Peer-to-Peer (P2P) is. Due to its generality, many applications were headed under the title of P2P. Their application domains include distributed computing (e.g. SETI@Home [7]), instant messaging (e.g. ICQ [3]), collaborative networking (e.g. Groove [2]), file sharing (e.g. Napster [6], Kazaa [4], Morpheus [5]), and Internet telephony (e.g. Skype [8]).

P2P systems are usually attributed with such properties as autonomy, distribution, decentralization, equal peers’ functionality, point-to-point peer interaction [112]. However, all these properties are rarely found in existing P2P systems. For instance, in SETI@Home a centralized server sends computational tasks to peers, which then send the result of computation back to the server. No peer intercommunication takes place. Instant messaging and first file sharing applications (e.g. Napster) run a central server to maintain an index of available resources (such as users or files), which is used for resource discovery. Point-to-point communication in these applications takes place only when users exchange messages or files with each other. In more recent file sharing applications (e.g. Kazaa) the index is distributed among special peers, which are called super-peers, and which

are connected in a P2P fashion. As it can be noted, not all peers support equal functionality. Finally, in the file sharing applications all peers have the same schema, used for describing file metadata, and for submitting queries. Thus, peers are not totally autonomous in their operation.

Recently, a lot of research has been done in the area of heterogeneous schema-based P2P data management systems [31, 82, 120, 56, 117, 132, 40, 18]. These approaches endow peers with more autonomy by allowing independent schema specification, i.e. they favour design autonomy. Many researchers motivated their work in this field by contrasting with the data integration approach, where the notion of a unique global schema is central to the system design and operation [101]. However, the idea that a data management system can run without a global schema, and that connections between system components can be made in the point-to-point fashion is far not new. Back in 1985 a very similar idea was introduced in the context of federated database systems [88] and multidatabase systems [103] (for more details see Sections 2.3.4 and 2.3.5). The fundamental difference of these approaches and heterogeneous schema-based data management P2P systems consists of the following two principles:

- **Locality:** peers have only a *partial* view of the system, i.e. they know only a (small) subset of other peers, called *acquaintances*, and they know only about connections between them and their acquaintances. Intuitively, *knows* for peers means that a peer knows about another peer, apart from the fact of its existence, what kind of data is stored at it; and, *knows* for peer connections means that a peer knows how to send a query from one peer to another. In Figure 3.1a we schematically illustrate the locality principle. There, circles stand for peers and arrows stand for connections between peers. Dashed objects indicate the unknown part of the system for peer 1. Thus, for instance, it is not aware of peers 4, 5, and 6, of connections between these and other

3.1. DEFINITION

peers, as well as of connections between its acquaintances, peers 2 and 3;

- **Transitivity:** when a peer receives a request or data from another peer, as part of their processing, the peer can propagate the request or the data to another peer. This propagation occurs *transitively* from peer to peer, activating at each intermediate peer its connections with its acquaintances¹. The principle of locality is applied also to transitivity. Namely, intermediate peers do not know what peer originated the request, they consider it as if it were originated by a neighbouring peer. In Figure 3.1b we schematically illustrate the transitivity principle. There, peer 1 sends a request to peer 2, and the latter, as part of the processing of the request, propagates it to peer 3. Peer 3 does not know that peer 1 originated the request, it perceives the request as if it were originated by peer 2.

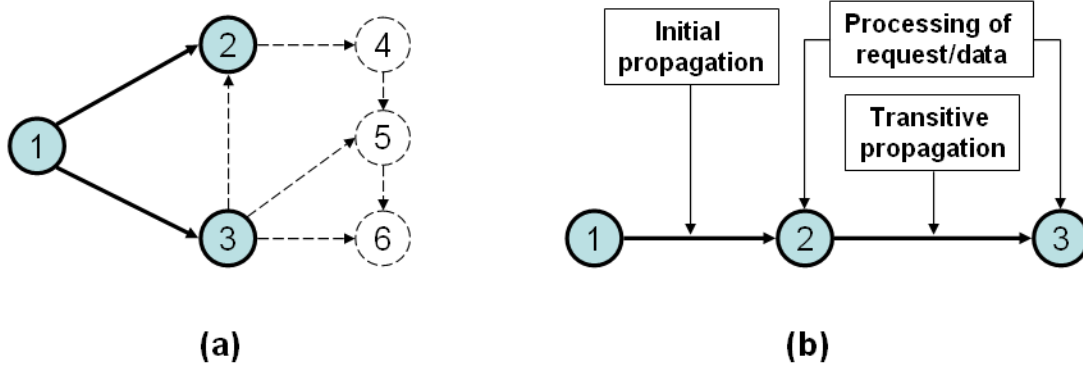


Figure 3.1: Principles of locality and transitivity.

Note that total autonomy and full distribution make it technically very hard to maintain a consistent global view of the whole system by all the

¹We thank Phil Bernstein for help in understanding the crucial role of the transitivity principle in discriminating P2P data management systems from the traditional ones.

peers at all times. Therefore, in practice, total autonomy and full distribution stipulate the locality principle. Taking this consideration into account, we define the notion of Peer-to-Peer in the context of data management systems as:

Peer-to-Peer is a fully distributed communication model in which totally autonomous parties have equivalent functional capabilities in providing each other with data and services, and they do it in a transitive point-to-point way.

This definition has several keystones:

- “.. fully distributed ..” means physical, logical, and operational distribution as discussed in Section 2.2.3;
- “.. totally autonomous parties ..” means that the parties possess design, communication, execution, association, and participation autonomy as discussed in Section 2.2.1;
- “.. equivalent functional capabilities ..” means that all peers can perform the same set of functions. Note that some peers can still be more powerful in terms of processing capability, storage, and/or network bandwidth;
- “.. in a transitive point-to-point way.” means that the principle of transitivity holds.

Since peers are fully autonomous, their *LSSs* can be highly heterogeneous, i.e. they can possess all the forms of heterogeneity discussed in Section 2.2.2. Apart from this, full autonomy stipulates that the P2P system can be highly dynamic (see Section 2.2.4). On the other hand, P2P systems are said to be much more scalable than traditional centralized systems due to avoiding the dependency on centralized points [112].

3.1. DEFINITION

Therefore, IMSs, built on top of P2P architectures, have the potential to overcome the shortcomings of traditional IMSs discussed in Section 2.5. In Figure 3.2 we show where P2P IMSs can be positioned in the dimensions of heterogeneity, dynamics, and scalability (compare it with Figure 2.6).

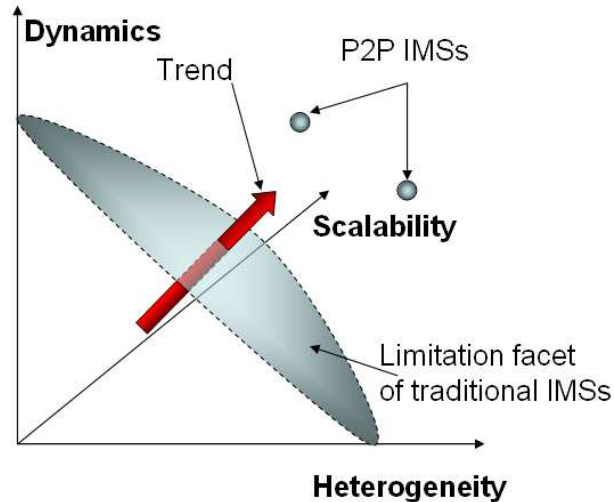


Figure 3.2: P2P IMSs and the limitation facet of traditional IMSs.

However, as recent research shows, inefficient routing of requests and data may lead to the flooding effect in the system, thus causing a denial of service. There are other factors that make scalability in P2P systems challenging, such as discovery of resources in a fully distributed environment. Therefore, scalability is a promising, desirable, but still vulnerable property of P2P data management systems.

We define a Peer-to-Peer information system (P2P IS) as *an IS consisting of multiple totally autonomous LISs, which is fully distributed, and which is scalable and dynamic*. A Peer-to-Peer information management system (P2P IMS) is *an IMS which permits the management of a P2P IS, and which implements the management based on the transitivity principle*. A peer, in the context of a P2P IS, is a LIS, a LSS, and a copy of the P2P IMS software used for the management of the LIS. Conceptually, each peer

has only one LIS, even if factually it may have several distinct data sources. Each peer can be driven by a user, who initiates requests to the P2P IS.

3.2 A P2P Information Management Model

Development of a P2P IMS is a challenging task. Roughly speaking, its ultimate goal is to allow peers and users to access and manipulate information stored in the P2P IS, at the same time keeping participation cost for an individual peer low, and its benefit essential and immediate. To reach this goal, a P2P IMS must introduce order to the chaotic set of LISs *without* sacrificing their autonomy, and support dynamic and scalable self-organization of the P2P IS.

In this section we propose a novel information management model which addresses the above described challenges, and which can be implemented within a P2P IMS. There are four notions which form the core of the model. The notions are: *interest group*, *acquaintance*, *coordination rule*, and *correspondence rule*. We discuss the notions in the following sections.

3.2.1 Interest groups

Full participation autonomy allows for an arbitrary large number of peers that can join a P2P network. Since a P2P IS is fully distributed, it becomes very hard to locate peers which may potentially be able to answer a given query. We define the notion of *interest group* as a *non-empty set of peers which are able to answer queries about a certain topic*. Intuitively, *tourism in Trentino*, *sport cars*, *medical care*, are all possible topics. A topic can be formalized as a set of keywords (as used in [97]), as an ontology (as used in [140]), or as a context [60] (as used in [90]).

A peer may belong to multiple groups, and this favours the participation autonomy of peers. In a P2P IS, interest groups have three main purposes, discussed below:

- first, interest groups allow it to describe the contents of a P2P IS at a meta-level by grouping peers having similar contents and by assigning to these groups topic descriptions, which, in an aggregated manner, describe the contents of all the peers in the groups. When a new peer joins the P2P IS, it looks for one or more peer groups, it is interested to be a part of;
- second, since peers that join the same interest group have similar contents, they are likely to share a vocabulary *a priori*, which eases their further interoperation and allows for more meaningful query answers. In fact, asking a peer about a query whose topic is unknown to the peer will hardly produce any meaningful answer. Apart from this, interest groups provide grounds for further semantic vocabulary convergence through meaning negotiation and gradually reaching an agreement about the meaning of terms [133];
- third, interest groups are the starting point for peers to submit queries to a P2P IS. Namely, instead of searching for single peers that could answer its query, a peer looks for one or more interest groups according to their topic. This means that the input query must be associated with a topic (this could be done by the user or by the system itself), which is then compared with the available interest groups' topics. Then, the scope of the propagation of a query is limited to peers of the group(s), which are selected for the query propagation. Note that in order to be able to submit a query to an interest group, a peer must be a member of that group.

There are two main approaches to the way of creating interest groups: *bottom-up* and *top-down*. In the bottom-up approach, interest groups are identified by analyzing descriptions of existing peers in the system (e.g. see [98]), or by analyzing connections established among peers (e.g. see [54]). Note that the latter approach does not let peers be members of different groups simultaneously [97]. In the top-down approach peers explicitly and deliberately create, join and leave interest groups. The latter approach is implemented, for instance, in the P2P platform JXTA [1], and, consequently, in many its applications².

In our approach we follow the top-down philosophy and we support the organization of interest groups into *hierarchies* (see Figure 3.3 for an example of an interest group hierarchy). First, the top-down approach provides peers with more control of their participation in the P2P IS, thus favouring their autonomy. Second, the hierarchical representation of interest groups provides a better structuring of the information storage space of the P2P IS. Participation of peers in interest groups, and peer group self-organization is conditioned by the following principles:

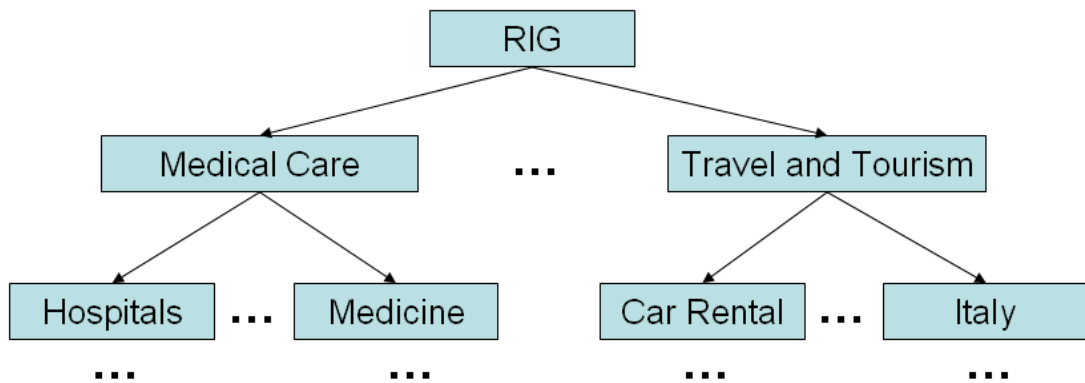


Figure 3.3: Interest group hierarchy.

- the root interest group (RIG) in the hierarchy is the entrance point for

²See <http://apps.jxta.org/> for a list of applications built on top of JXTA.

new peers. Peers with arbitrary content are free to become members of RIG;

- any peer can create a (child) interest group in any (parent) interest group. That peer is called the *group manager* for this interest group. Group managers define the topic for their groups, and are expected to be committed to supporting the group, i.e. be more available, more active and more capable (e.g. in terms of contents, or CPU power) in servicing peers' requests than other peers eventually joining the group;
- when a new peer group is created, an announcement containing its topic and its position in the hierarchy is broadcast to the other peers in the P2P IS;
- when a peer receives an announcement of a new interest group, it merges it into the structure of the group hierarchy, maintained locally by the peer;
- since peers may temporarily leave the P2P IS, their view of the interest group hierarchy may be partial, and therefore it represents local peer knowledge about interest groups in the P2P IS;
- when a new peer enters the P2P IS through joining RIG, it queries a subset of peers for interest group structure information, and merges received responses into one monolithic interest group hierarchy. Potential inconsistencies are resolved by using a majority rule, as it is used, for instance, in [46];
- since the interest group hierarchy can be rather large, the search for an appropriate group can be facilitated by automatic classification of the peer into the interest group hierarchy based on semantic analysis [61, 62]. In [61, 62], we discuss how semantic information, implicitly

encoded into the labels and structure of a classification hierarchy, can be made explicit, and how it can be used for automating classification of documents into the hierarchy. This approach can be largely re-used in a P2P IS by substituting the notion of classification hierarchy with the one of interest group hierarchy, and by considering peer descriptions instead of document descriptions in the classification process. In a similar manner user queries can be classified into the interest group hierarchy in order to identify relevant interest groups for query propagation;

- in order to join an interest group (which is not RIG), a peer must show an evidence that it has contents relevant to the interest group topic. In order to do this, the newcomer peer locates a member peer of the interest group and submits to it its membership application containing information about the peer's LSS. The member of the group performs matchmaking of the LSS to the group topic, and, if the result of matching satisfies some criteria, then the membership application is accepted. We call this procedure, the *membership application protocol*. If the topic of an interest group is a schema, then the matchmaking process can be performed through similarity analysis of the two schemata, which returns some numerical coefficient (e.g. see [107]); or through the so called "semantic matching" [65, 67, 64, 66, 63], which returns logical relations (e.g. subsumption) that hold between elements of the two schemata³. The criteria in the former case may be that the similarity coefficient must be higher than some predefined threshold. In the latter case the criteria may be that certain relations must hold for certain elements in the topic schema.

Note that the above principles allow for the organization of peers into

³For surveys of schema matching approaches see [143, 129].

interest groups in a *completely decentralized* manner. Moreover, the distributed membership application protocol allows it to introduce some order into the chaotic self-organization process by requiring peers of the same interest group to be similar enough in their data contents.

3.2.2 Acquaintances

Once a peer has joined an interest group, it may make *acquaintances* with other peers in the group. Acquaintances are peers that a peer knows about and which have data that can be used to answer a *specific query*. Intuitively, if a peer is an acquaintance, then there must be a way to reformulate the specific query w.r.t. its LSS, to interpret query results coming from it, and to reconcile them with the results coming from other acquaintances [68].

We formalize the above intuition in the notion of *acquaintance query*. An acquaintance query is a triple $\mathcal{A}Q^{ij} = \langle Q_i, Q_j, M_{ij} \rangle$, where:

- Q_i is a query formulated w.r.t. the LSS of peer i . In the following we call Q_i the *local definition* of the acquaintance query $\mathcal{A}Q^{ij}$;
- Q_j is a query formulated w.r.t. the LSS of peer j , which is an acquaintance to peer i . In the following we call Q_j the *remote definition* of the acquaintance query $\mathcal{A}Q^{ij}$; and
- M_{ij} is a set of mappings relating schema elements referred by Q_i to schema elements referred by Q_j .

Queries Q_i and Q_j are meant to be semantically related, i.e. they query semantically related data on the two peers. Note that the notion of acquaintance query allows it to (partly) solve the problem of schema level heterogeneity among the two peer schemata. Namely, an acquaintance query defines two semantically correlated queries on the schemata of the two peers and mappings between corresponding elements of the two queries.

The notion of acquaintance is directional – if one peer is an acquaintance of another, it does not necessarily mean that the reverse also holds. We say that a peer is an *acquainted peer* for some other peer, if the latter is an acquaintance of the former. Acquaintances can therefore be thought of as *logical links*, directed from one peer to another, and labelled by acquaintance queries.

The notion of acquaintance query is conceptually different from the notion of interest group topic. An interest group topic defines the outer boundary for the shared contents of the LISs of peers in the interest group, whereas an acquaintance query defines a way of how particular contents can be imported from an acquaintance. As a consequence, an interest group topic must be general enough to capture most of the relevant peers, but not too general to avoid inefficient query answering. On other hand, an acquaintance query must be as specific as possible in order to ensure effective query propagation, preserving the query semantics at the best possible level of approximation.

The notion of acquaintance complements the notion of interest group in the sense that together they form a basic *logical infrastructure* for peer intercommunication. In fact, while interest groups define the scope of query propagation, acquaintances define logical links through which queries and data can be propagated. The intuition is that a peer will try to propagate a query to its acquaintances according to the definitions of appropriate acquaintance queries.

Apart from acquaintance queries, logical links between peers can also be labelled by trust values, which show how much one peer trusts answers coming from its acquaintance [166, 43, 92]. This information can then be used, for instance, to encourage peers to share data with other peers (as discussed in [92]), or to compute a relative value of how much a peer trusts an *aggregated* answer, computed transitively through chains of peers (as

we discuss in [166]).

Example 1 *Let us consider an example taken from the database domain. In Figure 3.4 we show two peers and their relational LSSs, each consisting of two relations. Peer 2 is an acquaintance of peer 1 w.r.t. the following query (expressed w.r.t. LSS_1):*

$$Q_1(\text{title}, \text{genre}, \text{budget}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{genre}, \text{budget})^4.$$

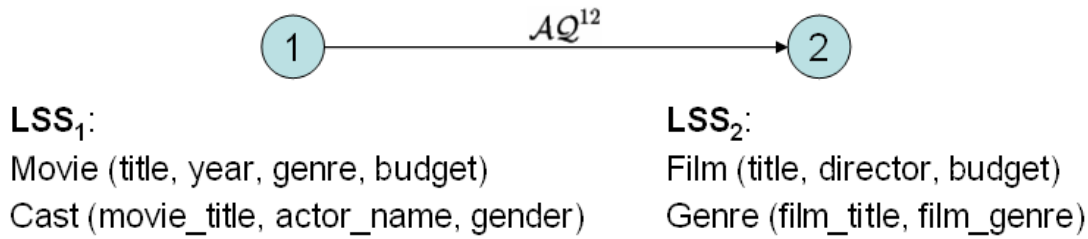


Figure 3.4: Acquaintance query.

As from the definition of acquaintance, peer 1 knows how to answer Q_1 using the data of peer 2. In order to technically implement it, peer 1 stores query Q_2 , which is expressed w.r.t. LSS_2 :

$$Q_2(\text{title}, \text{film_genre}, \text{budget}) \leftarrow \text{Film}(\text{title}, \text{director}, \text{budget});$$

$$\text{Genre}(\text{film_title}, \text{film_genre}); \text{title} = \text{film_title}$$

Note that Q_1 and Q_2 are semantically related, i.e. they retrieve semantically relevant data. In order to link the two queries, peer 1 stores a set of mappings, which relate attributes referred by Q_1 to attributes referred by Q_2 :

$$M_{12} = \{1 : \text{title} \rightarrow 2 : \text{title}, 1 : \text{genre} \rightarrow 2 : \text{film_genre}, 1 : \text{budget} \rightarrow 2 : \text{budget}\}.$$

⁴In this and other examples we use the conjunctive query notation to represent relational queries [153]. We discuss conjunctive queries in more detail in Section 4.2.

3.2.3 Coordination rules

Having a logical infrastructure for peer intercommunication is not enough to enable the operation of a P2P IS. In fact, given a query or a new piece of data, a peer has to understand how to process the query or the data. Possible options include the identification of other peers it should be propagated to, the identification of if and how the query or the data must be modified in order to be valid w.r.t. a particular peer's LSS, and so on. To cope with this problem we introduce the notion of *coordination rule*. A coordination rule is a procedure, stored and implemented locally by a peer, whose input is an event (such as a query from the user), and whose output is a sequence of actions (such as query propagation or results reconciliation). The output of a coordination rule depends on the kind of event (e.g. a query or an update), as well as on other conditions (e.g. what elements of LSS are referenced, if and when a similar query was submitted in the past). On Figure 3.5 we represent a functional view of a coordination rule.

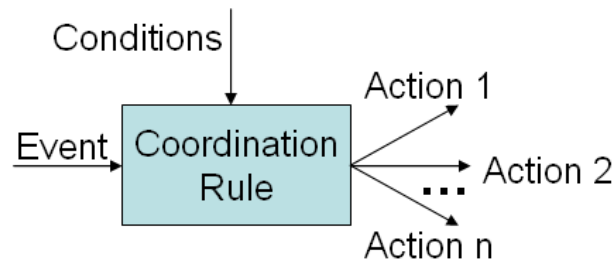


Figure 3.5: Coordination rule.

Each peer maintains a set of coordination rules, and it uses them to specify, at *runtime*, under what conditions, when, how and where to propagate queries, query results, and updates to acquaintances and acquainted peers. Coordination rules are the key enabling mechanism of the query answering algorithm, discussed in Section 3.3. An action of a coordination rule by some peer may trigger an event for another coordination rule by

another peer (see Figure 3.6). This is how coordination rules implement the transitivity principle.

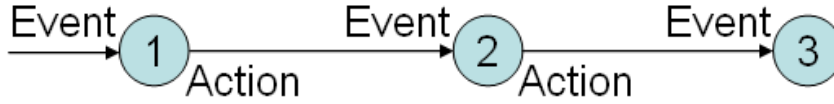


Figure 3.6: Transitivity property of the coordination rules.

3.2.4 Correspondence rules

As discussed earlier, acquaintance queries allow it to address the problem of schema level heterogeneity. However, it is still not enough for a meaningful data exchange among peers, as different peers may use different values to represent semantically the same pieces of raw data, i.e. data level heterogeneity may still take place. In order to solve this problem we introduce the notion of *correspondence rule*.

A correspondence rule is a tuple $\mathcal{CR} = \langle f_{ij}, f_{ji} \rangle$, where:

- f_{ij} is a function that takes a raw data value from the domain of peer i as input and returns a corresponding raw data value in the domain of acquaintance peer j as output; and
- f_{ji} is a function that takes a raw data value from the domain of peer j as input and returns a corresponding raw data value in the domain of acquaintance peer i as output.

Correspondence rules are used for the translation of queries propagated to (when raw data values are specified as part of query constraints), and query results propagated from an acquaintance. For each acquaintance j , node i defines a (possibly empty) set of correspondence rules. More specifically, a node may associate a correspondence rule on the domain of

a schema element referred in the local definition Q_i of any acquaintance query $\mathcal{AQ}^{ij} = \langle Q_i, Q_j, M_{ij} \rangle$. When no correspondence rule for a schema element is defined, then it means that the peer considers the domain of this element to be shared with the one of acquaintance, i.e. it is equivalent to defining the following correspondence rule: $\mathcal{CR} = \langle f_{ij}(x) = x, f_{ji}(x) = x \rangle$.

Example 2 Recall the example of an acquaintance peer from Example 1. Suppose that peer 1 saves budget data in Euros, and peer 2 saves it in US dollars. Thus, each time a query with a constant for the attribute `budget` is issued by peer 1, or when query results from peer 2 with data for the attribute `budget` are returned, peer 1 has to translate between the two currencies. In order to address this problem, peer 1 specifies the following correspondence rule for the attribute `budget`: $\mathcal{CR} = \langle f_{12}(x) = x * 1.21, f_{21}(x) = x/1.21 \rangle$.

Function f_{ij} is not necessarily an inverse function for f_{ji} , i.e. it is not necessarily the case that the following holds: $f_{ij}(f_{ji}(x)) = x$ (as it is the case in the previous example). An example of the situation when it does not hold is when there is a precision conflict among the two schemata. For instance, suppose that peer i measures distance in miles and stores this information as integer values, and its acquaintance j measures distance in kilometers and also stores it as integer values. Then, a correspondence rule set up at peer i might be: $\mathcal{CR} = \langle f_{ij}(x) = \text{int}(x * 1.6), f_{ji}(x) = \text{int}(x/1.6) \rangle$. Note that $f_{ij}(6) = 9$ and $f_{ji}(9) = 5$.

3.3 Principles of a Query Answering Algorithm

Search for data is probably the most important operation in an IS. In a P2P IS this is also the point where peers collect dividends for their investments.

Namely, for their effort to establish a small set of acquaintances, peers are rewarded with immediate access to data stored potentially on a large set of peers. As discussed earlier, the intuition is that peers collaborate in processing user requests by propagating *transitively* to one another queries, query results, and updates. However, in a fully distributed environment, where peer connections are absolutely arbitrary, this process may be by far not trivial due to the presence of loops, “bottlenecks” in data propagation, and so on.

In this section we discuss general principles of a query answering algorithm for a P2P IS, highlight potential threats, and suggest how they can be avoided. The discussion in this section is general, so that its arguments can be applied to various kinds of P2P ISs (e.g. database-based, ontology-based). We provide technical details of a query answering algorithm for the case of database-based P2P IS in Section 4.4.2.

The proposed query answering algorithm is built on top of the four notions discussed in the previous section. It can logically be split into three sub-algorithms: *query propagation*, *query results propagation*, and *query answering termination algorithms*. In the following sections we discuss these algorithms in detail.

3.3.1 Query propagation

The query propagation algorithm can be decomposed into five steps. We discuss these steps below and we represent some of them in Figure 3.7 (numbers on the left in the figure stand for the associated steps from below):

1. **Get a query:** query propagation starts when a user submits query Q_u at some peer i . We call this query, a *user query*. User queries are expressed w.r.t. the LSS of the peer where they are submitted;

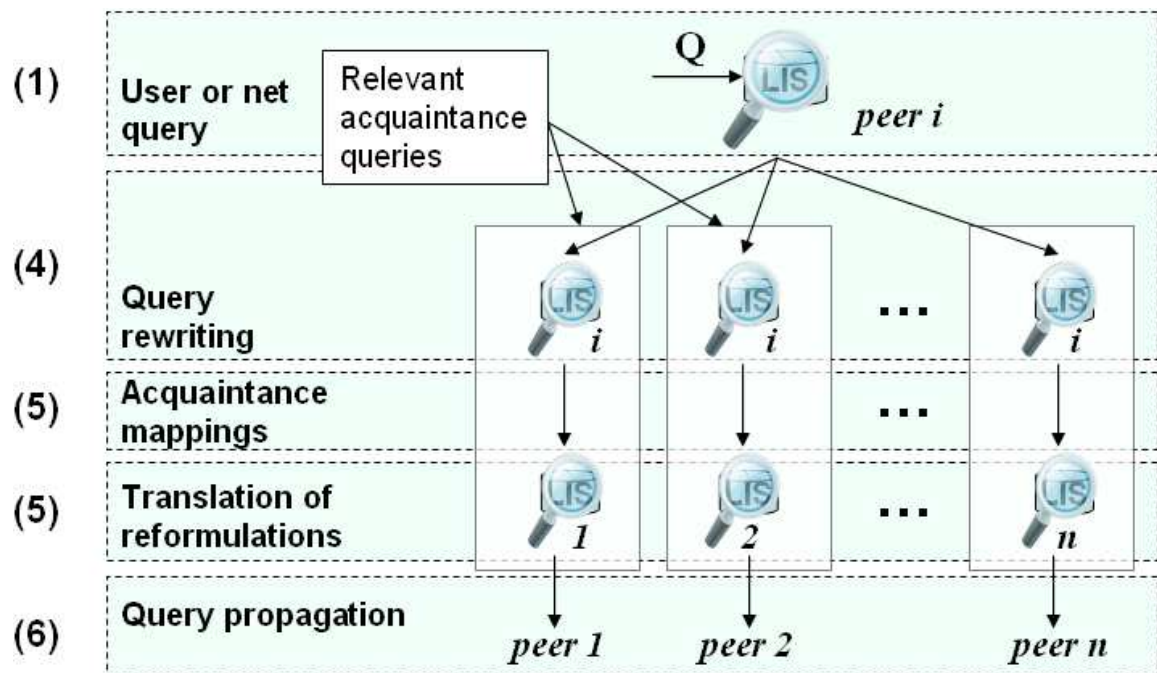


Figure 3.7: Query propagation.

- 2. Compute interest groups:** possibly with help of the user, or by means of a formal classification of the user query into the interest group hierarchy [62], the user query is associated with one or more interest groups. Once relevant interest groups are decided, query propagation is limited to the peers of these groups;
- 3. Compute local answer:** Q_u is answered using data from LIS_i . The user gets the first result for his/her query;
- 4. Rewrite query:** for each of the selected interest groups, $peer\ i$ looks for acquaintances, which belong to the interest group, and which can (partly) answer Q_u . Since each acquaintance query $\mathcal{A}Q^{ij} = \langle Q_i, Q_j, M_{ij} \rangle$ encodes how some particular local query Q_i (i.e. the local definition of $\mathcal{A}Q^{ij}$) can be answered using data from acquaintance j , $peer\ i$ looks for how Q_u can be expressed in terms of the local definitions of the corresponding acquaintance queries. Conceptually,

this problem is equivalent to the problem of *rewriting queries using views*, which is thoroughly studied in the context of database systems and data integration [80], and which only starts to be explored in the context of ontology-based systems [38, 149]. In this process, constraints defined in Q_u are “pushed” to the local definitions of the acquaintance queries. Hereafter we call local definitions which are part of the rewriting of a query, and which potentially include constraints of that query, *reformulations* of the query;

5. **Translate reformulations:** reformulations of the query are translated, using acquaintance mappings and correspondence rules, into queries w.r.t. the LSSs of the corresponding acquaintances. Namely, any constraints of Q_u , which are “pushed” into the reformulations, are further “pushed” into the corresponding remote definitions according to the acquaintance mappings. If these constraints include data instances from the domain of peer i , then the data instances are translated according to the correspondence rules;
6. **Propagate queries:** for each selected interest group, peer i propagates the translated reformulations to the corresponding acquaintances. Along with the query, the interest group identifier is also propagated.

Example 3 Recall the two peers and acquaintance query AQ^{12} from Example 1. Suppose now that the user of peer 1 searches for movies starring Tom Cruise with the budget of more than eighty million Euros (recall that peer 1 stores budget information in Euros). Thus, the user specifies the following query:

$$Q_u(\text{title}, \text{genre}, \text{budget}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{genre}, \text{budget});$$

$$\text{Cast}(\text{movie_title}, \text{actor_name}, \text{gender}); \text{title} = \text{movie_title};$$

$$\text{actor_name} = \text{“Tom Cruise”}; \text{budget} > 80000000$$

Suppose that peer 2 is in an interest group, selected at step 2. Then, the user query can be rewritten at step 4, taking into consideration local definition Q_1 of acquaintance query AQ^{12} , into the following query:

$Q_u(\text{title}, \text{genre}, \text{budget}) \leftarrow Q_1(\text{title}, \text{genre}, \text{budget});$
 $\text{Cast}(\text{movie_title}, \text{actor_name}, \text{gender}); \text{title} = \text{movie_title};$
 $\text{actor_name} = \text{"Tom Cruise"}; \text{budget} > 80,000,000$

Then, taking into account the acquaintance query mappings (which map attribute **budget** in LSS_1 to the corresponding attribute in LSS_2), and the correspondence rule presented in Example 2, we translate the reformulation at step 5 into the following query, expressed w.r.t. LSS_2 :

$Q_2(\text{title}, \text{film_genre}, \text{budget}) \leftarrow \text{Film}(\text{title}, \text{director}, \text{budget});$
 $\text{Genre}(\text{film_title}, \text{film_genre}); \text{title} = \text{film_title}; \text{budget} > 96,800,000$

Note how the constraint ($\text{budget} > 80,000,000$), specified in user query Q_u , is “pushed” into the rewritten query Q_1 , and then further “pushed” into remote definition Q_2 . Finally, query Q_2 is submitted to peer 2 at step 6 of the query propagation algorithm.

As the result of step 6 of the algorithm presented above, acquaintances receive queries from their acquainted peers. We call these queries, *net queries*. Thus, the execution of step 6 by peers results in the initialization of the algorithm from step 1 by their acquaintances. Peers process net queries and transitively propagate them further to their acquaintances. Thus, one user query may result in a set of net queries propagated in the network. We say that each of these net queries *originated* from the user query.

Note that it is not necessary the case that peers use *all* their acquaintance queries for the rewriting of some net (or user) query (see step 4). In fact, peers include only those acquaintance queries, whose local definitions

are part of the rewriting of the given net (or user) query. We call these acquaintance queries, *related acquaintance queries* for the given query. Related acquaintance queries are shown in Figure 3.7 as vertical rectangles.

When peers receive net queries, they skip step 2 of the algorithm, and propagate queries only within the interest group indicated in the net query. They then follow steps 3, 4, 5, and 6, taking into account the following considerations:

- At step 3 peers send back query result to the corresponding acquainted peer;
- Because peers possess communication autonomy, query propagation from each single peer can be absolutely arbitrary; and, due to the fact that queries are propagated *transitively*, the same peer may receive more than one (net) query, originated from the same user query. When a peer receives a net query for a known user query, then we call the newly received net query, a *repeated query*. There may be two possible causes of repeated queries: (a) when different net queries, originated from the same user query, reach a peer following two different paths; and (b) when a net query reaches a peer as the result of a previous propagation of a user or another net query from this peer, or, roughly speaking, when there is a loop in query propagation. In this latter case we call the repeated queries, *repeated loop queries*. An example of situation (a) is shown in Figure 3.8a (there, peer 4 receives two net queries originated from the same user query), and an example of situation (b) is shown in Figure 3.8b (there, peer 1 receives Q_2 as the result of a previous propagation of query Q_1);
- In order to keep track of repeated queries, each user query is associated at step 1 with a pseudo-unique identifier, and this identifier is propagated with all the subsequent net queries. In the totally dis-

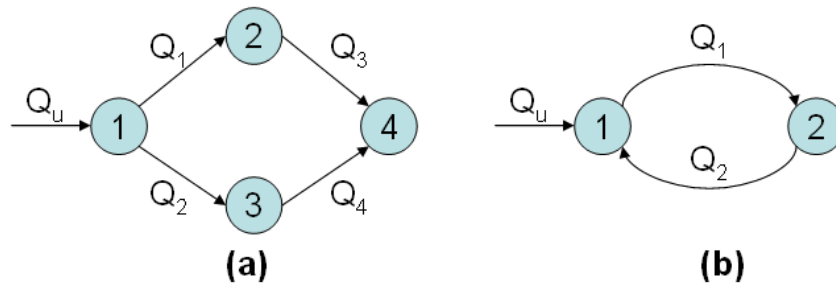


Figure 3.8: Repeated queries in query propagation.

tributed settings, it is not possible to guarantee that two user queries are assigned distinct identifiers, but it is possible to use an ID generation procedure, such that the probability that two identical identifiers are produced (by different peers) during the life time of a single user query is negligibly small. This approach is widely used in practice; for instance, this is how resource identifiers are generated in the project JXTA [1];

- Because they possess execution autonomy, peers decide how to process repeated queries. The first choice a peer has to make is either to compute the query using its LIS (i.e. execute step 3 of the algorithm), or reject the query without computing it. If the query is accepted, then the second decision becomes if and where to propagate the query (i.e. how to execute step 4). Possible options for the first choice include: (1a) all repeated queries are rejected; (1b) a repeated query is rejected if it is not the first query from the same acquainted peer; (1c) a repeated query is rejected if it is not the first query sent using the same acquaintance query link; (1d) all repeated queries are accepted. Propagation strategies for repeated queries include: (2a) no propagation; (2b) a repeated query is propagated only if it is the first query from the given acquainted peer; (2c) a repeated query is propagated if it is the first query received through the given acquaintance query

link; (2d) a repeated query is (partly) propagated only if it cannot be answered using data brought by previously propagated net queries; (2e) repeated queries are always propagated.

Option (2d) requires an additional note: the propagation plan of a repeated query may include queries, which are *more specific* than some previously propagated net queries. In the data integration literature, this relation is called *query containment* [152]. Query A is said to be *contained* in query B (written $A \sqsubseteq B$) if the result set of query A is always a subset of the result set of query B for any instantiation of the underlying data set. Therefore, if a reformulation of a repeated query is contained in some previously propagated net query, then the reformulation can be computed using only the *LIS* of the peer *and* the results brought by the net query.

The two choices discussed above largely affect two important metrics of the P2P IS performance: completeness of the results for the original user query⁵, and network load (mainly due to the following query results propagation). Both consequently affect the QoS provided by the P2P IS to the end users. For instance, option (1a) represents a “minimal collaboration” scenario in which a peer agrees to provide an answer for some user query only once. This approach minimizes network load but negatively affects completeness of user query results. On the other hand, option (2e) may lead to excessive network load, bottlenecks in query answering (when peers become incapable to process all incoming query results), but it may provide the user with a more complete answer. The two choices are *independently* made by each individual peer, and they depend on various parameters as processing capacity of the peer, amount of data stored in its LIS, number of

⁵We discuss what completeness for query results means in a P2P IS in Section 3.4. Here, this notion is considered in its widely accepted interpretation.

acquaintances, its current load, and so on.

- Query propagation is guaranteed to terminate under options (2a), (2b) and (2c). The reason is that, during the time of query propagation, each peer has a limited constant number of static acquaintance queries (i.e. at the moment we assume that the P2P IS is static). Options (2d) and (2e) are not guaranteed to terminate even when the network is static. This problem can be dealt with by limiting the number of times a repeated query can be propagated by a peer. This approach is similar to the one used in Gnutella [21], where the so called Time-To-Live (TTL) integer parameter is associated with each query, and it is decremented with each peer, reached by the query. When TTL becomes zero, then no further propagation takes place;
- P2P IS may change during the time of the propagation of a query. With respect to query propagation, dynamics can be modelled by removing, adding, and modifying acquaintance queries, related to the original user query. For instance, if a peer becomes (temporarily) unavailable, then, in terms of query propagation, it is equivalent to the situation when all the related acquaintance queries pointing to this peer are dropped. Therefore, we can overcome dynamics, and consider the P2P IS to be static by assuming that:
 - if a related acquaintance query is added by some peer after the user query was submitted, but before the corresponding net query reached the peer, then the net query is processed as if the acquaintance query existed before the user query was submitted;
 - if a related acquaintance query is deleted by some peer after the user query was submitted, but before the corresponding net query reached the peer, then the net query is processed as if the acquaintance query did not exist before the user query was submitted;

- if a related acquaintance query is added by some peer, and, by this time, the peer has already processed a net query, whose rewriting would include the newly added acquaintance query, then the acquaintance query is *not* considered for this net query. In other words, the effect is the same as if the acquaintance query were not added.

Thus, if the implementation of the query propagation algorithm relies on the above assumptions, then all the above discussed arguments about query propagation are also valid for the case when the P2P IS is dynamic.

The set of net queries, originated from some user query, represents the set of edges of the *query propagation graph* of the user query. The nodes of this graph are the *LISs*, queried by the net queries. The direction of edges in the graph is the direction of propagation of corresponding net queries. Technically speaking, the nodes of the graph are represented by the *portion* of the *LISs*, queried by the net queries. If two queries refer to different parts of the same *LIS*, then on the graph these parts are represented as two different nodes. Analogously, if two net queries refer to the same part of some *LIS*, then on the graph it is represented as a single node with two incoming edges. The two graphs shown in Figure 3.8 are examples of two query propagation graphs. Thus, for instance, in Figure 3.8a it is assumed that queries Q_3 and Q_4 refer to the same part of the *LIS* of peer 4. Figure 3.7 essentially represents an “opened” node with an incoming and some outgoing edges in the graph. We use the notion of query propagation graph in the following sections.

3.3.2 Query result propagation

An incoming query result from an acquaintance can be viewed as an (additional) answer to the corresponding reformulation, which is part of the rewriting of some net (or user) query. This, in turn, may lead to new results produced for the net (or user) query, if the latter is recomputed taking into account the new data. Finally, if it is a net query, then the newly computed results need to be propagated further back to the corresponding acquainted peer. As follows from this description, query results are propagated *transitively* from one peer to another.

For any user query, its *query result propagation graph* is built from the corresponding query propagation graph by inverting the direction of its edges. A fundamental difference between the two graphs is that in the query propagation graph, each edge represents a *single* query propagation, whereas, as we discuss in the following, in the query result propagation graph, each edge represents zero, one or more messages containing a non-empty result set.

Potentially, a peer may receive several answers from the same peer and for the same rewriting. Therefore, peers need to keep track of query results propagated through them in order to avoid the propagation of duplicate query answers. Below we present a query result propagation algorithm, which takes this consideration into account, and which shows in more detail how the principle of transitivity is supported. In our presentation, we explain the steps of the algorithm with Figure 3.9 (numbers on the left in the figure stand for the associated steps listed below). This figure essentially represents an “opened” node with an incoming and an outgoing edge of a query result propagation graph.

1. **Get net result:** a peer receives a query result coming from an acquaintance for some reformulation. We call these results, *net results*.

We show in Figure 3.9 net results as nr_i , where i is the identifier of the corresponding reformulation;

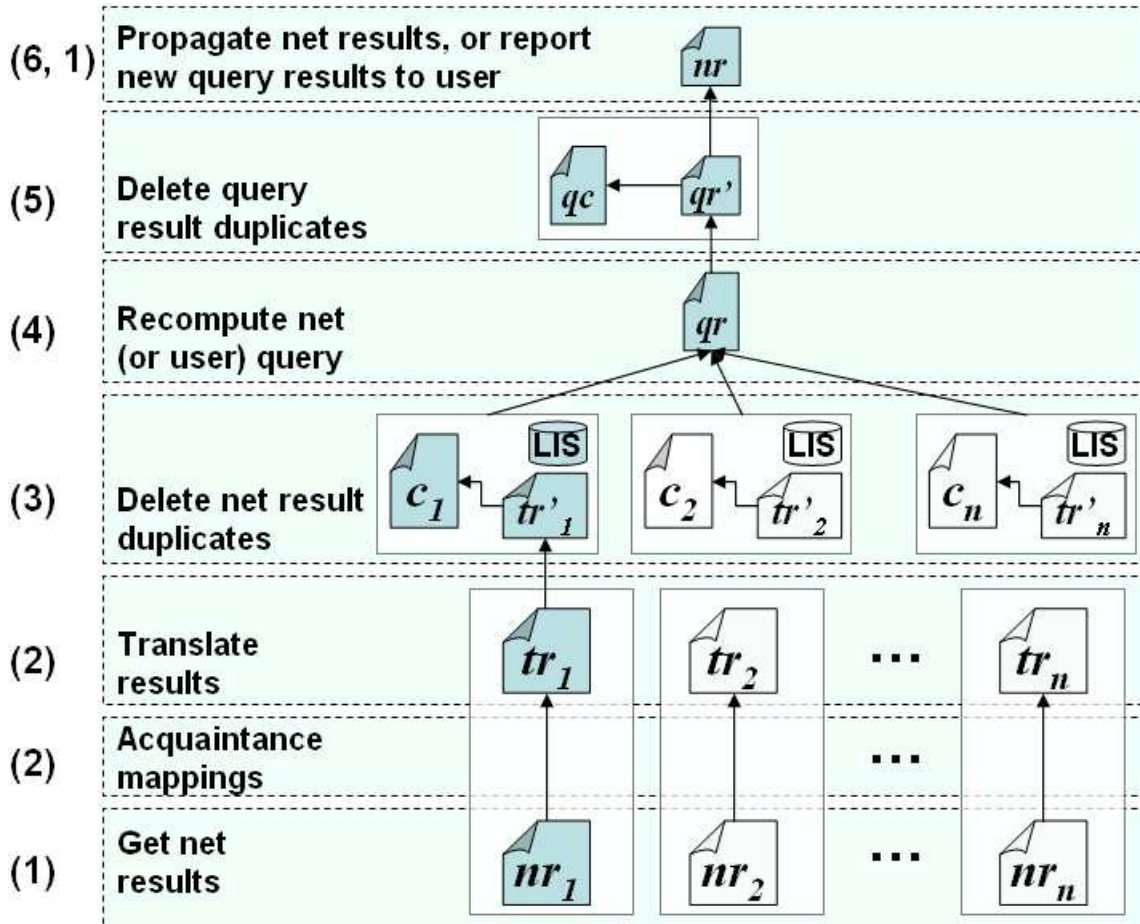


Figure 3.9: Query result propagation.

- 2. Translate results:** the peer translates net query results using appropriate correspondence rules and acquaintance mappings. After this step, net query result values correspond to the domain of the peer, and the answer is represented according to the structure of the corresponding reformulation. We show in Figure 3.9 translated net results as tr_i ;
- 3. Delete net result duplicates:** the translated result set may con-

tain instances that either are already present in the *LIS*, or that are included in the set of previously received and translated net results. These result set instances are recognized and deleted. In order to keep track of received results, each peer maintains a cache of previously received and translated net results for each reformulation (denoted by c_i in Figure 3.9). After all the duplicate instances are deleted from tr_i , the resulting set (denoted by tr'_i) is merged with c_i ;

4. **Recompute net query:** when not empty, the set tr'_i can be viewed as an (additional) answer for the corresponding reformulation, which is part of the rewriting of a net (or user) query. At this step the net (or user) query is recomputed taking into account the new data, and, as the consequence, query result set qr is produced. Note that the local data, the new net result, and net results received earlier may need to be *combined* in order to recompute the net query;
5. **Delete query result duplicates:** if the set qr , computed at the previous step, is not empty, it becomes a subject to further propagation to the appropriate acquainted peer. However, in order to avoid the propagation of duplicate query result instances, possible duplicates have to be deleted from qr . In order to do that, each peer maintains a cache of previously propagated query results (denoted by qc in Figure 3.9). After all the duplicate instances are deleted from qr , the resulting set (denoted by qr') is merged with qc ;
6. **Propagate net results:** if qr' is not empty, it is propagated to the corresponding acquainted peer (if the recomputed query is a net query), or a new (additional) answer is reported to the user (if the recomputed query is a user query).

Example 4 Recall the two peers from Example 1, and the query, propagated from peer 1 to peer 2, as it is described in Example 3. Now suppose

that peer 2 produces the following answer to query Q_2 (which becomes a net result for peer 1):

<i>title</i>	<i>film_genre</i>	<i>budget</i>
<i>Last Samurai</i>	<i>Action</i>	<i>120,000,000</i>
<i>Titanic</i>	<i>Drama</i>	<i>200,000,000</i>
<i>War of the Worlds</i>	<i>Sci-Fi</i>	<i>132,000,000</i>

Peer 1 receives the net result, and it uses the acquaintance query mappings (see Example 1) to align corresponding query elements, and it uses correspondence rules (see Example 2) to translate values for the attribute **budget**. Thus, as the result of step 2, peer 1 gets the following translated result set for Q_1 :

<i>title</i>	<i>genre</i>	<i>budget</i>
<i>Last Samurai</i>	<i>Action</i>	<i>99,173,553</i>
<i>Titanic</i>	<i>Drama</i>	<i>165,289,256</i>
<i>War of the Worlds</i>	<i>Sci-Fi</i>	<i>109,090,909</i>

Now suppose that peer 1 already has a tuple corresponding to the result instance with the “Last Samurai” movie in its LIS. This tuple is deleted from the translated result set at step 3, and the result set accordingly becomes:

<i>title</i>	<i>genre</i>	<i>budget</i>
<i>Titanic</i>	<i>Drama</i>	<i>165,289,256</i>
<i>War of the Worlds</i>	<i>Sci-Fi</i>	<i>109,090,909</i>

At step 4 of the algorithm we recompute user query Q_u taking into account the above new two tuples. Supposing that in the relation **Cast** peer 1 has a tuple associating actor Tom Cruise to movie “War of the Worlds”, peer 1 produces only one (additional) tuple as a result to the user query:

<i>title</i>	<i>genre</i>	<i>budget</i>
<i>War of the Worlds</i>	<i>Sci-Fi</i>	<i>109,090,909</i>

In this particular example, no duplicates are deleted at step 5, and the query result shown above is reported to the user. If it were a net query, then the resulting tuple would be propagated at step 6 to the corresponding acquainted peer.

Note that step 6 of the above algorithm results into the initialization of the algorithm from step 1 at the appropriate acquainted peer. Thus, the algorithm supports a transitive propagation of query results from one peer to another. Some additional considerations need to be made about the query result propagation algorithm:

- Net query results received by a peer at step 1 may yield a non-empty query result at step 5 not only for the corresponding net (or user) query, but also for other net queries, which are originated from the same user query, or even from different user queries. The intuition is that different queries may be *correlated*, in the sense that they refer to a common part of the *LSS*, and, therefore, a result for one of these queries may yield a result for other correlated queries. Peers can take advantage of it by computing also these correlated queries and propagating results to the appropriate acquainted peers. From the user view point, it results as query results delivered earlier, and, therefore, it positively affects the QoS provided to the user. On the other hand, it may lead to excessive network load and more computations at each peer;
- A peer may receive a net query result in a *loop*, i.e. this result reaches the peer as the consequence of a previous propagation from this peer of another net query result. An example of a query result propagation graph is shown in Figure 3.10. If a peer receives net query results in a loop, then it still propagates them as long as they produce a new non-empty net result. Assuming that peer *LISs* are static during the

result propagation, i.e. no new data are added, at some point a peer in the loop will reach a *fixpoint* [20], i.e. no new result will be produced. This is the moment when the query result propagation loop breaks. However, the assumption that peers maintain their *LISs* unchanged during query answering cannot be made and, therefore, loops in query results propagation can be broken similarly to the way loops in query propagation are broken in some cases. Namely, a TTL value can be associated with net result messages;

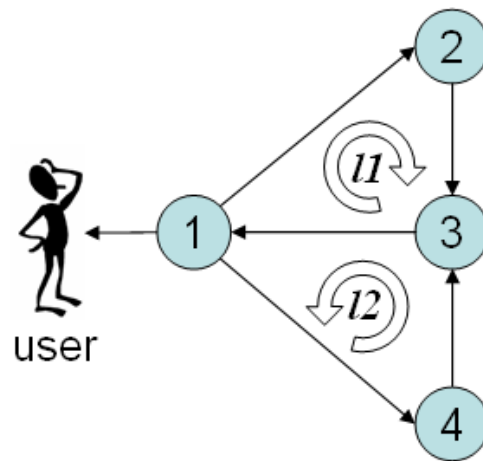


Figure 3.10: Cycles in a query result propagation graph.

- The computation of correlated queries in the presence of multiple loops in the query result propagation graph may lead to the situation, where a net result, received by a peer, leads, on average, to more than one net result, which is further propagated by the peer. This situation is dangerous as it may lead to the *flooding effect*, i.e. when the involved peers become incapable of processing all the query result messages sent within the P2P IS. Consider Figure 3.10, where two cycles ($l1$ and $l2$) in a query result propagation graph are presented. Suppose that a net result, received by peer 1 from peer 3, produces 2 net results, one for peer 2, and another for peer 4. Then, these two results are

further propagated to peer 3, and, as a consequence, they both may reach peer 1, thus, leading to the propagation of four more net results. This process can iterate and lead to the overload of peers 1 and 3. In order to address this problem, each peer can *accumulate* query results at step 6 of the algorithm until all the incoming net results are processed or until no net result is received during some (small) period of time. Only then are the accumulated net results further propagated to the corresponding acquainted peers. As from the previous example: when peer 1 receives the two net results through loops $l1$ and $l2$, it accumulates and then processes them as if it were a single net result. Therefore, it propagates only two (and not four) net results, one to peer 2 and one to peer 4;

- Intermediate peers in net result propagation can take advantage of the results passing through them. Namely, peers can tune up their coordination rules so that when a net query result instance satisfies some criteria, then it is merged with the peer's *LIS*. In this way users can enrich their *LISs* with data interesting for them, *without* explicit querying relevant acquaintance peers. For instance, it is easy to think about a scenario in which peers in a “Movies and Actors” interest group enrich their *LISs* with data about their favourite movies or actors (at step 3 of the algorithm);
- P2P IS may change during the time of the propagation of query results, which are related to some user query. Differently from query propagation, the dynamics with respect to query results propagation is conditioned by two factors: (a) changes made to the peers' *LISs*, and (b) changes made to acquaintance queries, related to the original user query. Item (a) influences what net results and when they are produced in the P2P IS, and, as it is discussed above, it may

potentially lead to an infinite query results propagation (unless it is limited by TTL). Item (b) particularly means that if a peer changes an acquaintance query, then no net results, relevant to the original user query, can be imported using it. In other words, it is equivalent to the deletion of the acquaintance query. We present this kind of situation in Figure 3.11a, where a query result propagation graph is shown. There, peer 1 changes its acquaintance query to peer 3, and, therefore, it starts considering it as irrelevant for the given user query. However, peer 1 can still import results from peer 2. Deletion of a peer can be modelled as the deletion of all the relevant acquaintance queries, originating from and received by that peer (see Figure 3.11b);

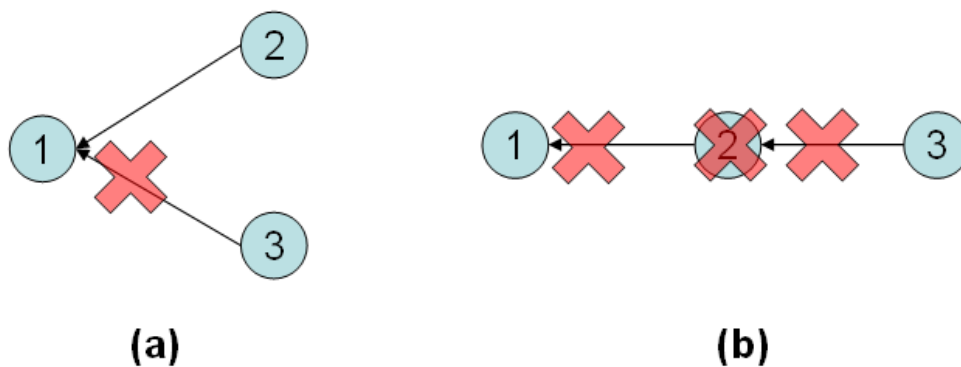


Figure 3.11: Dynamics in query results propagation.

- Cache data saved at steps 3 and 5 of the query result propagation algorithm can be stored until some disk quota is reached; until the answering of all the net queries, originated from the same user query, is complete; or until the peer has no active queries. The last option is especially relevant when a peer allows for the computation of correlated net queries, originated from different user queries. While the first option relies on local metrics, the other two depend on the identification of when the query answering is actually complete. We discuss

this problem in the next section.

3.3.3 Query answering termination

Query answering for some user query terminates when no non-empty net result is produced by any peer involved in the answering of the user query. When the system reaches this state can be shown by analyzing the query result propagation graph for some user query. In Figure 3.12 we show an example of such a graph.

Leaf nodes (i.e. nodes with no incoming edges) in the graph receive one or more net queries, and no further query propagation takes place from these nodes. In Figure 3.12 such nodes are peers 6 and 10. According to the query propagation algorithm, leaf nodes answer received queries at step 3 using their *LIS*, propagate net results back to appropriate acquainted peers, and then no further answer follows from them. Thus, after they answer, these peers can be excluded from further consideration by deleting corresponding nodes and the edges connecting them from the graph. This, in turn, may produce leaves out of other nodes (e.g. node 4 in Figure 3.12), which are also deleted accordingly.

If the initial graph is acyclic, then nodes and edges are removed iteratively from the graph until no nodes finally remain. At this point, query answering for the user query is terminated. Note that the node, corresponding to the peer where the user query was submitted, is removed last from the graph.

If the graph is cyclic, then it is reduced to a subgraph with no leaf nodes, and which contains one or more cycles. Thus, the example graph is reduced to the set of nodes 1, 2, 3, 5, 7, 8, 9, 11, and the edges connecting them. The resulting graph has five cycles: 1-3-2-1 (*l1*), 1-3-5-2-1 (*l2*), 7-9-8-7 (*l3*), 7-9-11-8-7 (*l4*), and 1-3-11-8-7-5-2-1 (*l5*).

As mentioned earlier, query results may continuously circulate in a loop

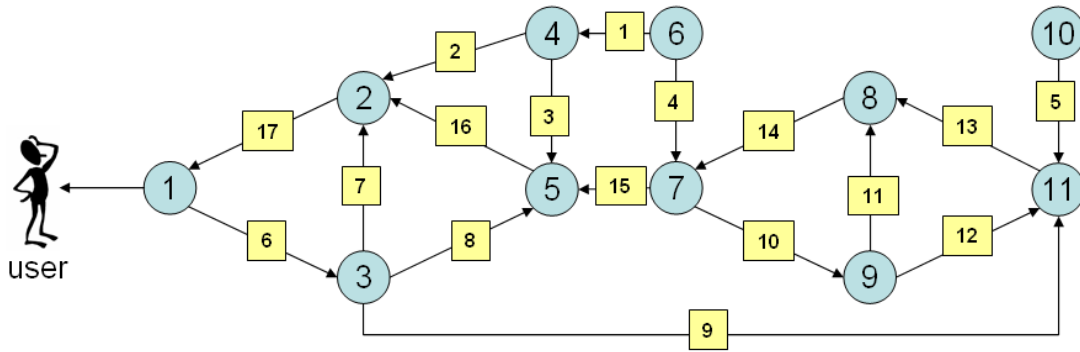


Figure 3.12: Query answering termination.

until an empty result is produced by some peer. The intuition is that the repeated loop query creates a recursive query definition and, therefore, query results, sent into the loop, may “return” to the same peer in the form of new data, so that a new query result can be computed again. Conceptually, this problem is equal to the problem of fixpoint computation for recursive datalog programs, which is well studied in the database field [20].

The problem of fixpoint computation becomes more complicated in the presence of multiple cycles, such that one cycle “brings” new data into another. This is the case, for example, when cycles have a node in common. In Figure 3.12 cycle l_2 brings data to node 3, which is part of cycle l_1 ; cycle l_5 “draws” new data from all the other cycles, as it has common nodes with all of them. However, if the P2P IS remains static, then the fixpoint will be reached in all the interrelated cycles when all the possible answers are computed. When a node in a loop reaches the fixpoint, then it is removed from the graph along with its connecting edges. It breaks the loop and makes some other nodes leaves, which are also removed as discussed earlier. Finally, when no nodes remain in the graph, query answering is complete.

The considerations presented above assume global knowledge of the query result propagation graph, which cannot be the case in a P2P IS due to the principle of locality. Therefore, peers, participating in query

answering, have to decide *locally* when query answering for a particular local net (or user) query is complete. In order to do that, peers send a “*no answer*” acknowledgement to their acquainted peers, then no (more) answer for the corresponding net queries can be produced. A peer sends such an acknowledgement when it is either a leaf node in the query result propagation graph, or when the fixpoint is reached for a peer in a loop.

Peers can identify locally when the fixpoint is reached by exploring the query (result) propagation graph, as it is suggested in [59]. The idea is that, when they send a net result onto some edge of the query result propagation graph, peers attach to the message also the identifier of that edge. Each peer generates and stores locally pseudo-unique identifiers for outgoing edges. When an intermediate peer propagates a net result, it also adds the identifier of the appropriate edge to the message. Therefore, when peers receive a net result, they also receive the path through which the result was delivered. It allows peers to build *locally* the subgraph of the query result propagation graph, which determines result delivery to these peers. Along with a net result, each peer on the path also sends a *timestamp*, which indicates when the result was computed. Peers store locally timestamps of the last computation of net results. Thus, if a net result arrives in a loop to a peer, then the peer can identify if this result comes as the consequence of the last result propagation into the loop from this peer. If this is the case and if the result produces no new net result after step 5 of the algorithm, then the peer logically closes this path. When all such paths of the subgraph are “closed” by some peer, then the peer has reached the fixpoint, and it therefore sends “*no answer*” acknowledgement onto outgoing edges of the graph.

When peers receive a “*no answer*” acknowledgement, they logically close the corresponding acquaintance queries, thus deleting edges from the graph. When all the acquaintance queries, related to some net query, are

“closed”, then the peer “closes” the net query and sends a “*no answer*” acknowledgement to the corresponding acquainted peer. The edges of the graph in Figure 3.12 are associated with numbers, which show a possible sequence of edges being “closed” locally by the peers.

For efficiency reasons, “*no answer*” messages can be sent along with net result messages. For instance, when node 6 in Figure 3.12 sends a net result to peers 4 and 7, it includes in the same message a flag that indicates that no more net results will follow from this peer.

3.4 Good-Enough Answers

The query answering algorithm presented in the previous section allows it to specify a user query at some peer, and to compute it in a fully decentralized manner involving a set of relevant peers. However, one important aspect of query answering still has to be addressed, namely, the *quality of query answers* to user queries in a P2P IS, which is fundamental to the QoS provided by the system.

The notion of *data quality* is well understood in the context of centralized ISs [131, 128, 160, 161]. Quality assessment in these approaches relies on the assumptions that data can be accessed and evaluated centrally, and that any piece of data in the system can be retrieved upon a user request. Therefore, the *quality of query answers* in these systems can be directly evaluated by assessing the *quality of the data*. Moreover, these assumptions give rise to some standard data quality dimensions such as correctness and completeness, which can be objectively evaluated given the global view of the data in the system.

In a P2P IS, the quality of query answers depends not only on data quality of particular *LISs*, but also on the *quality of mappings* (represented by acquaintance queries and correspondence rules) relating the *LISs*. In

fact, the semantics of a query can be distorted and/or data loss and misinterpretation can take place if some mappings are imperfect. Thus, for instance, the quality of query results may degrade due to their incorrect and/or incomplete translation when propagating from one peer to another.

Standard data quality metrics cannot be directly applied in the P2P setting due to its decentralized, dynamic and subjective nature. In fact, in a P2P IS query results are often incomplete (under the classical interpretation of the notion of completeness [160]) due to the fact that not all the data in the P2P IS are accessible to the peer where a user query is submitted. Furthermore, it is hard to evaluate the correctness of a query result due to the fact that each peer maintains its *subjective* view on the data. In fact, the correctness of data⁶ is referred to the extent in which the representation of some part of the real world, as it is inferred from the data stored in the IS, coincides with the *user's view* of this part of the world [160]. Thus, the same data can be considered as correct by one peer and as incorrect by another.

Another data quality metric, *consistency*, is defined as “*the degree to which the values of the attributes of an instance of a schema element satisfy the specific set of semantic rules defined on the schema element*” [138]. The problem with this metric in the P2P settings is in that different peers may define semantic rules on data differently, and, therefore, the same data can be considered as consistent in one *LIS* and as inconsistent in another.

Similar arguments can be made about other data quality metrics, such as reliability, timeliness, relevance, currency, availability, and others [162]. Thus, the standard data quality metrics have to be *reconsidered* in order to reflect the distinct characteristics of P2P ISs. Particularly, there are (at least) three kinds of unpredictable runtime factors, peculiar to P2P ISs, which influence the answer to a given user query, and, therefore, which also

⁶In the data quality literature correctness is often called accuracy [160].

influence the quality of query answers [68]:

- **Network (dependent) variance:** the P2P IS changes over time. Peers may change the data of their *LISs*, change their *LSSs*, redefine acquaintance queries and correspondence rules, finally, new peers may join the P2P IS and existing ones may leave it. Therefore, the same query submitted to the same peer but at *different times* may yield different answers and of different quality;
- **Peer (dependent) variance:** peers define acquaintance queries and correspondence rules differently from other peers. Therefore, the same query submitted at the same time but by *different peers* will result in different query propagation graphs. Consequently, the query results may be different and of different quality;
- **Query (dependent) variance:** *different queries* submitted to the same peer and at the same time may result in different query propagation graphs, and, therefore, may produce different results and of different quality.

Network dependent variance forces us to assume that data quality metrics must be evaluated in the context of the current state of the P2P IS. Peer dependent variance forces us to assume that they must be evaluated in the context of a particular peer. Finally, query dependent variance implies that data quality is also sensitive to the context of each particular query. As it can be seen, the three variances make it very challenging to develop a unified evaluation methodology for measuring the quality of query results returned in a P2P IS.

Taking into account the inapplicability of the standard data quality metrics, and the decentralized, dynamic and subjective nature of P2P ISs, we propose the novel notion of *good-enough answers* to assess the quality

of query answers in P2P ISs. As from [68], a good-enough answer is “*an answer to a user query which serves its purpose given the amount of effort made in computing it*”. There are two key points in the definition: that a query answer should serve its purpose, and that a query answer should be parametric on the initial effort. Below we discuss them in more detail:

- **Purpose-driven:** users submit queries in a specific context, giving the queries a specific purpose. Since *LISs* are developed *locally* taking into account their intended use, the *LSS*, w.r.t. which a user query is submitted, usually corresponds to the query context and serves the query purpose. However, as discussed earlier, different *LISs* are developed independently and, in most cases, they represent related concepts differently and assume a different context of their use. The latter particularly means that the different *LISs* can be heterogeneous at the application level (see Section 2.2.2). Therefore, when a query is propagated from one peer to another, it is likely to be evaluated in a different context, and to yield query results, which do not necessarily match the original query purpose. Given this, the user may receive an answer from another peer, which only to a certain extent serves the purpose of the query and corresponds to the initial context. Nevertheless, this result may still be useful for the user, and, when combined with results coming from other peers, it can better meet the user’s needs.

Suppose that the user of a peer is interested to know what the costs of flats in Trent are. The user submits an appropriate query at the peer, and the query is then propagated to its acquaintances. The acquaintances respond to the query, but, instead of the flat cost, some provide the cost of rent in suburbs *A* and *B*. However, this information is still useful for the user, as knowing the average rent-to-cost ratio allows it to estimate the cost of the flats. Since some other ac-

acquaintances provided the cost of similar flats in suburb A , the user can compute the flat rent-to-cost ratio for Trent and estimate the cost of the flats in suburb B . Therefore, even if the user query was evaluated in a different context, the combined query results still serve the query purpose.

- **Effort-driven:** as mentioned earlier, the main motivation for peers to join a P2P IS consists in their immediate access to a large pool of data sources in return for a relatively little effort of setting up a small set of acquaintances. The same holds for query answering, where the basic criterion becomes whether a query result justifies the effort, which the user made for its computation. The intuition is that users are likely to be willing to “invest” more in setting up acquaintances with the purpose of getting higher quality answers, and they will be happy with some answer if they set up acquaintances with much less care. In this respect, P2P ISs offer some advantages. First, it is enough that every peer establishes and maintains a small set of acquaintances for all the peers to benefit from collective query answering. Second, users express their queries w.r.t. the *LSSs* they own, and which they know well, and, therefore, in order to send a global query to the P2P IS, only knowledge of the local *LSS* is required. Finally, the fact that setting up acquaintances is still a cost⁷ suggests that peers should set up and change acquaintances *gradually*, reusing the same acquaintances for many user queries, thus “amortizing” their setup costs.

Requirements for being good-enough depend on the application domain. For instance, a partially incorrect result in the medical care domain may potentially lead to serious consequences (such as improper patient treat-

⁷Note that the cost of participation in heterogeneous schema-based P2P ISs is considered to be higher than the cost of participation in schema-less or homogeneous schema-based P2P systems. We discuss this issue in Section 6.3.2.

ment). A partially incorrect and incomplete result may be good-enough in the tourism domain as long as it serves the user needs (e.g. it correctly provides important data such as hotel costs).

Elaborating the previous statement further, it becomes evident that P2P ISs may not be suitable for all potential areas and applications. Critical applications with very high requirements to system stability and data quality, such as banking or health care, cannot admit any error in their operations. However, these applications can still benefit from using P2P architectures by limiting peers' autonomy and requiring constant availability of peers, high quality of mappings and data, and so on. Query answering in these systems can still be performed in the P2P fashion as described in this thesis.

There is no such notion as response time for good-enough answers. Unlike the traditional centralized ISs, where a single (and probably) correct and complete answer is provided to a user query, in a P2P IS users receive a continuous sequence of query answers until the query answering is complete. In such settings, the first few results may be already good-enough for the user, or a thorough query answering has to be performed before the overall answer becomes good-enough.

For an answer to be good-enough, it is not absolutely necessary that it satisfies all the constraints specified in the query. In fact, results which are not an exact match, but which are a “close” match to the requirements specified in the query, can still serve the purpose of the user. Therefore, query relaxation techniques can be used for query processing in P2P ISs (see [42] for an example of query relaxation for relational DBMSs, and [150] for that of ontology-based systems). For instance, when looking for a flat with a certain number of rooms and of a certain cost, the user may also want to consider flats, whose cost is slightly higher, but which have one more room.

The subjectivity dimension discussed above can be partly modelled with the computational notion of trust [108]. In fact, users tend to consider data and information from trustworthy sources more important than those coming from unknown or untrustworthy ones. As suggested in Section 3.2.2, acquaintance links can be labelled by trust values, which show how much one peer trusts answers for some query computed by an acquaintance. These trust values can be combined in the query answering algorithm in a transitive distributed manner in order to compute a relative value of user's trust in an answer, as we suggest to do in [166]. These relative values can serve as a subjective metric for evaluating good-enough answers in P2P ISs.

3.5 System Requirements and Logical Architecture

The P2P information management model proposed in Section 3.2 forms the core of the P2P IMS. However, for the P2P IMS to become a viable technology, some system- and operation-level requirements have to be addressed. As discussed earlier, each peer runs a copy of the P2P IMS. Below we summarize the requirements for each copy of the P2P IMS software:

- **Symmetric design:** As from the definition of P2P, each peer has equivalent functional capabilities. In terms of system design, it means that all the copies of the P2P IMS must be absolutely symmetric for all the peers at the level of logical architecture;
- **Backward compatibility:** the P2P IMS must be pluggable on top of various existing data repository systems (e.g. databases, ontology-based data repositories). Apart from the fact that it favours design autonomy of peers, it also provides for backward compatibility with

existing systems which can potentially benefit from being part of a P2P IS;

- **System independence:** the P2P IMS must be capable of being run potentially from any computational device (e.g. PC, PDA), on any system platform (e.g. Windows, UNIX), and support multiple transport protocols. This requirement favours design autonomy of peers, and addresses the problem of system-level heterogeneity;
- **IP independence:** peer addressing scheme, used to identify peers in the P2P IS, must be largely independent from the physical IP addresses of the peers. This requirement favours participation autonomy of peers and supports persistent naming of peers in the presence of dynamics (when a peer changes its IP address);
- **Resource discovery:** since the P2P is fully distributed and dynamic, and due to the principle of locality, the P2P IMS must provide efficient and effective means for resource discovery in the P2P IS. The resources are interest groups, peers, data stored at peers, and so on.

In Figure 3.13 we report a logical architecture of a P2P IS⁸. Each peer in the P2P IS consists of a *LIS*, a *LSS*, and a copy of the P2P IMS software. The *LSS* represents the *shared* part of the *LIS*, which allows it to respect the association autonomy of peers. Decoupling the P2P IMS from the information source allows it to respect the backward compatibility requirement. The *LIS* can be an optional component (therefore this component has a dashed border in the figure), but the *LSS* must be present as user queries as well as acquaintance queries are expressed w.r.t. the *LSS*. Note that since all peers conform to this logical architecture, the symmetric design requirement is respected.

⁸The architecture was originally reported in [31], and then further elaborated in [71].

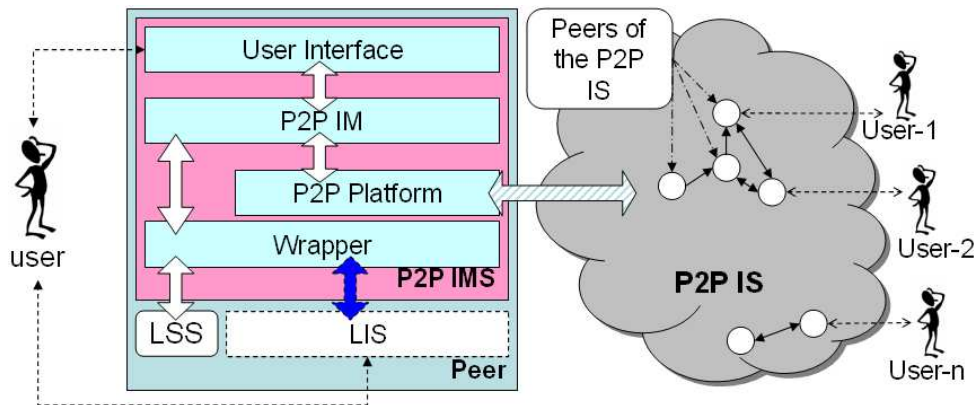


Figure 3.13: A logical architecture of a P2P IS.

The P2P IMS consists of a user interface, a P2P information management (P2P IM) component, a P2P platform, and a wrapper. The user interface component implements all the necessary interaction procedures between the user and the P2P IS. P2P IM implements the information management model. It connects to the *LIS* through the wrapper, and it communicates with other peers through the P2P platform.

The wrapper translates from the data model specific to the *LIS* to some *common data model* (CDM) used within the P2P IMS. It allows it to address the heterogeneity problem at the data model level, and it also favours the backward compatibility requirement. The user can access the *LIS* in two ways: (1) through the P2P IMS, when the user wishes to interoperate with other peers on the P2P IS; and (2) through the native interface of the *LIS*, when only local operations are desired. In the former case the user queries are expressed in the CDM. The latter option supports backward compatibility. While the *LIS* is organized according to the specific local data model, the *LSS* is represented in the CDM.

Note that the arrow connecting the wrapper with the *LIS* has a different notation from the one connecting it with the *LSS* and from the ones connecting the components within the P2P IMS. This is because the arrow

denotes a *LIS*-dependent communication. The arrow that connects the P2P platform with the network has yet another notation, as it represents the P2P platform-dependent communication.

The P2P platform is responsible for providing a medium for peers' intercommunication and for resource discovery. It should therefore support multiple networking protocols and be largely IP-independent. Since resource discovery is dependent on the underlying information management model, and, particularly, on the metadata specific to the model, these metadata are partly encoded into the discovery machinery of the P2P platform. Finally, as the notion of interest group is network-centric, the P2P platform partly implements it, especially by providing interest group formation and access mechanisms.

Chapter 4

Architecture and Implementation

In this chapter we provide the details of the implementation of a prototype of a P2P IMS. The prototype is built on top of the information management model discussed in the previous chapter. We discuss the rationale behind our choice in selecting the underlying P2P platform and data management technology. Finally, we discuss the properties of the solution and compare them to those of conventional IMSs.

The chapter structure is as follows. In Section 4.1 we motivate our choice of a P2P platform, on top of which we build our IMS. In Section 4.2 we motivate our choice of the technology, used to implement the data management in the P2P IMS. In Section 4.3 we discuss the system architecture of the implemented P2P IMS. In Section 4.4 we discuss the details of the implementation, provide screenshots and a description of the implemented prototype. In Section 4.5 we discuss the properties of the proposed solution in the light of the analysis dimensions introduced in Section 2.2.

4.1 P2P Platform selection: JXTA

There are various P2P platforms available for development of P2P applications, such as JXTA [1], JADE [29], PASTRY [136], and many others¹. Among all the P2P platforms we have chosen JXTA as the core platform for the implementation of the P2P IMS. JXTA allows it to respect some of the requirements discussed in Section 3.5. We summarize JXTA advantages as follows:

- it uses IP-independent naming space to address peers and other resources, which allows it to satisfy the IP-independence requirement;
- it is designed to be compatible with various computational devices, it can be run potentially from any system platform, and it supports multiple transport protocols (e.g. TCP/IP, HTTP, Bluetooth). Thus, it allows it to satisfy the system-independence requirement;
- it implements a complete machinery for efficient discovery of resources. Thus, it allows it to satisfy the resource discovery requirement;
- it provides powerful tools for the implementation of some of the information management model notions, such as interest groups.

The JXTA software architecture is shown in Figure 4.1. It has three layers (the description is taken from [111]):

- The *platform layer*, also known as the JXTA core, encapsulates minimal and essential primitives that are common to P2P networking. It includes building blocks to enable key mechanisms for P2P applications, including discovery, transport (including firewall handling), the creation of peers and peer groups, and associated security primitives;

¹See [50] for an overview of existing P2P platforms and their comparative analysis.

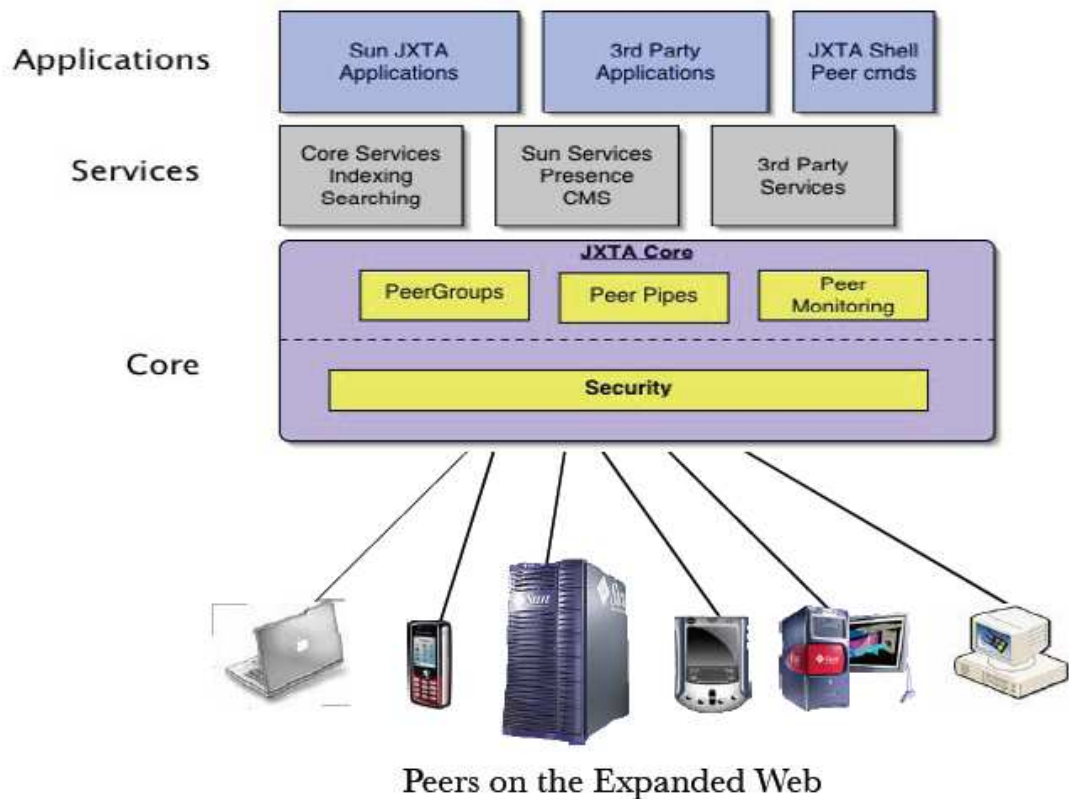


Figure 4.1: The JXTA software architecture. Source: [111].

- The *services layer* includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in the P2P environment. Examples of network services include searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services;
- The *applications layer* includes implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P Email systems, distributed auction systems, and many others.

4.1. P2P PLATFORM SELECTION: JXTA

Note that JXTA allows for the definition of 3rd party services and the development of 3rd party applications (see Figure 4.1). This is where we incorporate the functionality of the P2P IMS as it is discussed in Section 4.4.1. Below we briefly discuss the main JXTA concepts, which are relevant for the implementation of the P2P IMS. Readers interested in a complete specification are referred to [111].

JXTA provides an open set of *protocols* which allow it to build P2P applications. *JXTA peers* are devices which implement one or more JXTA protocols. JXTA-powered applications, amongst other things, can: create groups of peers, locate peers on the network, create messages, where a message can carry arbitrary type of data (e.g. images, code, query results), create communication links (called *pipes*) with other peers and send messages onto pipes. The pipe *endpoints* are referred to as the *input pipe* (the receiving end) and as the *output pipe* (the sending end). Pipe endpoints correspond to available peer network interfaces (e.g. TCP port and IP address). JXTA allows the definition of a set of *services* that a peer makes available for other peers. Services fall into two categories: *peer services* and *peer group services*. Peer services are provided by single peers, and, should a peer fail, the service also fails. Peer group services are provided by a collective set of peers, and, should a peer fail, the service does *not* fail, assuming that there are other peers providing this service. JXTA defines the core set of services necessary for a fully-functional operation of a peer. Some of them are: *Discovery Service*, *Membership Service* and *Pipe Service*. The Discovery Service allows peers to locate and publish information on the network. The Membership Service is used by current members of a peer group to reject or accept a new group membership application. The Pipe Service allows peers to create pipes with peers from the same group.

Peer groups may include (a subset of) the core services as well as *custom services*. Custom services allow for the creation of peer groups, which

provide their peers with desirable functionality. A JXTA peer group is a set of peers which agree on the common set of services. All *network resources* in JXTA (i.e. peers, peer groups, pipes, etc.) are described by *advertisements*, which are XML documents. Peers can publish, discover and use advertisements (e.g. create a pipe from an advertisement).

4.2 Technology Selection: Databases

Among all the possible technologies that could be used to implement the CDM of the P2P IS, we have chosen the (relational) database technology to demonstrate how the ideas worded in this thesis can actually be implemented in practice. There are several reasons for this choice:

Maturity: The database technology has been around for several decades; it is a well studied and matured field. The spectrum of database systems includes a variety of products (see Section 2.3); the semantics of query languages and data manipulation is well understood. DB-based P2P ISs can benefit from reusing the achievements of the database technology; and, on the other hand, they can also be compared to the state of the art systems in the database domain;

Wide usage: There are lots of databases on the web. According to some estimates, the size of the “deep” web is about 500 times larger than the size of the “surface” web, and, more than a half of the deep web content resides in databases [30]. Apart from this, nowadays almost each enterprise and small company has a database to manage its corporate data. Databases have been and remain one of the most growing IT market areas. For instance, according to the Bureau of Labor Statistics, the need for database administrators will grow by 74.8% from 2002 to 2012 [123]. All these facts provide solid grounds for a variety of applications for DB-based P2P ISs.

Knowledge supportive: database technology provides a comprehensive

support for the notions of raw data, information, and knowledge. In terms of a relational DBMS: raw data are attribute data values stored in a table; the database schema makes information from the raw data by introducing an explicit semantics (in the form of attribute and schema names); and, the ability of a DBMS to combine raw data and schemata (e.g. by means of a query) provides technological support for knowledge acquisition and management;

The underlying technology, used to implement the CDM, defines the syntax and semantics of queries that can be posed in the P2P IS. We use the *conjunctive query* notation to represent user and net queries. Conjunctive queries can express select-project-join queries, and they have the following syntax [153]:

$$r(\bar{X}) : -r_1(\bar{X}_1), \dots, r_i(\bar{X}_i), c_{i+1}(\bar{X}_{i+1}), \dots, c_n(\bar{X}_n)$$

where r , r_1, \dots, r_i and c_{i+1}, \dots, c_n are predicate names. Predicate $r(\bar{X})$ is called the *head* of the query, $r_1(\bar{X}_1), \dots, r_i(\bar{X}_i)$ are the *relation subgoals*, and $c_{i+1}(\bar{X}_{i+1}), \dots, c_n(\bar{X}_n)$ are the *comparison subgoals* of the *body* of the query. Comparison subgoals stand for arithmetic comparisons, such as $<$, \leq , $=$, \neq . Tuples $\bar{X}, \bar{X}_1, \dots, \bar{X}_i$ contain variables, whereas tuples $\bar{X}_{i+1}, \dots, \bar{X}_n$ contain either constants or variables from the relation subgoals. Variables from $\bar{X}_1, \dots, \bar{X}_n$ are called *body variables*, and variables from \bar{X} are called *head variables*. We allow only safe queries, i.e. we require that all the head variables be also body variables.

The head of a conjunctive query stand for the answer relation, and the relation subgoals refer to the relations in the underlying database schema. Common variables in relation subgoals define join predicates for the corresponding relations. It is not necessary that variable names in the relation subgoals correspond to actual attribute names in the schema relations.

Given that a relation subgoal and the corresponding schema relation both have the same arity, the subgoal variables are mapped to the actual attribute names according to their order in the subgoal. Conjunctive queries can be reformulated into equivalent SQL queries. Let us consider an example.

Example 5 *Recall the relations of peer 1 from Example 1. Suppose that the user is looking for action movies produced after 2004 and starring Bruce Willis. This query can be expressed in the conjunctive form:*

*Answer(title, year) : –Movie(title, year, genre, budget),
Cast(title, actor, gender), genre = ‘Action’,
year > 2004, actor = ‘BruceWillis’*

This query can be rewritten into an equivalent SQL query (for the schema of peer 1):

```
select title, year  
from Movie as m, Cast as c  
where m.title=c.movie_title and m.genre=‘Action’  
and m.year > 2004 and c.actor_name=‘Bruce Willis’
```

4.3 System Architecture

We materialize the logical architecture of a P2P IS, shown in Figure 3.13, by instantiating the P2P platform component with JXTA, and the generic wrapper component with a database wrapper. The LSS is represented as a relational database schema. We show the instantiated logical architecture in Figure 4.2. In this section we discuss the internals of the P2P IM and JXTA components.

Readers, interested in architectural and functional details of database wrappers, are referred to [134, 84]. Noteworthy, protocols to provide SQL

4.3. SYSTEM ARCHITECTURE

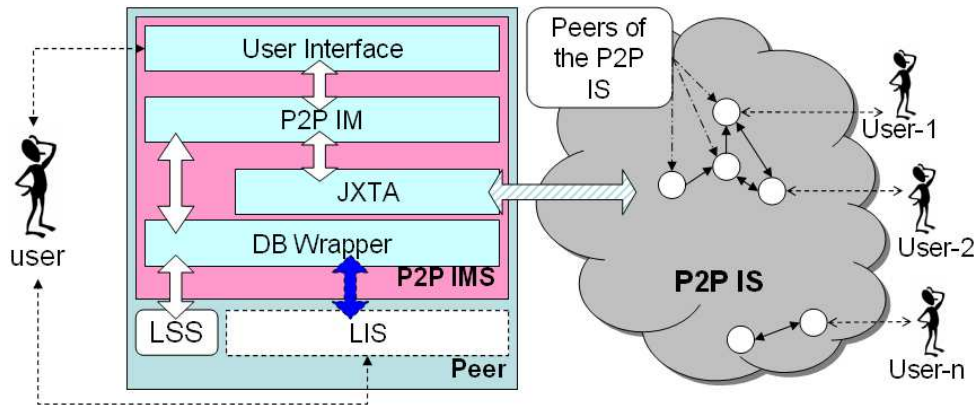


Figure 4.2: A logical architecture of a P2P IS instantiated with JXTA and a DB wrapper.

interface to non-SQL data were standardized [16], and nowadays they are used in various industrial data integration systems (e.g. see [78]). Note that in order to perform data manipulation (e.g. joins), a relational database machinery needs to be installed at each peer unless the LIS of the peer is a database. In the following we assume that peers support select-project-join operations on relational data.

In Figure 4.3 we show an expanded logical view of the P2P IMS and JXTA components. Rectangles with rounded corners stand for data repositories, and normal rectangles stand for functional modules. Arrows in the figure show only some principal connections between the different components. The P2P information management model is largely implemented in P2P IM. The Query Manager and Result Manager modules implement the query answering algorithm. The Update Manager module implements update propagation among database peers in the P2P IS. Update propagation is not the core focus of this thesis; the interested reader is referred to our work on this topic reported in [55, 56]. DB Manager maintains a connection with the LIS through the wrapper.

Consider the JXTA component. We extend the JXTA peer advertisement with LSS information. Thus, when a peer is discovered, its LSS is

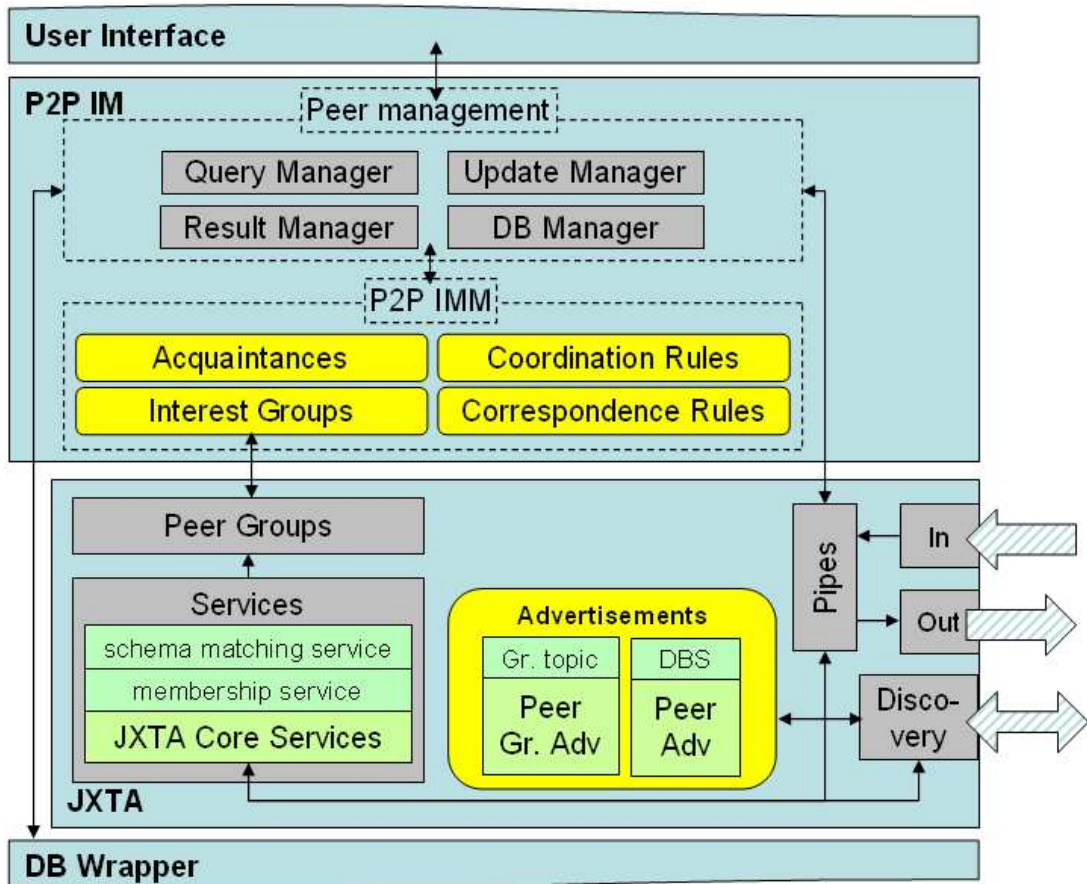


Figure 4.3: A logical architecture of the JXTA and P2P IM components.

available for the other peers to decide if they want to make an acquaintance with the given peer. Analogously, we extend the interest group advertisement with topic information, so that when they discover new interest groups, peers can evaluate their interest in joining these groups on the base of the topic information.

We extend the core JXTA services by a schema matching service [65], which is used by peers for generating acquaintance queries. This service is also called from the extended JXTA membership service, which evaluates the relevance of peers' schemata to the interest group topic as described in Section 3.2.1. The set of the core and custom services constitutes the set of

services provided by JXTA peer groups. Together with interest group topic and hierarchy information (managed within the corresponding component in P2P IM), JXTA peer groups form the basis of the implementation of interest groups in the P2P IS.

In the discovery of peers and interest groups we completely rely on the JXTA resource discovery mechanism. We also reuse JXTA pipe mechanism for establishing connections between peers, and for sending queries, query results and updates among them.

4.4 Implementation

4.4.1 Information management model

The notions of the information management model, which are most affected by the technology implementing the CDM, are acquaintances and correspondence rules. In this case, correspondence rules define correspondences *only* between attribute values of schema relations of different peers. Acquaintances are worthy of more attention.

We represent acquaintance queries as conjunctive queries. In fact, conjunctive queries are a compact way of representing acquaintance queries [69]. Namely, the head of the query refers to the local definition, and the body refers to the remote definition of an acquaintance query. The mappings of the acquaintance query are encoded in the set of head variables, which appear also in the body of the query. Let us consider an example.

Example 6 *Recall peer relations from Example 1. Peer 1 might have specified the following conjunctive query as an acquaintance query for peer 2:*

*Movie(title, genre, budget) : –
Film(title, director, budget); Genre(title, genre)*

The head of the conjunctive query refers to **Movie** relation at peer 1, and the two relation subgoals refer to the relations of peer 2. The mappings are defined by the conjunctive query as follows: the first attribute in relation **Movie** of peer 1 is mapped to the first attribute of relation **Film** and to the first attribute of relation **Genre** of peer 2. The other attributes are mapped analogously.

Note that we have a rather limited local definition of acquaintance queries. The reason is that it allows for a simpler and more flexible query rewriting.

4.4.2 Query answering algorithm

In this section we specify some technical details of the query answering algorithm reported in Section 3.3, which are dependent on the technology implementing the CDM.

As it was mentioned earlier, both received user and net queries (step 1) are represented in the conjunctive query notation. Query rewriting (step 4) is based on the standard query rewriting techniques [80]. In Figure 4.4 we show the base relations (shown as $R_1 \dots R_n$) of a peer; the queries, submitted to the peer, which are relevant to some user query (shown as $Q_1 \dots Q_m$); and the relation subgoals of the queries (the subgoal of query Q_i , that refers to relation R_j , is shown as S_{ij}). Arrows from the relation subgoals indicate possible further propagation of reformulated queries. Correlated queries are those, which have a relation in common. In Figure 4.4 queries Q_1 and Q_2 as well as queries Q_1 and Q_m are correlated. Thus, for instance, a net result for relation subgoal S_{11} , which imports new tuples to relation R_1 and which may produce an answer for Q_1 , may produce a result also for query Q_m , as the computation of the last query also depends on the contents of relation R_1 .

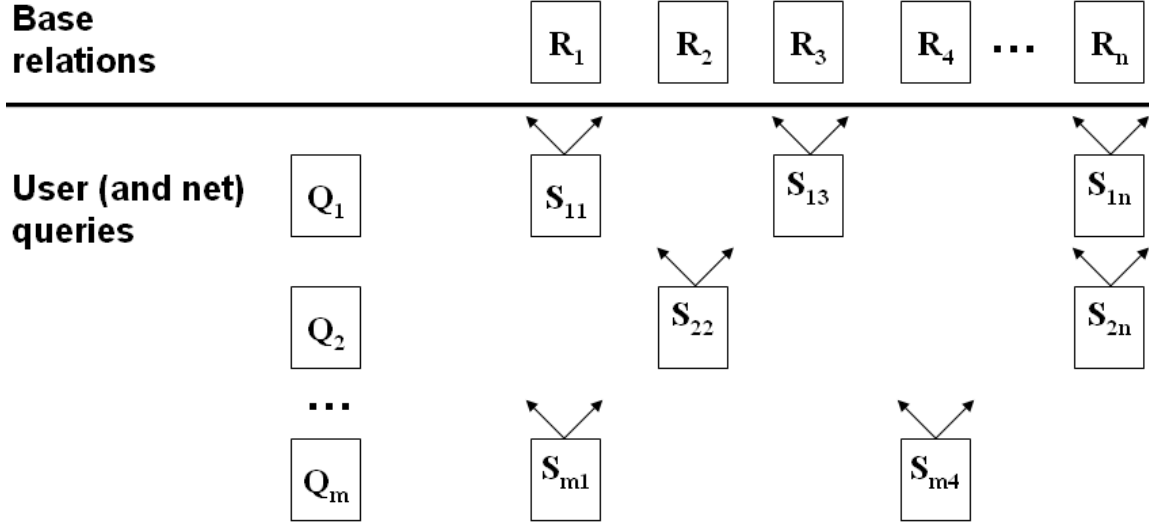


Figure 4.4: Query propagation and correlated queries in DB-based P2P ISs.

Reformulations of repeated queries can be propagated until there is a previously propagated net query, such that the reformulation is contained in it. Containment check is performed on the two conjunctive query definitions [152]. Below we show an example of the situation when repeated queries are propagated several times by the same peer using the same acquaintance.

Example 7 Consider the two peers and their relations shown in Figure 4.5. Attributes of all the relations are defined on the domain of integer values. Peers are acquaintances of each other with respect to the following acquaintance queries:

$$AQ^{12} : A(a, b) : -B(a, z), C(z, b) \qquad AQ^{21} : C(c, d) : -A(d, c)$$

Suppose that the user of peer 1 submits the following query:

$$Q_u(a, b) : -A(a, b), b < 3$$

This query is reformulated and sent to peer 2 as the following query:

$$Q_1(a, b) : -B(a, z), C(z, b), b < 3$$

Peer 2 reformulates Q_2 and sends the following query back to peer 1:

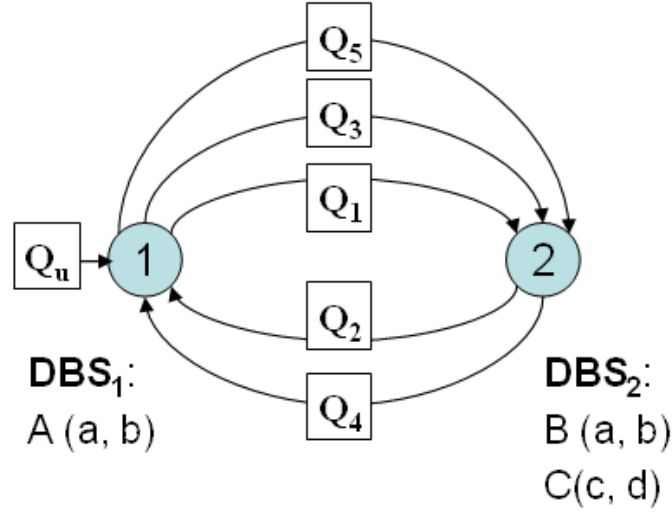


Figure 4.5: Multiple repeated loop queries.

$$Q_2(c, d) : -A(d, c), d < 3$$

The rewriting of Q_2 at peer 1 produces the following query:

$$Q_3(a, b) : -B(a, z), C(z, b), a < 3$$

Note that Q_3 is not contained in the previously sent to peer 2 query Q_1 .

Therefore, Q_3 is propagated to peer 2, where it is reformulated into:

$$Q_4(c, d) : -A(d, c)$$

Query Q_4 is not contained in the previously sent to peer 1 query Q_2 . Therefore, Q_4 is propagated to peer 1, where it is reformulated into:

$$Q_5(a, b) : -B(a, z), C(z, b)$$

Query Q_5 is still propagated to peer 2, where it is reformulated into:

$$Q_6(c, d) : -A(d, c)$$

Note that query Q_6 is contained in query Q_4 (or, more precisely, the two queries are equivalent), which was previously propagated from peer 2.

Therefore, peer 2 does not propagate Q_5 any further.

Thus, in this simple example, one user query leads to five net queries propagated between the two peers. The resulting query propagation graph is shown in Figure 4.5.

4.4. IMPLEMENTATION

Technology-dependent aspects of the query result propagation algorithm are: (a) query recomputation, given a new net result (step 4); and (b) the organization and use of net and query result caches (steps 3 and 5). In the following we describe them on the example of Figure 4.6.

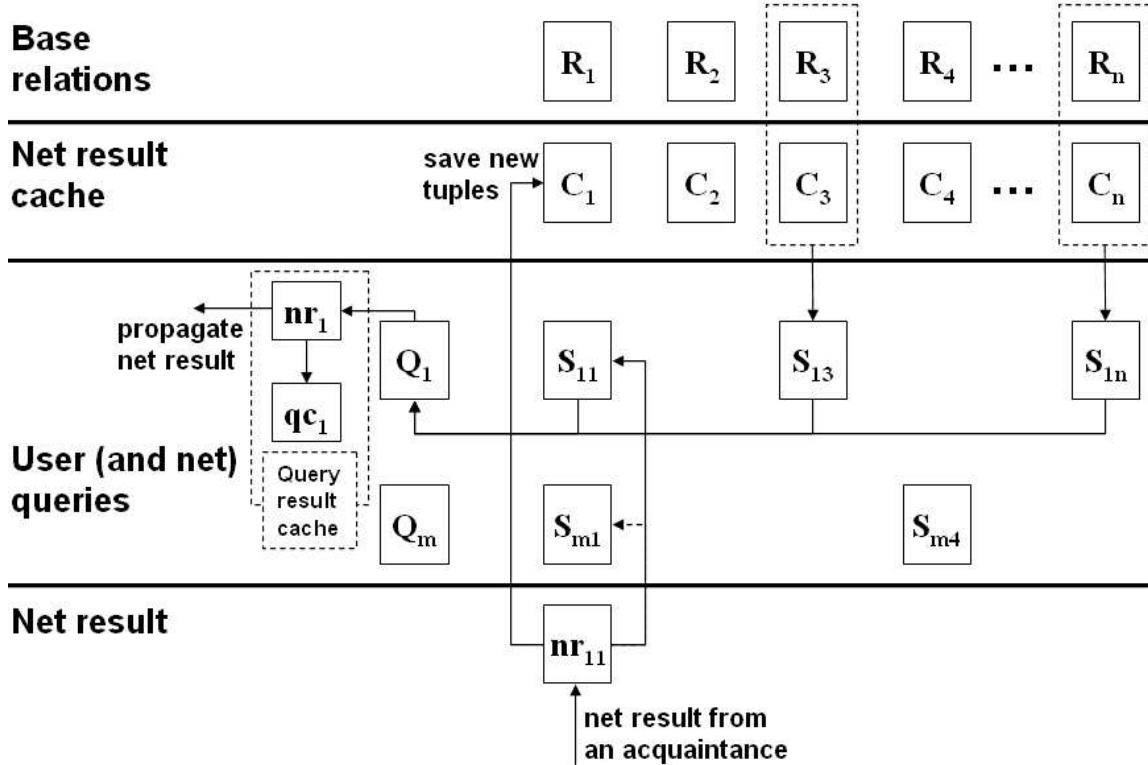


Figure 4.6: Results caching and computation of correlated queries in DB-based ISs.

We implement the caches as temporary tables in the database. Particularly, we create one such table for each base relation of the peer (shown as $C_1 \dots C_n$ in the figure). The temporary tables store the set of net results received for each relation from acquaintances during the answering of a single user query (i.e. the cache is not shared among different user queries). All the temporary tables are deleted automatically when the connection from the P2P IMS to the underlying database is dropped.

Suppose that a new net result arrives for relation subgoal S_{11} . The new results are put into a temporary table in the database (shown in the figure

as nr_{11}). We first delete from nr_{11} all the tuples which are present in R_1 or C_1 . The remaining tuples in nr_{11} are then added to C_1 . By doing this, we perform step 3 of the algorithm.

For the computation of Q_1 we must instantiate data for all its relation subgoals. We use nr_{11} as the data for subgoal S_{11} , and we compute the union of corresponding relations and caches as the data for the other subgoals. For instance, the data for subgoal S_{13} are computed as the union of R_3 and C_3 . In this way we take into account the results received earlier for the other subgoals (and, therefore, we take the union of the relations and caches), and we avoid redundant computation of tuples in the join (and, therefore, we consider only nr_{11} for S_{11} and not the union of R_1 and C_1). Finally, Q_1 is computed as the standard select-project-join operation performed on the relation subgoals and their instantiated data (i.e. we perform step 4).

The result of the computation of Q_1 is added to a temporary table nr_1 . As the next step we need to delete those tuples from nr_1 , which were computed earlier for Q_1 . For this reason we maintain another cache table qc_1 , which stores the set of previously computed query results. Then, those tuples of nr_1 , which are also present in qc_1 , are deleted from nr_1 . Finally, the remaining tuples of nr_1 are added to qc_1 . This is how step 5 is implemented.

As discussed in Section 3.3.2, peers can accumulate query results before sending them to their acquainted peers. In this case, the results produced from the recomputation of Q_1 are continuously added to nr_1 (and, accordingly, to qc_1). Then, the set of accumulated results stored in nr_1 is sent to the corresponding acquainted query, for which the result appears as a new single result set for some relation subgoal of some query.

Note that a net result for the relation subgoal S_{11} of query Q_1 can be also seen as a new result for the subgoal S_{m1} of query Q_m , which can be

also recomputed as it is discussed in Section 3.3.2.

4.4.3 Prototype Screenshots and Description

In Figure 4.7 we show several P2P clients running on the same machine. In Figure 4.8 we show the principal UI window, where the user can launch the peer discovery routine, see the list of discovered peers, establish pipes with acquaintances, and perform some other operations. For testing purposes, each peer can read acquaintance information from a file (which includes IDs of acquainted and acquaintance peers, and acquaintance queries) for all the peers, and propagate this information in the form of a JXTA advertisement to all the peers in the P2P IS. When peers receive such an advertisement, they save the acquaintance information related to them and set up pipes with their acquaintances, as shown in Figure 4.9.

In Figure 4.10 we show the tab, where the user can set up and configure a wrapper². Particularly, it is possible to specify the details of a connection to a database, and to extract and save the database schema. At the moment the wrapper supports only the family of relational databases in the role of the peer's *LIS*, and only the whole schema can be shared with other peers, i.e. no support for association autonomy is currently implemented.

On a separate tab, the user can specify his/her queries in the conjunctive query form as shown in Figure 4.11. In the same window the user can browse the *LSS*, and see the history of the submitted queries. For each of the submitted queries, the user can open a new window and see the query results received by that moment. In this window the user can find detailed information about the query, such as its SQL equivalent, starting and execution times, etc. An example of such a window is shown in Figure 4.12. Analogously, each intermediate peer in query answering can browse net queries and see their accumulated results in a similar window.

²We thank Alexander Ivanyukovich for his help in developing the wrapper component.

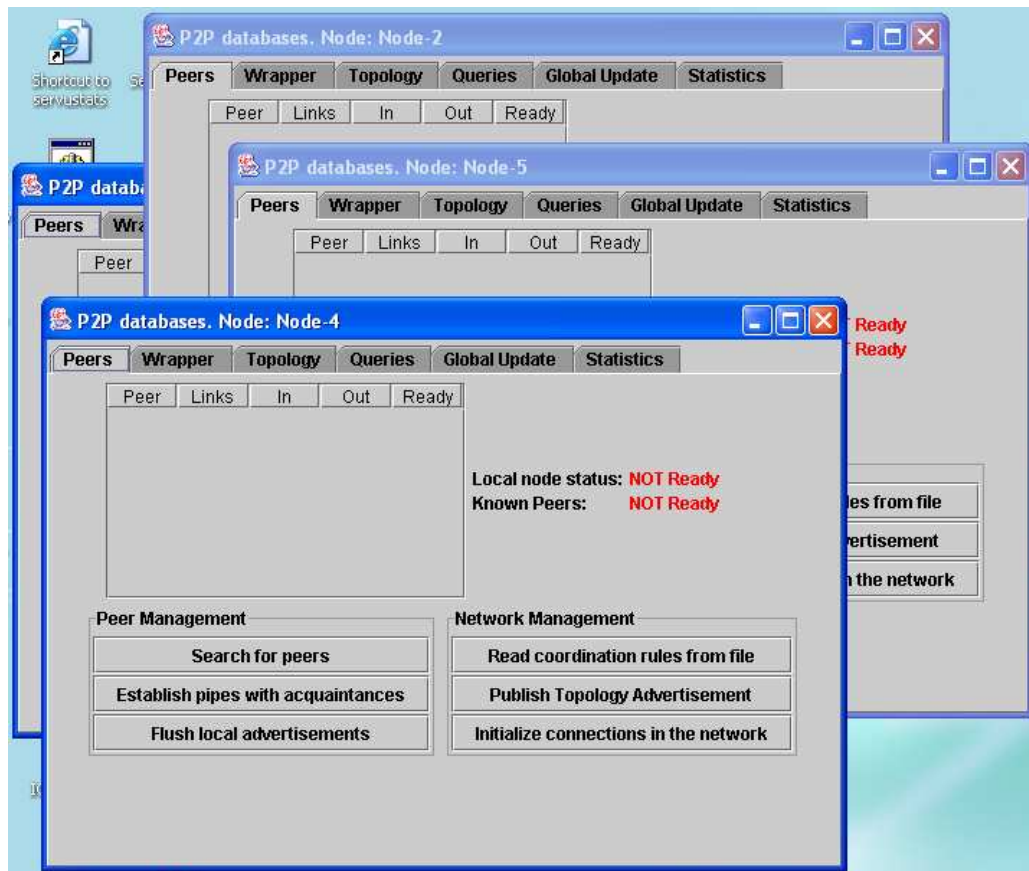


Figure 4.7: Running peers.

The processing of net queries at the intermediate peers happens seamlessly on the background, without any notification to the users of these peers.

In Figure 4.13, we show a special tab, which is devoted to the processing of updates. Any peer can require to update its *LIS* using data from its acquaintances; they, in turn, update their *LIS*s by requesting data from their acquaintances, and so on. For the details of the algorithm and for more detail of the implementation refer to [55, 56]. In this window the user can see the details of the requested updates, such as starting and execution times, number of received updates, and so on.

In Figure 4.14, we show a special tab, which is used by the user for the exploration of the network topology. Particularly, the user can run a fully distributed topology discovery algorithm, which computes the set of

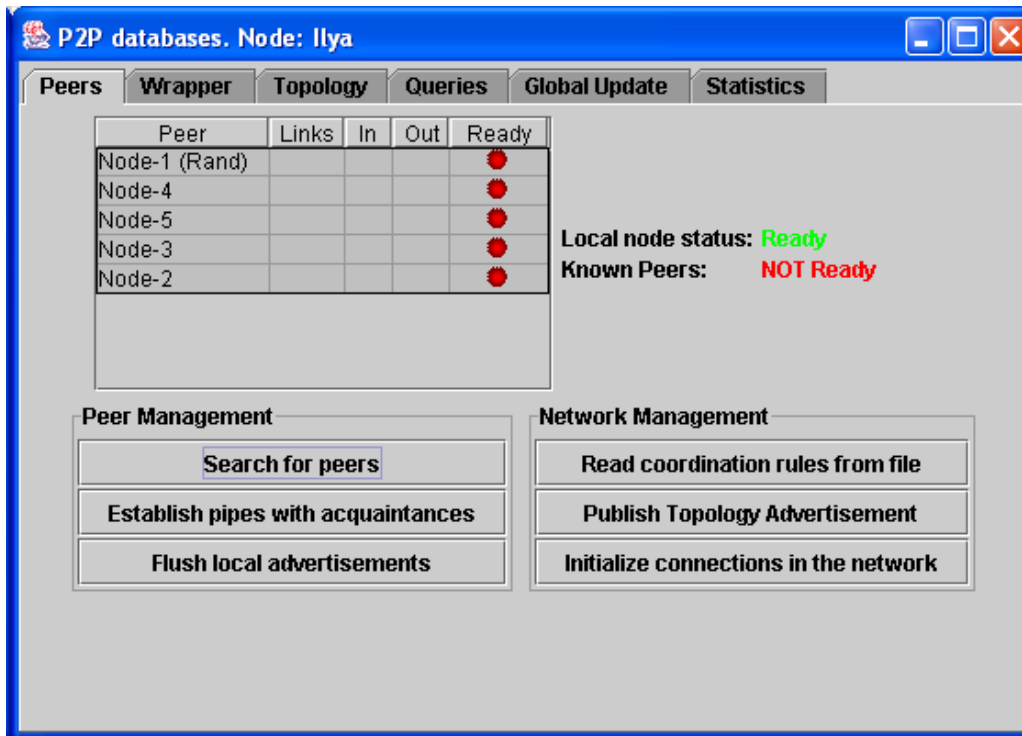


Figure 4.8: Discovery of peers on the network.

all the paths which originate from the given peer, and which are formed from acquaintance queries. Optionally, the user can specify a user query and, in this case, the algorithm computes the set of paths in the query propagation graph of this query.

The developed software allows it to accumulate statistical data at each peer. These data include number of queries, query results, and updates processed at each peer, average execution times, the volumes of transmitted data, and so on. Again, for testing purposes, each peer can request to collect statistical data from all the peers in the P2P IS, and provide a summarized report to the user. An example of such a report is shown in Figure 4.15.

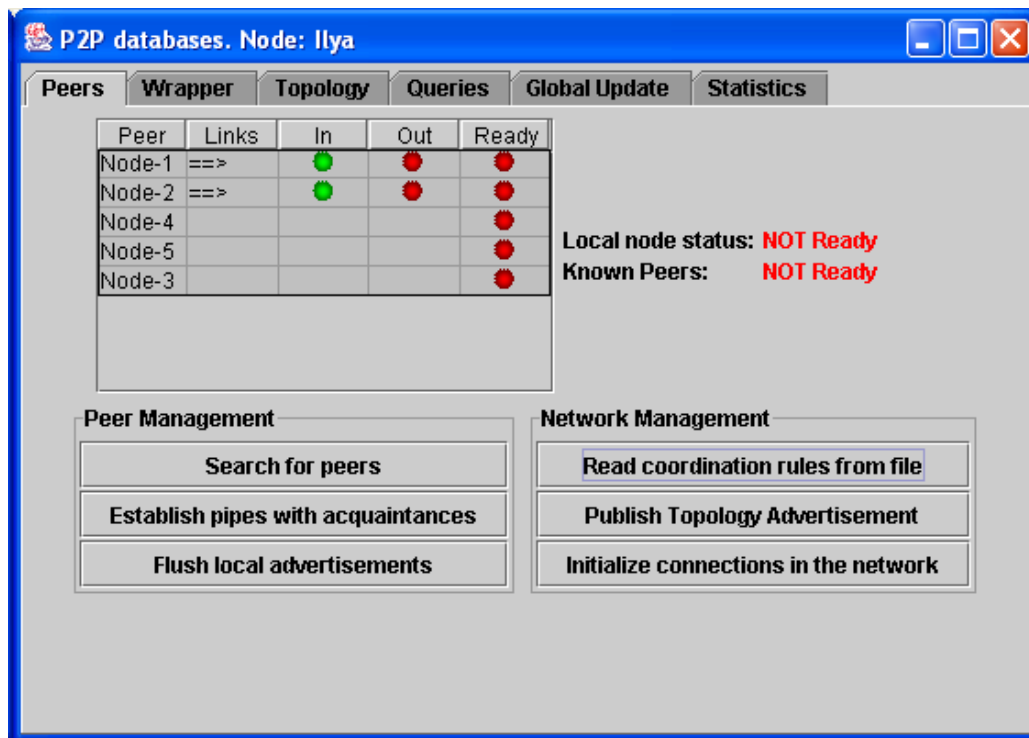


Figure 4.9: Setting up acquaintance queries and pipes.

4.5 Properties of the Solution

In this section we discuss properties of the proposed solution in the light of the analysis dimensions introduced in Section 2.2. We summarize the properties in Table 4.1, in which we also duplicate corresponding summaries for database- and ontology-based information systems. It allows us to compare all the ISs, discussed in this thesis, in one single spreadsheet. Apart from this, we show in what respect P2P database systems are different from their closest ancestors, discussed in Section 2.3.

Autonomy: as discussed earlier, peers are largely independent in how they design their LISs and what data they store, i.e. design autonomy of peers is respected. Peers are also free to decide which other peers to make acquaintances and define acquaintance queries with. This means that communication autonomy of peers is also respected. The query prop-

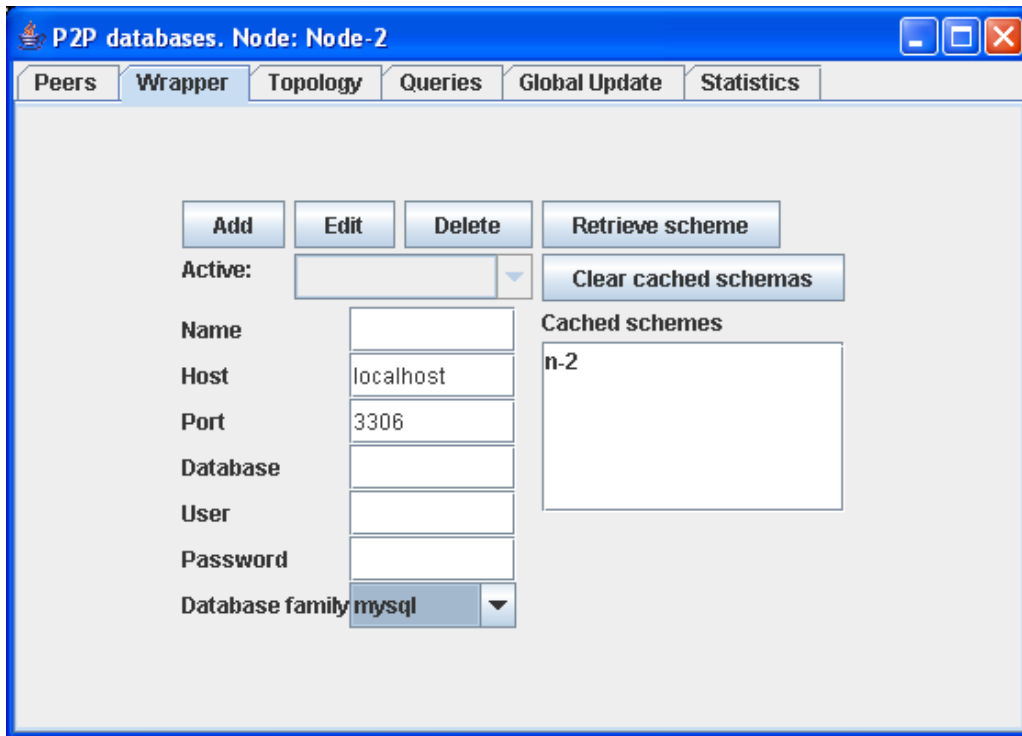


Figure 4.10: Wrapper configuration.

agation and query result propagation algorithms, presented in this thesis, provide for different modalities for peers to answer other peers' requests. In its turn, it gives peers significant execution autonomy. The presented architecture allows for sharing of a part of the peer's LIS, which respects association autonomy of peers. Finally, peers are free to enter and leave the network, and this ensures participation autonomy.

Heterogeneity: the autonomy of peers leads to heterogeneity at all the levels. We solve the data model heterogeneity by exploiting the wrapper-based architecture in order to encapsulate the heterogeneous LISs, and to provide a translation layer from local data model to the data model used within the P2P IS (i.e. to the relational data model). Acquaintance queries solve the schema level heterogeneity problem by specifying two semantically correlated queries, expressed against two heterogeneous schemata of two peers, and a set of mappings, which relate the corresponding elements

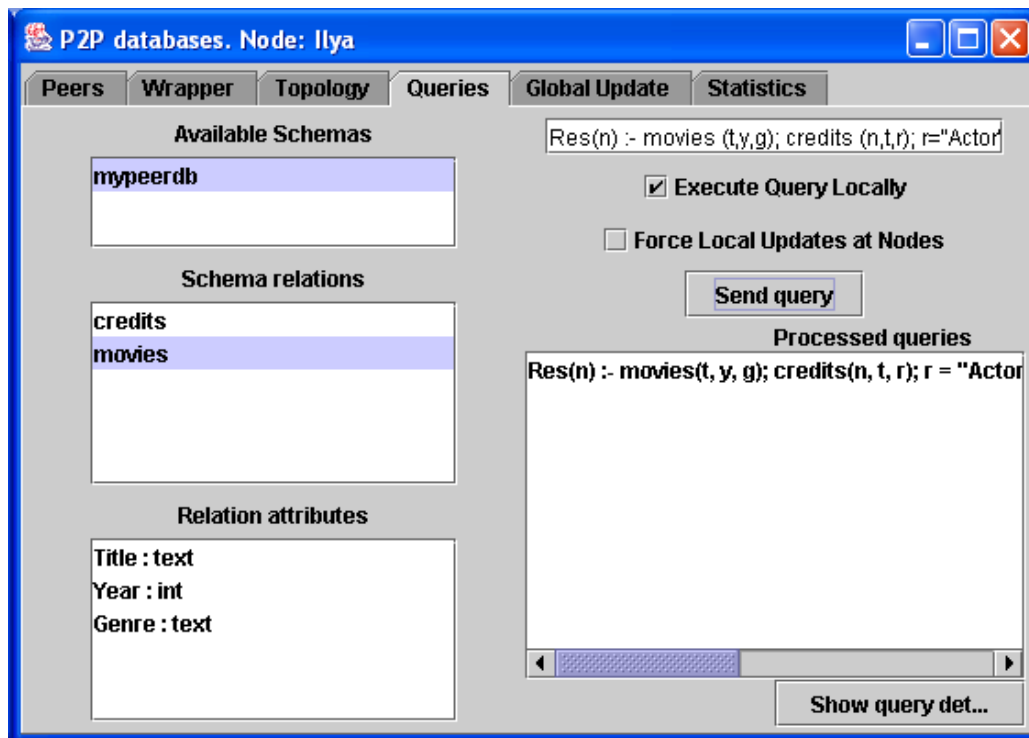


Figure 4.11: User query submission.

of the two queries. As discussed earlier, the data level heterogeneity is largely solved by correspondence rules. Finally, we address the problem of the system-level heterogeneity by building our solution on top of the system-independent platform JXTA.

Distribution: a P2P IS is distributed in all the three dimensions. There is no global schema, which would integrate the resources of the IS, nor is there a global register of resources, available on the network. There is no single party with a global view of the P2P IS. It also means that the principle of locality is respected.

Dynamics: significant dynamics can be supported within a P2P IS. The two main derivative problems, heterogeneity and discovery, are addressed as follows. Due to the decentralized nature and point-to-point mappings definition, a change in a schema of a peer affects only a small part of the P2P IS. Namely, only affected acquaintance queries will need to

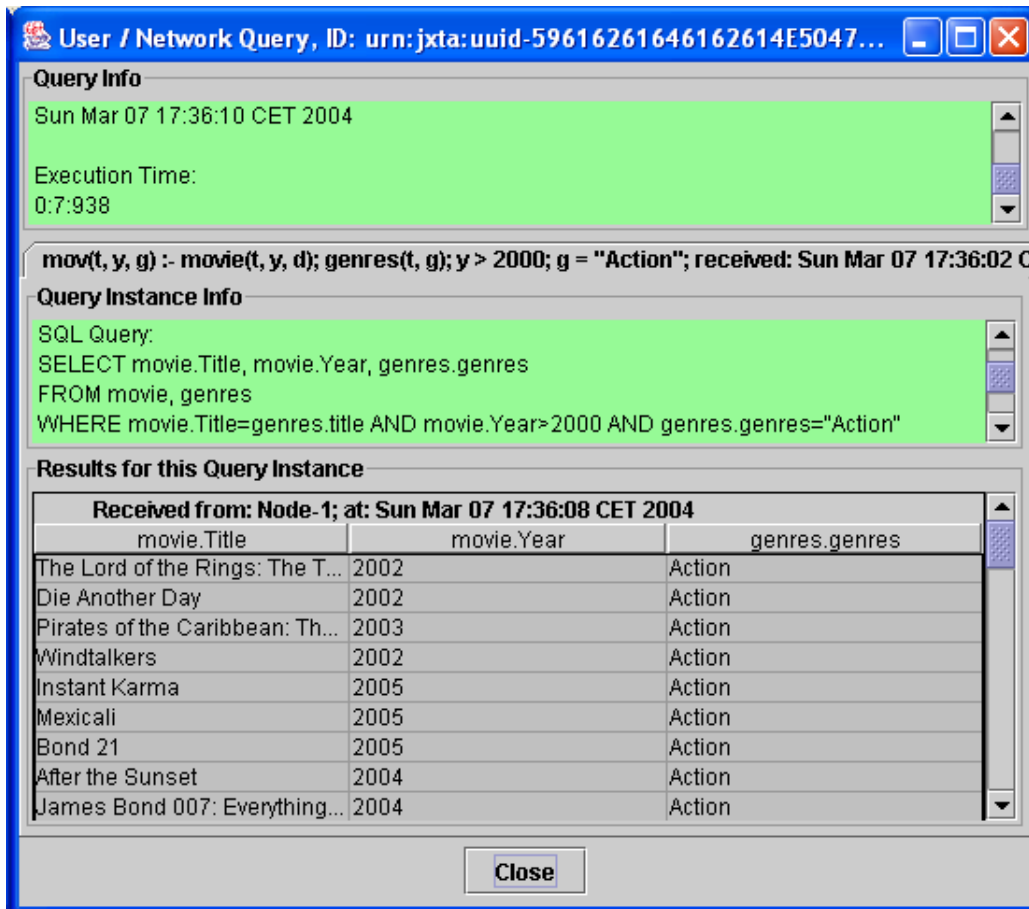


Figure 4.12: Query results (at an intermediate peer).

be reconsidered. This is done locally, with no downtime caused to the whole system. Apart from this, query propagation and query result propagation algorithms are designed to be robust against runtime changes. We address the discovery problem by exploiting the sophisticated resource discovery mechanism of JXTA.

Scalability: the scalability dimension of a P2P IS is supported in several ways. First, interest groups allow it to partition the information storage space according to a topic, and this makes it easier for peers to create acquaintances and make more meaningful connections with them. It limits the propagation of irrelevant queries. Apart from this, interest groups are used to bind the query propagation scope, and this limits it to only

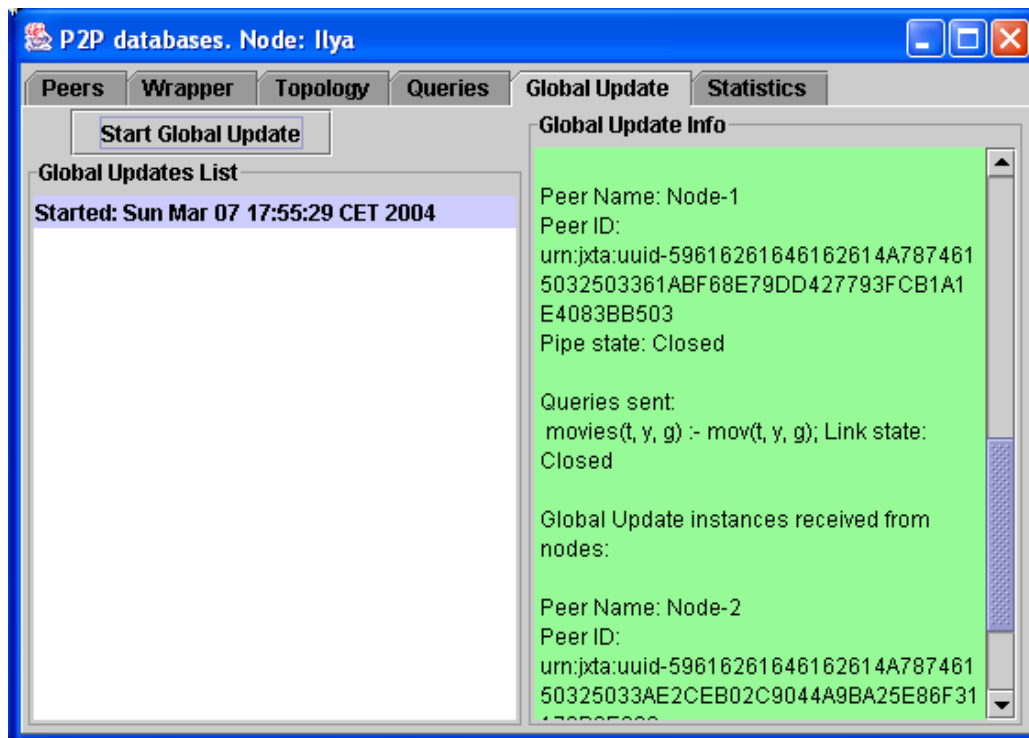


Figure 4.13: Global update processing.

relevant peers. Acquaintance queries define explicit semantics for query propagation, and this avoids the all-to-all propagation scheme leading to the undesirable flooding effect. The query propagation algorithm allows it to select only relevant acquaintance queries to some user query, and this also reduces the number of propagated queries. The implemented transitivity principle allows peers to make only a few acquaintances in order to access potentially all the relevant data sources in the system, and this also favours scalability. Finally, the supported execution autonomy allows peers to balance their load by limiting the processing of net queries and net results. It prevents potential bottlenecks in the system, and also improves scalability.

Apart from the above described properties, a P2P IS is *fault-tolerant*, in the sense that the ability of the system to provide the user with data degrades *gracefully* as less information or fewer nodes are available. This

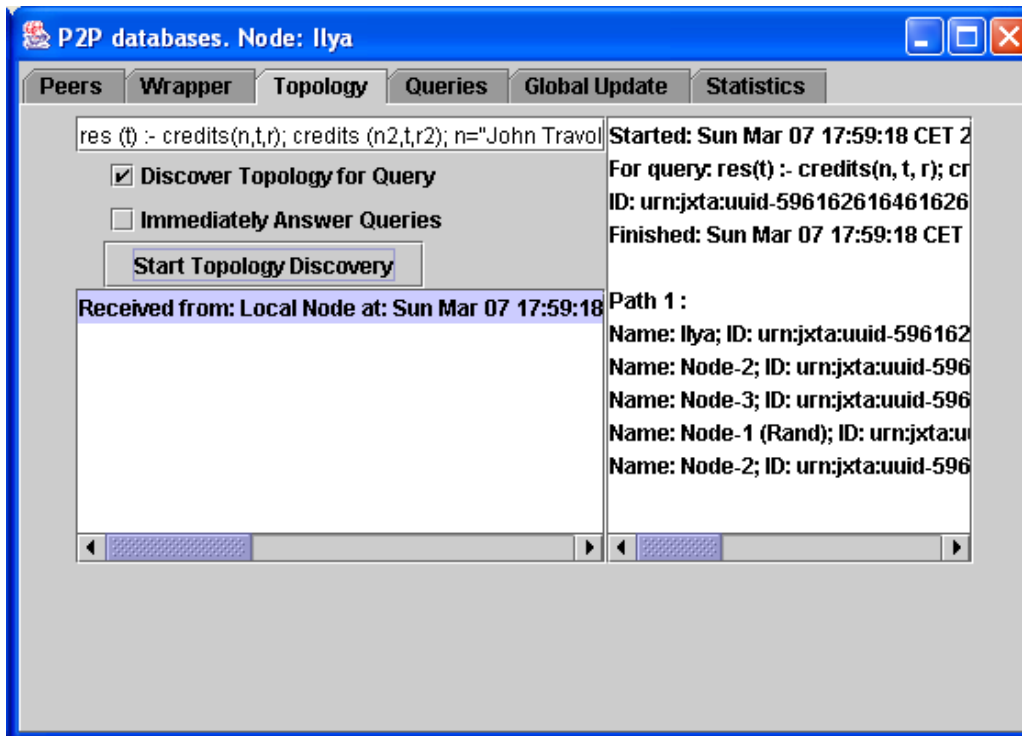


Figure 4.14: Topology discovery for a user query.

is not the case for most data integration solutions, including the federated servers architecture of SQL Server and the shared-nothing architecture of DB2 UDB [37, 22]. In these systems, changing the system configuration (e.g. adding/removing a node) leads to a significant amount of downtime of the whole system.

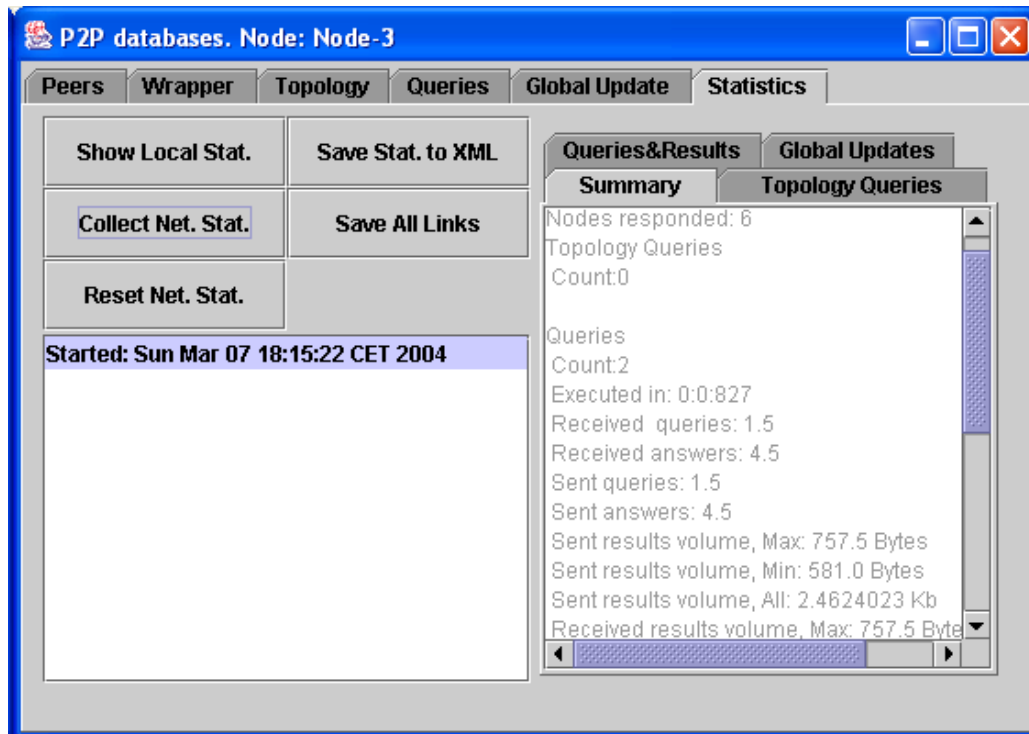


Figure 4.15: Network statistics.

IS	Autonomy					Heterogeneity			Distribution			Dyn.	Sc.
	D.	C.	Ex.	As.	P.	D. M.	S. L.	D. L.	P.	L.	Op.		
C. DBS	Y	NA	Y	NA	NA	NA	NA	NA	C	C	C	Y	NA
DDBS	N	N	N	N	N	N	NA	NA	D	C	C	N	N
T.C. FDBS	Y	N	N	Y	N	Y	Y	Y	D	H	C	N	N
L.C. FDBS	Y	Y	N	Y	N	Y	Y	Y	D	D	C	Y	N
MDBS	Y	Y	N	Y	N	Y	Y	Y	D	D	D	Y	N
Ont.-based	L	Y	Y	Y	N	N	Y	Y	D	C	C	N	N
P2P IS	Y	Y	Y	Y	Y	Y	Y	Y	D	D	D	Y	Y

Table 4.1: A classification of information systems. Legend: Y = dimension present; N = dimension absent; C = centralized; D = distributed; H = hybrid; L=limited; NA = not applicable.

Chapter 5

Evaluation

In this chapter we provide an evaluation of the proposed approach. We generated four P2P ISs of different sizes and conducted a series of experiments measuring various indicators of the performance of the systems. Based on the obtained results, we propose how the query answering algorithm can be further optimized.

The chapter structure is as follows. In Section 5.1 we discuss the settings in which we ran our experiments. In Section 5.2 we report the results of the conducted experiments. In Section 5.3 we summarize the results of the experiments and make our conclusions.

5.1 Experimental Setup

Because of the desire to evaluate the performance of relatively large P2P ISs, and due to lack of available processing resources to support the JXTA-based implementation of such ISs, we implemented a simulator version of the P2P IMS software as a single Java-based application. In the simulator version, all peers are run within the same Java Virtual Machine. Each peer is implemented as a distinct thread. Each peer implements an inbound FIFO queue, which is used by other peers for sending messages to that peer.

Our experimental environment consisted of a single Linux server with 4 Intel Xeon CPUs 3.00GHz each, with 1MB internal cache, and 4Gb RAM. On this server we ran both the simulator application and the peers' databases. The databases were implemented within the PostgreSQL 8.0 database environment. During the experiments, each peer maintained a connection to its database. For efficiency reasons, peers propagated query result messages by sending a Java ResultSet object (which points to the query results in the database) into the inbound queues of other peers.

We experimented with P2P ISs of different sizes. Namely, we considered four systems consisting of 100, 400, 700, and 1000 peers. For each of the systems, we generated schemata and contents of the peers' databases, as well as peers' acquaintances. All the peers had an equal average number of acquaintances, which were selected randomly from the available peers. We used nearly the same configuration parameters of the P2P IS generator tool for the generation of the four systems. The only variant parameter was the average number of acquaintances per peer. We provide a summary of the configuration parameters in Table 5.1.

5.2 Experimental Results

For submitting user queries, we selected peers with the greatest number of acquaintances. The submitted queries referred to all the relations of the selected peers. We allowed the propagation of repeated queries unless their reformulations were contained in previously propagated net queries. In other words, we used option 2d of the query propagation algorithm discussed in Section 3.3.1. Thus, in our experiments, we evaluated the performance of the systems on rather complex queries, and we maximized the propagation of queries and query results.

We submitted two kinds of user queries: without comparison predicates,

Parameter	Value
Number of peers	100, 400, 700, and 1000
Number of relations per peer	Min: 2; max: 4
Number of attributes per relation	Min: 3; max: 5
Number of tuples per relation	Min: 10; max: 50
Domain of the attributes	Integers from 0 to 80
Number of acquaintances per peer	4.91 for 100 peers; 6.01 for 400 peers; 7.00 for 700 peers; and 7.99 for 1000 peers
Number of relation subgoals in acquaintance queries	About 1/2 of the number of the relations of the corresponding acquaintance peers
Number of join attributes between two adjacent relation subgoals in acquaintance queries	Min:1; max:2

Table 5.1: P2P IS generator parameters.

and with a single comparison predicate defined for *each* body variable of the queries. We allowed only the *greater than* ($>$) comparison predicate to be used in user queries. In Figure 5.1 we report query answering times for the four systems and for the two types of user queries.

Note, that the query answering times reported in Figure 5.1 should be understood not as absolute but as relative figures. In fact, since all the peers ran on the same machine, there were not network delays. On the other hand, some delays were introduced because of many concurrent I/O disk operations. In real settings, one factor may compensate the other to a certain extent. The main purpose of Figure 5.1 is to show how query answering time can grow with the growth of the number of peers and with the use of comparison predicates in user queries.

In Figure 5.2 we report the communication complexity of the query answering algorithm. For the four examined systems, we show the number of query and query result messages for queries with and without comparison subgoals. In Figure 5.3 we show the total number of new query result tuples

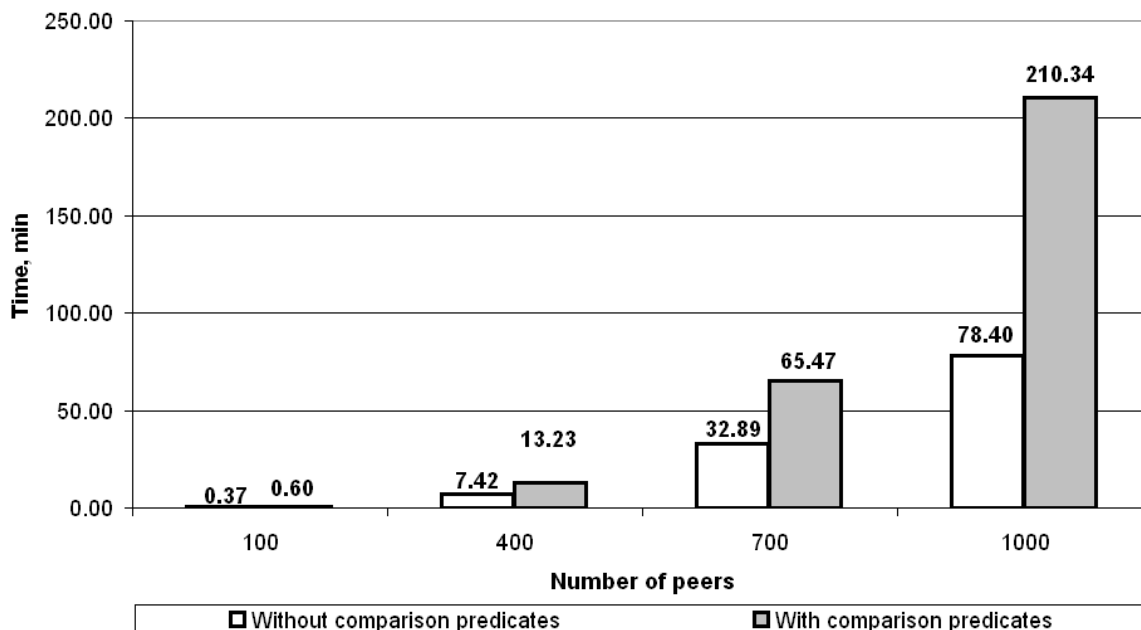


Figure 5.1: Query answering time.

produced by all peers during query answering in the four systems.

The presence of comparison subgoals in user queries stipulates the cyclicity of the query propagation graph. We show this dependency in Figure 5.4, where we report the number of repeated loop queries for the two kinds of user queries. Note that, differently from time measurements, this and the previous two evaluations are independent of whether the P2P IS is implemented in a single machine or is distributed on the network.

On the example of the system consisting of 400 peers, we show the dynamics of the involvement of new peers in the answering of a user query (without comparison predicates). We report our findings in Figure 5.5. As it can be seen, almost all the peers participate in the query answering after about 1/3 of the total query answering time has elapsed.

In Figure 5.6 we show how the average and maximum cache sizes grow during the query answering in the P2P IS consisting of 400 peers. The cache size of a peer is measured as the total number of tuples the peer

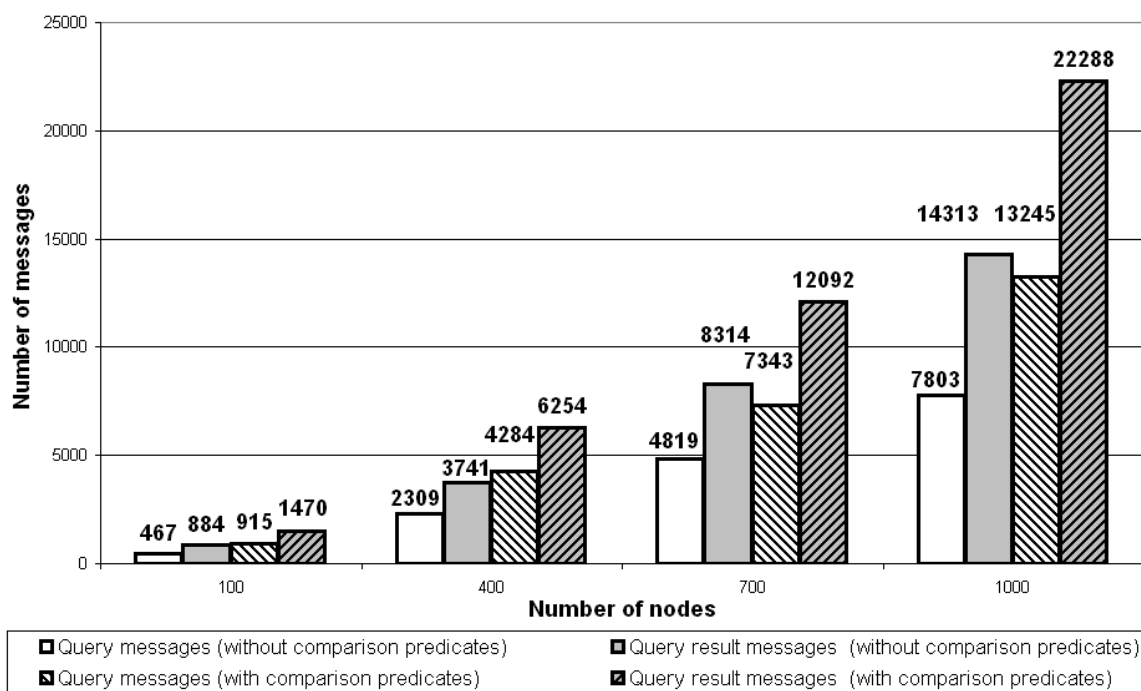


Figure 5.2: Communication complexity of query answering.

temporary stores during the query result propagation (see Sections 3.3.2 and 4.4.2). The maximum value represents the maximum cache size of a peer involved in the query answering. For any given moment of time, the average value is computed as the sum of the cache sizes of all the involved peers divided by the number of these peers.

In all the above reported evaluations, the query result accumulation technique was not used, and correlated queries were not computed (see Section 3.3.2). On the example of the system consisting of 400 peers, we report the effects of using these two techniques in Figure 5.7. As it can be seen, the computation of correlated queries leads to longer processing times and to significantly higher numbers of query result messages which are circulated in the system. Therefore, for this particular system configuration, the use of this technique is not advisable. On the other hand, the result accumulation technique allows it to significantly cut down the num-

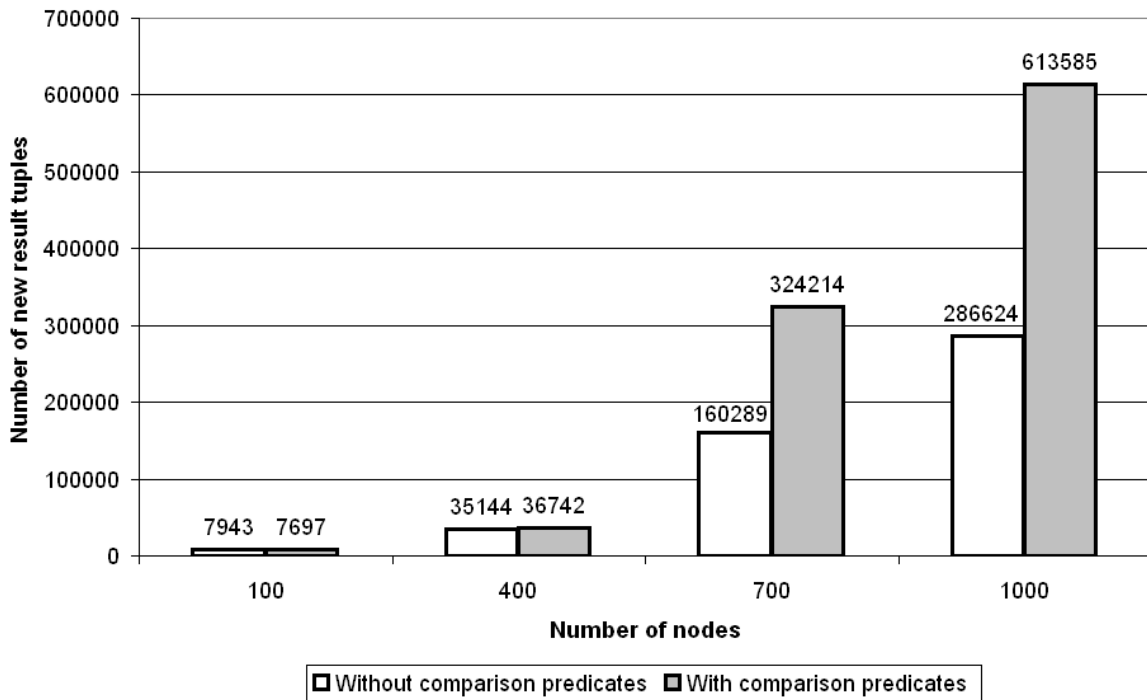


Figure 5.3: Total new query results produced in the P2P IS by all peers.

ber of messages (by about 25%). The best of the four evaluated options, both in terms of the processing time and the number of propagated query result messages, is the one which uses the result accumulation technique and which does not compute correlated queries.

We measured the load of the system as the number of query and query result messages, which are pending in the peers' inbound queues. In Figure 5.8 we report the evaluation results for the system consisting of 400 peers. In this figure we compare the performance of the system with and without result accumulation. As it follows from the figure, result accumulation allows it to reduce substantially the number of pending query result messages (by about 25%) and to complete query answering faster (by about 12%).

In Figure 5.9 we show the number of new user query results (after duplicate elimination) as the function of time. Noteworthy, the result accumu-

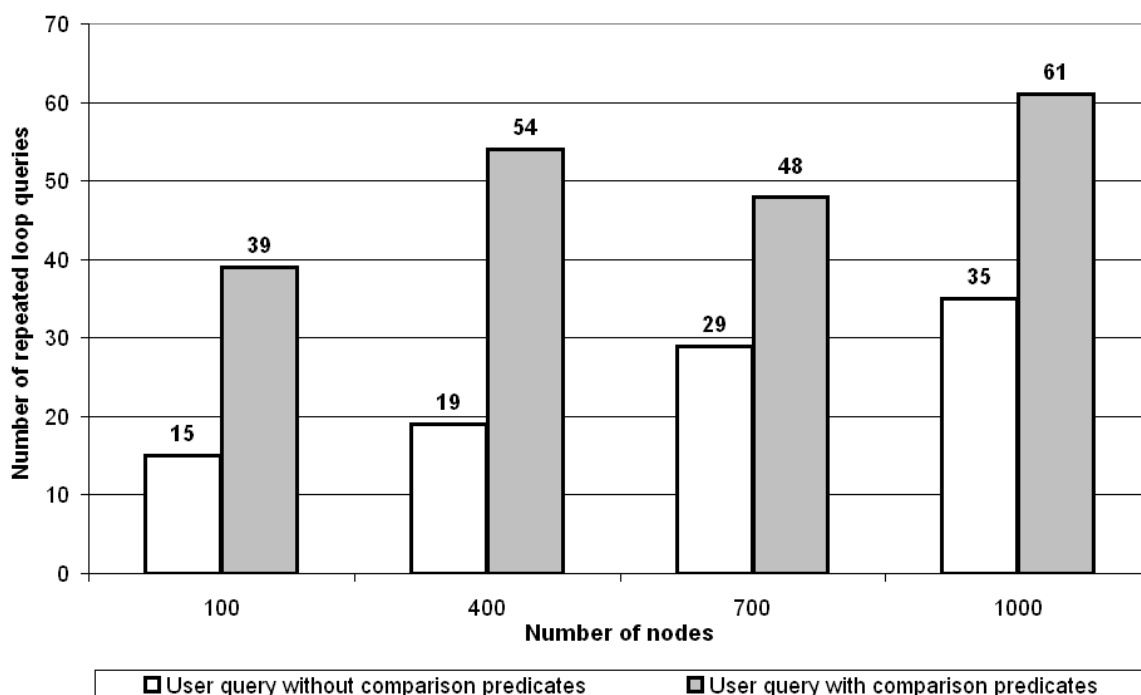


Figure 5.4: Repeated loop queries in the query propagation graphs.

lation strategy without the computation of correlated queries outperforms the other strategies in that it allows to deliver results earlier to the user.

We evaluated relative computational requirements of some of the steps of the query result propagation algorithm. We present its performance evaluation in Figure 5.10. The reported figures are relative time shares of some steps of the algorithm in the processing of net results. The figures are average for all the four systems, which were evaluated with no result accumulation and no computation of correlated queries. Steps 4 and 5 are shown as one category because in the current implementation they are both executed in a single SQL query. Step 2 is missing because no result translation is currently implemented.

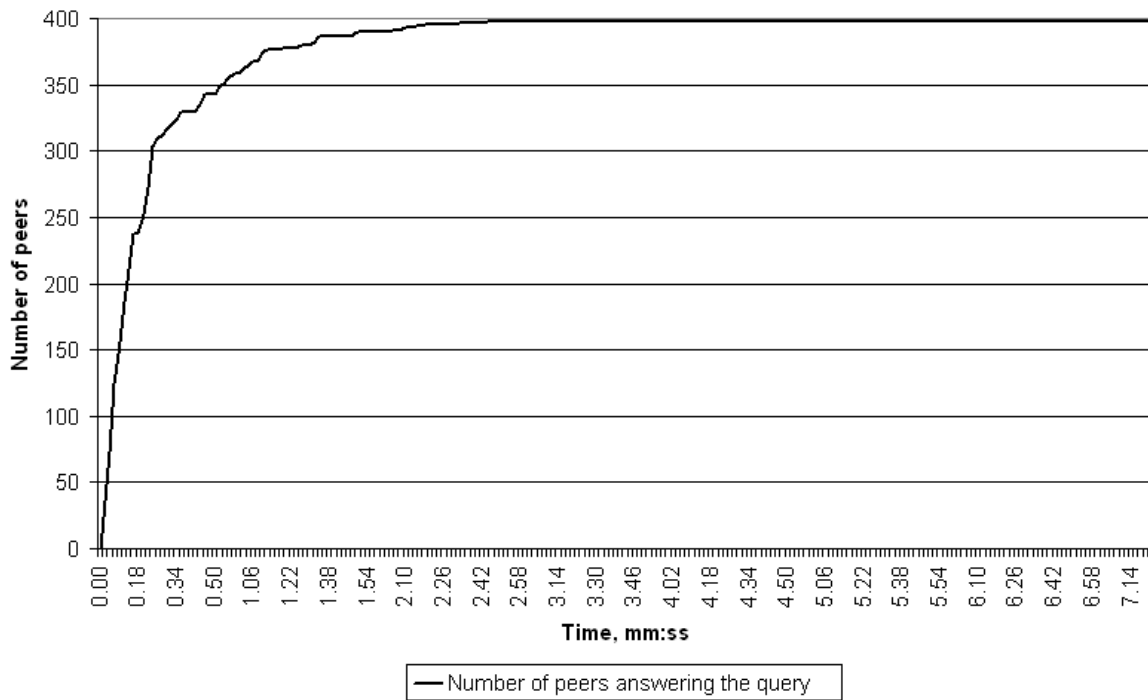


Figure 5.5: Involvement of peers in query answering in the P2P IS consisting of 400 peers.

5.3 Conclusions

The most important conclusion of the experiments is that the proposed solution demonstrates the proof-of-concept of P2P IMSs. However, some considerations need to be made. The reported query answering times are rather high for applications requiring nearly immediate responses from the system. Apart from this, because the user may not be able to immediately see and manipulate query results, the incoming user query results may need to be saved in the local database for future reference. Therefore, the range of applications of the database-based instantiation of the P2P IMS shifts to less time-critical domains.

As we have seen, the result accumulation technique allows it to improve the performance of the P2P IS. The computation of correlated queries is not good for database-based P2P ISs. The reason being is that the

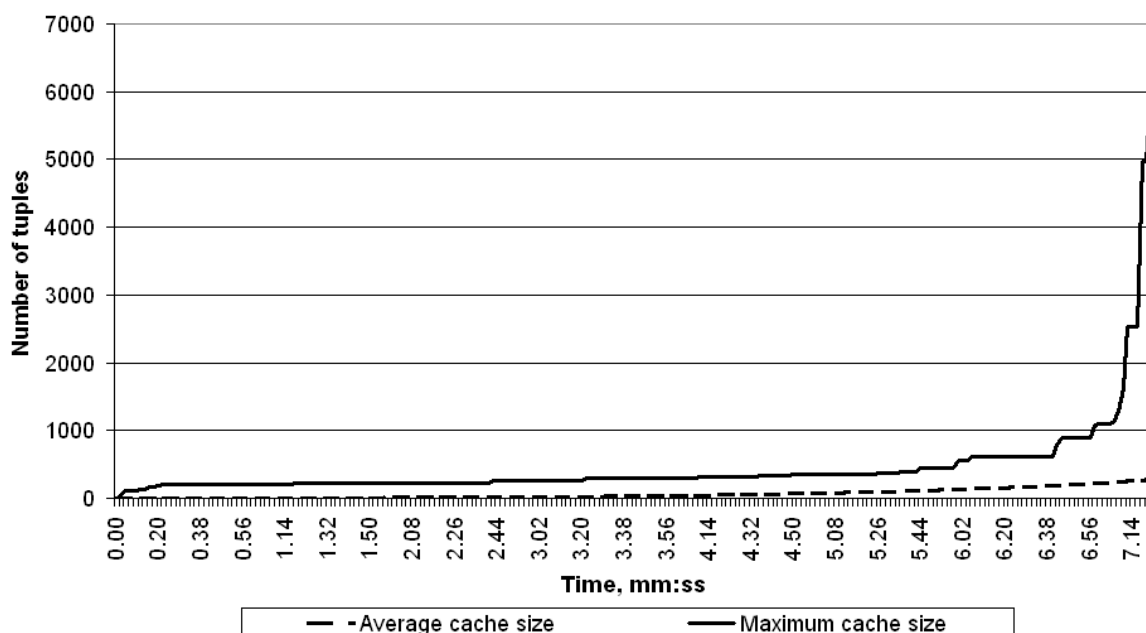


Figure 5.6: Average and maximum cache sizes in the P2P IS consisting of 400 peers.

query computation is a costly operation. Therefore, the benefit of the computation of correlated queries is hindered by high processing overheads.

The presence of comparison subgoals in user queries may increase both time and communication complexity of the query answering. Apart from this, it may also result in a greater cyclicity of the query propagation graph.

The space requirements for the temporary cache are rather moderate. However, in the presence of multiple concurrent user queries, these requirements may increase significantly. To address this problem, the cache can be shared among all the user queries as it is discussed in Section 3.3.2. A negative effect of this, however, is that different user queries may be correlated and, therefore, affect each other's termination.

Based on the analysis of the simulation results, we see several possibilities for the improvement of the proposed query answering algorithm. Given the relatively high cost of the query computation, a certain improvement can be reached by accumulating translated net results after step 3 of the

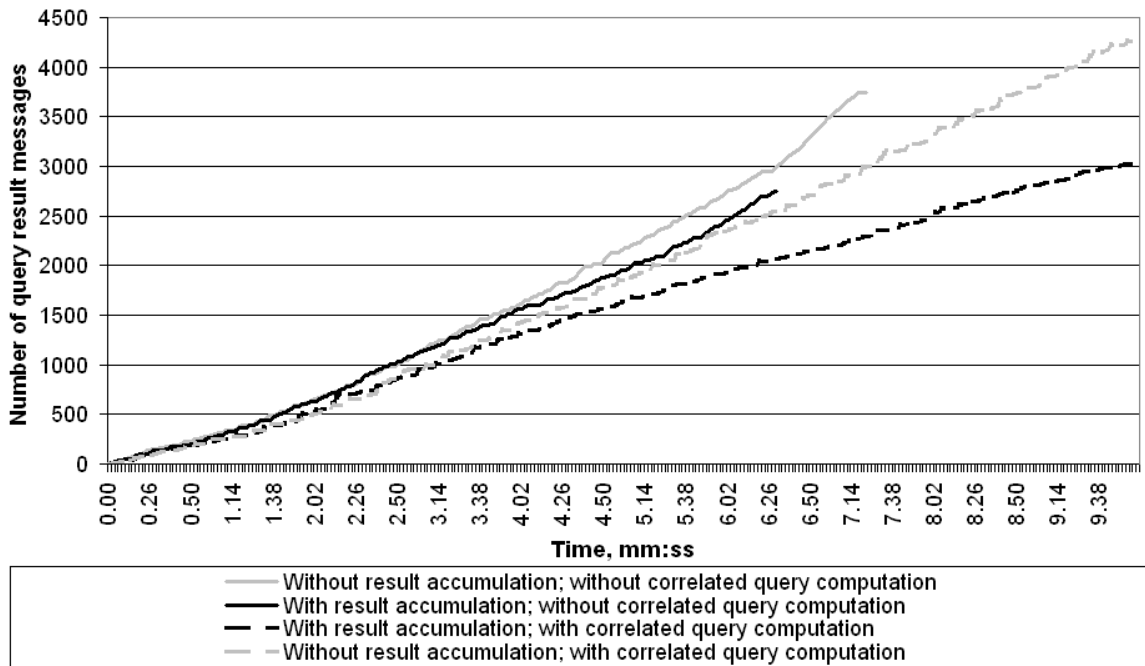


Figure 5.7: The effects of the result accumulation and of the correlated query computation in the P2P IS consisting of 400 peers.

query result propagation algorithm. All the relevant queries are computed at step 4 when there is no net result messages in the peer’s inbound queue.

Another possibility to improve performance is to propagate net queries without including comparison predicates in these queries. The comparison predicates are then used for the computation of the final answer at the peer, which submitted the original user query. However, the obvious benefit of using this optimization for the four modelled systems may become less evident or, even, it may become a disadvantage if different system settings and/or different comparison predicates are used. A further study is required to address this problem.

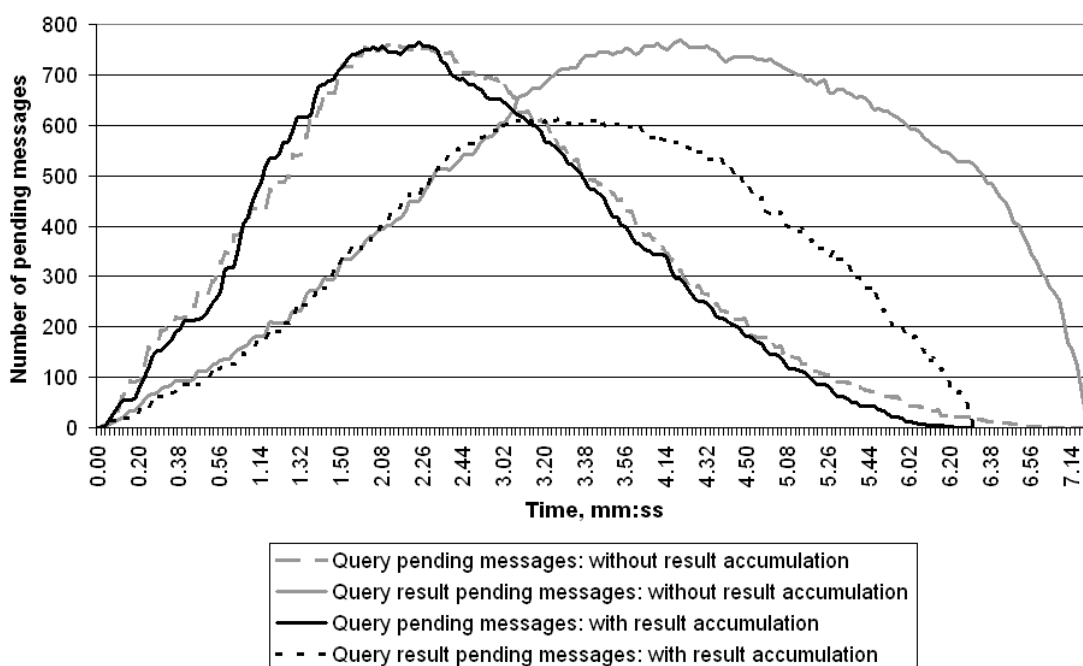


Figure 5.8: Pending messages with and without result accumulation in the P2P IS consisting of 400 peers.

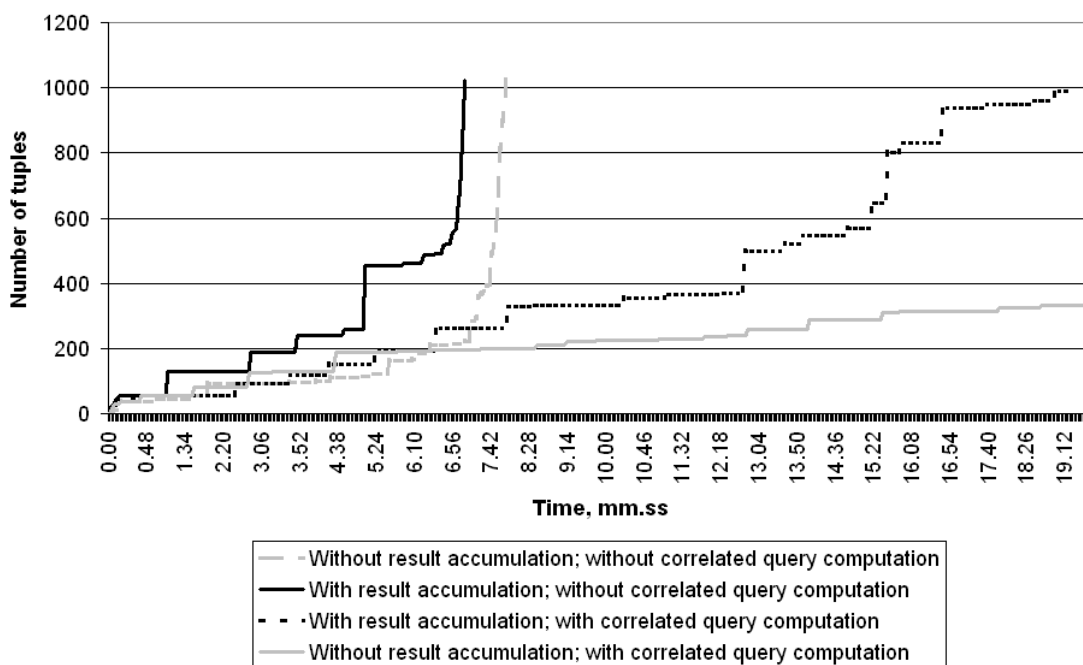


Figure 5.9: New user query results received from the P2P IS consisting of 400 peers.

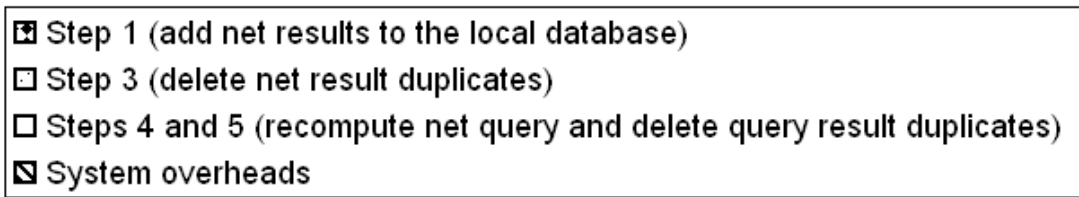
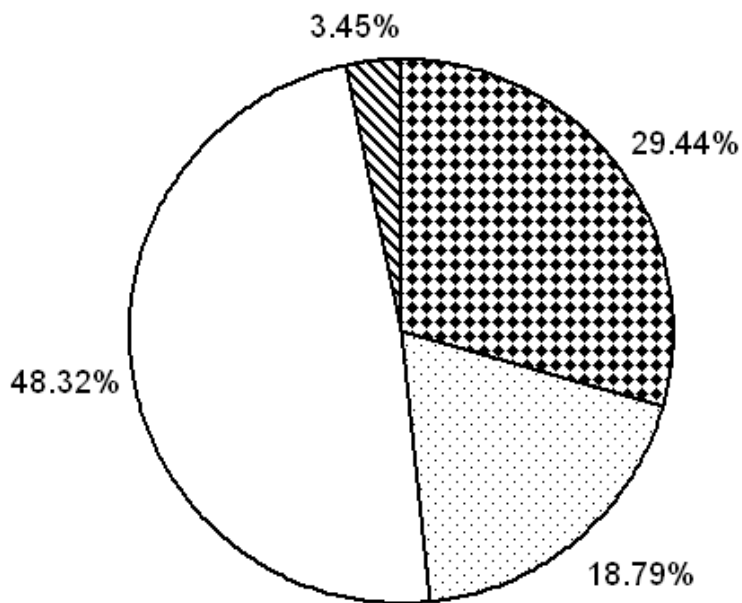


Figure 5.10: Performance evaluation of the query result propagation algorithm.

Chapter 6

Related work

In this chapter we provide a detailed analysis of the related work. Particularly, we make it clear in what respect the proposed solution is new, and how it compares to other existing approaches. We provide a fine-grained self-assessment along several dimensions where we report innovative results: the foundations of P2P IMSs, the P2P information management model, query answering in heterogeneous P2P data management systems, and good-enough answers.

The chapter structure is as follows. In Section 6.1 we discuss how the proposed definition of a P2P IMS is related to the ones proposed in the literature. In Section 6.2 we show in what respect P2P IMSs are different from their closest ancestors from the family of database systems. In Section 6.3 we discuss the related work for each of the core notions of the proposed P2P information management model. In Section 6.4 we relate the proposed query answering algorithm to other similar algorithms proposed in the literature. Finally, in Section 6.5 we discuss what has been done in the field of data quality for P2P systems and we relate the notion of good-enough answers to the state-of-the-art in the field.

6.1 Foundations of P2P Data Management Systems

Peer-to-Peer has been the focus of rapt attention in various areas of research. For instance, at the time of this writing, a query at CiteSeer for “P2P” or “Peer-to-Peer” returned more than 12000 documents and more than 11500 citations, most of them dating from the three years period from 2001 to 2003. The spectrum that addresses the problem of *data management* in P2P networks includes, but is by no means limited to the following fundamental works: [74, 31, 82, 120, 56, 117, 132, 40, 18]. Notably, that one of the pioneer works in the field was our work, where the *Local Relational Model* was introduced as a data model specifically designed for P2P data management applications [31].

Despite the comprehensive work on P2P data management, there is no well established consensus of what a P2P data management system is, and, to the best of our knowledge, there is no report which would provide a detailed account of how P2P data management systems are related to the conventional ones.

Let us consider some examples. Yang and Garcia-Molina define P2P systems as “*distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other*” [164]. Gribble et al. define it as a distributed system “*in which participants rely on one another for service, rather than solely relying on dedicated and often centralized infrastructure*” [74]. The IT consulting group OVUM defines a P2P application as “*any application or process that uses a distributed architecture and direct bidirectional communication between autonomous resources without central co-ordination and management*” [26]. In a recent survey Androutsellis-Theotokis and Spinellis accentuate the problem of multiple definitions of P2P systems, and define them as “*distributed systems con-*

sisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth . . . without requiring the intermediation or support of a global centralized server or authority' [23].

While they capture some of the essential characteristics of P2P, the above definitions still allow an ambiguous interpretation, in the sense that they do not fully discriminate P2P from the conventional distributed data management systems. For instance, the notions of (partial) distribution, decentralization, equal capabilities, (partial) autonomy, and point-to-point communication can be also attributed to loosely coupled FDBSs and/or multidatabase language systems (see Section 2.3). The commonly missing discriminating characteristics of these and many other definitions are the principles of locality and transitivity (see Section 3.1).

The absolute majority of reports about P2P (information) systems do not provide a definition at all. The notion is either assumed to be well understood in the community, or it is described by its properties and benefits, such as significant autonomy, distribution, and scalability. Some of the properties are considered to be commonly agreed on, and some are either assumed implicitly or are missing. For instance, in their tutorial on P2P information systems, Aberer and Hauswirth explicitly identify such properties as full distribution, autonomy, scalability, but miss transitivity [19]. The concept of P2P in the Hyperion project [132] includes such attributes as full decentralization, autonomy, and equal peer capabilities, but it also misses transitivity.

The crucial role of the transitivity principle as a discriminating factor of P2P database systems is highlighted in the PeerDB project [120]. This paper, as well as another independent work [19], provides a comparison of P2P data management systems with distributed databases. However, in both cases only tightly integrated architectures of database systems

are considered, and no reference is given to the more P2P-like ones (see Section 2.3).

To our best knowledge, this thesis is the first attempt to provide a definition of a P2P information management system, which unambiguously discriminates it from the previously developed data management paradigms. Apart from this, it is the first report known to us, which provides a detailed comparison of P2P information management systems with the different topologies of relational multi-database systems and ontology-based systems in the light of crisply defined analysis dimensions.

6.2 P2P vs. Conventional Database Systems

The closest ancestor of the P2P IS in the family of relational database systems is the loosely coupled FDBS. In fact, in both approaches, there is no global schema; mappings are defined locally among component databases; and they are used for local query rewriting and propagation in the point-to-point fashion. Apart from some differences in the autonomy, distribution, and scalability dimensions (see Table 4.1), the crucial difference between the two approaches is in the *transitivity* and *locality principles*. Recall the schema architecture of a loosely coupled FDBS shown in Figure 2.5, and the two principles graphically represented in Figure 3.1. In Figure 6.1 we compare the schema architectures of the two approaches. For the sake of presentation we use the schema notation of a FDBS to represent the schema of a P2P DBS.

In a loosely coupled FDBS, when federated functionality is desired, a query is posed against a federated schema of a component database (see Figure 6.1a). The query is then rewritten against the export schemata of the corresponding component databases in the system. No further query

propagation takes place. In a P2P DBS, a query is formulated against the shared export schema of a component database. The query is then rewritten into queries against export schemata of other component databases according to the mappings defined from the original export schema to the export schemata of acquaintances. It results in a query propagation from one component database to another. When a component database receives such a query, it processes it in a *similar manner*, and this results in its further transitive propagation (see Figure 6.1b).

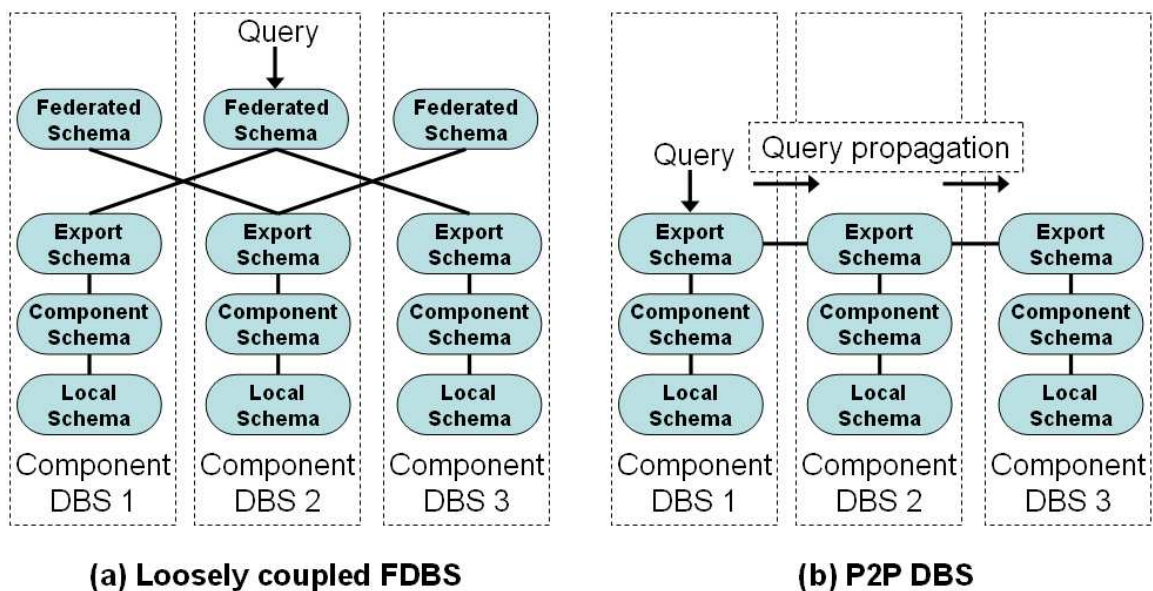


Figure 6.1: Schema architectures of a loosely coupled FDBS and a P2P DBS, compared.

The two architectures are different because they are based on two contrary assumptions. In the loosely coupled FDBS, it is assumed that each component database can potentially connect with all the relevant component databases in the system, i.e. that there exists a global view of the whole system. Therefore, only one query rewriting is sufficient to retrieve a (likely correct and complete) answer from the system. In the P2P DBS the assumption is that each component database has only a partial view of the data in the system, i.e. the locality principle holds. In such settings, each

component database participating in query answering may know about other new component databases, that can be used to answer the query. Therefore, a query may need to be rewritten and propagated in a chain of component databases, and this is reflected in the architecture of P2P DBSs.

Taking into account the above considerations we make the following conclusion: even if some of the conventional multi-database systems are largely distributed, support point-to-point inter-operation of the component DBSs without relying on a global schema, and possess other characteristics peculiar to P2P, these systems are still built based on the data integration principles, which are encoded into the architecture of these systems. Controversially, P2P ISs are built on the principle of *data coordination* [68], which rejects the assumption of the global view and/or the global schema, which are fundamental to the conventional systems. Instead, it favours only local partial views of the system, and a point-to-point local transitive interaction.

6.3 Information Management Model

6.3.1 Interest groups

The idea that data can be physically placed in nodes close to where relevant queries originate, was used in early distributed database systems [100], and it is fundamental to the architecture of the current state-of-the-art multi-database systems (see Section 2.3.6). However, in the pure P2P settings, where data placement cannot be known a priori or enforced, this idea evolved into the concept of logical proximity of peers carrying similar content. To the best of our knowledge, in the context of P2P systems, this concept was introduced for the first time as interest groups in our early work in September 2002 [68]. Shortly later it was independently introduced

in [97], where it was called a P2P community; and in [46], where it was called a Semantic Overlay Network. In [105], one year later the authors define it as a Semantic Overlay Cluster.

In [97] the authors define a P2P community as a set of peers sharing some common interests, whereas common interests are defined as the intersection of keywords-based attributes claimed by each peer. Similarly to interest groups, P2P communities are used for structuring the information storage space, discovering resources, and pruning the search space. The difference from our approach is in that P2P communities are created implicitly by the fact that several peers share some attributes. It makes community identification and discovery a challenging task. There is no crisp boundary of a P2P community, and, at any given time, peers cannot know exactly what communities they belong to.

Crespo and Garcia-Molina [46] propose the idea that connections between peers should be influenced by similar content of the peers, and that these semantically related peers can be grouped together into Semantic Overlay Networks (SONs). They formalize the notion of SON by labelling links between nodes with a string, and by defining a SON as a set of links with the same label. Similarly to our approach, SONs are organized into a hierarchy, and query propagation is limited to the nodes of relevant SONs. The difference from our approach is in that we distinguish between two notions of an interest group and a link between peers (i.e. acquaintances), and in the proposed approach the two notions are collected into the one of SON.

Reference [46] is also relevant to our work in another respect. Namely, it suggests that nodes can be *classified* into the SON hierarchy, which can be represented as a tree of concepts. In its essence it is similar to our proposal to classify nodes and user queries into the interest group hierarchy. However, while in [46] the authors provide a basic idea of how the

classification can be performed, basing it on machine learning approaches, in [61, 62], we provide a detailed account of how the classification can be performed, using a logics-based approach.

An approach similar to [46] is proposed in [105], where peers are grouped into Semantic Overlay Clusters (SOCs) by means of clustering of their schemata. Various clustering techniques are discussed in [117]. The difference between [105] and [46] is in that peers are grouped in SOC based on their schema similarities, and not based on link properties. Another difference is that SOC are build on top of a super-peer architecture [165], such that each super-peer forms a SOC. Similarly to our approach, a certain threshold needs to be reached in the matchmaking of the clustering policies of a SOC and the schema of a peer, in order to allow that peer to join the SOC. SOC are different from our approach because query propagation is not limited to particular SOC; and the interest groups are built on top of a pure P2P architecture, whereas SOC are exploiting a hybrid P2P architecture.

Some other approaches are worth mentioning. The relatively newly distributed approach to organizational knowledge management [32, 33] introduces the concept of (peer) federation as a group of autonomous and locally managed knowledge sources, which exhibit a common interest in a topic. The goal of the P2PEOPLE IST FP5 project [15] is to develop a P2P platform which would facilitate the formation of “common interest” user groups, and which would provide the members of these groups with new ways of communicating in a collaborative (business) environment.

Note that there is no such notion as an interest group in traditional data management systems. The need of introducing this notion in P2P data management systems comes from the fact that P2P systems are open-ended, and, therefore, they can potentially include a very large number of peers. And, since peers have only some partial local knowledge about other

peers, they need to have a mechanism which would provide them with an aggregated (and probably still partial) view of the system, in order to make resource discovery and query answering more efficient and effective (see Section 3.2.1 for the details).

6.3.2 Acquaintances

The notion of acquaintance represents a logical (directed) link from one peer to another, which is used to propagate queries and data. The links play different roles in different types of P2P data management systems; and, the cost of their establishment and maintenance is also different. In Table 6.1 we provide a classification of (acquaintance) links, where we classify them on the base of the heterogeneity property of the schemata, which are related by the links¹.

File sharing systems such as Gnutella [21] and Kazaa [4] use a *homogeneous schema* at all peers to describe shared files, and they use keywords-based queries against this schema to submit queries to the system. In these systems the role of links between peers is minimal, and their essence is in *query propagation*, which essentially means that a peer knows the address of another peer and knows how to pass a query message to it. Once a desired resource is found, it is downloaded directly to the requesting peer without any link consideration. Because they use a homogeneous schema, peers can easily create links with any other peers in the system; and, therefore, the cost of link maintenance is low. Queries can be propagated to all the peers, a peer has links with. It has a negative effect on scalability due to the possible flooding effect, which is normally dealt with by development of effective query routing algorithms [45]. These systems have an *unstructured* network architecture, as the placement of content is independent of

¹A similar classification is reported in [117], where it is built on the base of the criteria of schema capabilities and distribution.

	Homogeneous schemata / keywords	DHT-based	Heterogeneous schemata
Cost of link establishment and maintenance	Low	Medium	High
Role of links	Query propagation	Identification of resource locations, query propagation	Identification of relevant peers to a query, query rewriting, propagation of queries, query results, and updates
Network architecture	Unstructured	Structured	Semi-structured
Drawbacks	Poor scalability due to the flooding effect	Exact-match queries	High cost of maintenance
Examples of systems	Gnutella [21], Kazaa [4]	Chord [147], CAN [130]	The current proposal, coDB [57], Edutella [118], Piazza [82], Hyperion [132]

Table 6.1: A classification of acquaintance links.

the topology of links [23].

As a means to address the problem of unstructured P2P systems, a new class of *structured* systems was developed, whose underlying idea is that content placement can be controlled through a globally agreed-upon scheme, and that the location of any content item (e.g. a file) can be identified in a limited number of messages by using distributed hash tables (DHTs). Commonly, each content item is assigned a key, and the set of keys forms the keyspace, which is then partitioned among the peers, such that each peer stores contents identified by some portion of the keyspace. Given a key, any peer in the system either stores the content item for

this key, or has a link to the peer, which is closer to the item in terms of the keyspace distance. Typical systems of this category are Chord [147] and CAN [130]. Links in this approach are given a more important role as, apart from query propagation, they identify the location of a resource. The cost of links maintenance is higher as DHTs have to be updated when peers enter or leave the system. The main drawback of structured P2P systems is that they support only exact-match queries, i.e. in order to find a resource, its key has to be known [23].

In the present proposal, links are used for the query rewriting. Thus, they identify those acquaintance peers, which are relevant to a given query. They are also used for the propagation of queries, and, differently from the other two approaches, of query results and updates (for an example of update propagation, see [56]). Since they are used for query rewriting, links (partly) encode the location of particular contents at other peers. Therefore, we say that this kind of systems has a *semi-structured* network architecture. The cost of link creation and maintenance is considered to be higher than in the previous two approaches, because the user may need to be involved in the process. The use of links described above is (to a large extent) supported by all heterogeneous schema-based data management systems (e.g. coDB [57], Edutella [118], Piazza [82], Hyperion [132]).

We have provided such a comprehensive introduction to the issue of links, to underline the importance of the notion of link in P2P systems, especially in those supporting heterogeneous schemata. In fact, in a recent SIGMOD issue on P2P data management, it was pointed out that “*introducing (heterogeneous) data schemas into P2P systems is a problem at the heart of (P2P) data management*” [17]. It makes links a first level architectural notion, as suggested in this thesis. Note that there is no such notion in conventional data management systems (see Section 2.3), which is mainly conditioned by the global view assumption, and by no transitive

propagation among the system components (see Section 6.2).

To the best of our knowledge, the important role of links as a distinct architectural notion in P2P systems was realized for the first time in [68], where we introduced the notion of acquaintance. Khambatti et al. [97] decouple the two notions of P2P community and link between peers, and motivate the need for links. In this respect, it is the closest reference (which we know about) to our work reported in [68]. The difference is that in our work we parameterize an acquaintance link by a query (see Section 3.2.2), and in [97] the authors consider simple connections between peers, used mainly for P2P community discovery and query propagation. According to the classification shown in Table 6.1, the work reported in [97] falls into the category of homogeneous / keywords-based systems, where the role of links is less important than in heterogeneous schema-based systems.

There are some other relevant approaches presented in the literature. The idea of representing a link as a query over the relations of some peer(s), whose answer conforms to the schema of a given peer, is common to many heterogeneous schema-based P2P data management systems (e.g. coDB [57], Piazza [82], Hyperion [132]). The most generic one is Piazza [82], which allows it to define link queries involving *multiple* peers. In our terminology, this means that it allows both local and remote queries to be queries over relations of multiple peers. However, this approach becomes especially vulnerable to dynamics as the stability of a mapping depends on the stability of multiple peers, whereas in our approach, mappings are defined only between pairs of peers.

6.3.3 Coordination rules

The need for a *coordination* mechanism suitable to support component interoperation in a P2P system has been realized by the research community [31, 68, 24]. The technology used to implement conventional data

management systems is largely driven by the concept of (global) *integration*, and cannot be directly applied in the P2P settings. The main factors that condition its inapplicability are the principles of locality and transitivity, as well as significant scalability and dynamics of P2P data management systems (see Sections 3.1 and 6.2).

There are several proposals for the implementation of the P2P coordination mechanism. The Hyperion project uses an ECA (*Event-Condition-Action*) mechanism for peer coordination [93], whose origin mainly comes from the research done in the area of active database systems [47]. While we introduce the idea of using ECA rules for peer coordination in [68], and give examples of ECA rules for basic database events (e.g. DELETE, UPDATE) in [69], in [93] the authors discuss details of an implementation, which is based on the use of database triggers, extended to the possibility to refer to multiple peers in their Event, Condition, and Action parts. However, it requires programming of the local database at each peer, whereas different databases may provide a different trigger implementation, or not provide it at all. Another drawback is that triggers normally do not provide any support for query events processing, and, therefore, they are mostly suitable for the propagation of updates among peers.

A similar idea was introduced in [105], where an ECA-like mechanism is used for the encoding of clustering policies in a semantic overlay cluster network. There, an event can be a new peer joining a cluster, the condition can refer to some properties of a peer (e.g. it can check if the peer schema is related to a certain topic), and the action can be the acceptance or rejection of a peer membership application.

6.3.4 Correspondence rules

The autonomy of peers stipulates that different peers can use different values to refer to the same objects (i.e. they can be heterogeneous at

the data level as discussed in Section 2.2.2). This problem was realized in [31], and the solutions addressing it include the specification of *domain relations* [31], *correspondence rules* [68], and *mapping tables* [95]. While all the approaches are rather similar in their essence, reference [95] provides a detailed account of semantic and algorithmic issues related to the use of mapping tables.

6.4 Query Answering Algorithm

In spite of the many research groups working on various problems related to query answering in the context of heterogeneous schema-based (relational) P2P data management systems, there are very few comprehensive proposals of a query answering algorithm². In the following we briefly discuss about query answering issues, which are addressed in various approaches. We then relate the query answering algorithm (and its implementation), proposed in this thesis, to the most comprehensive algorithm proposed so far in [59].

Edutella forwards queries among super-peers, which are arranged in a HyperCup topology [119]. The topology is designed to support efficient query routing, and it guarantees that any super-peer is reached by a query at most once. The last one makes the query propagation graph a (rooted) tree. Because it is based on RDF, Edutella does not support complex query rewriting (as it is the case for select-project-join queries over relational schemata), and, therefore, it does not require cyclic query processing.

Calvanese et al. [40] propose a logical characterization of a relational P2P data management system, and provide a sketch of a query answering

²In our analysis we considered only those systems, which specify mappings among peer schemata and which use these mappings for the (rewriting and) propagation of queries, query results, and, possibly, updates. Thus, for instance, we do not consider the relational P2P PeerDB system [120], in which query rewriting is based on Information Retrieval methods, rather than on mapping consideration.

algorithm. The proposed algorithm addresses the issue of cyclic topologies by requiring that any mapping be used for the rewriting of a query only once within the same transaction. However, as we showed in Section 4.4.2, the same mapping may need to be activated more than once in order to provide a more complete answer to a query. While showing that a query propagation eventually terminates due to the fact that no mapping is used twice, the authors do not provide an account of when query results propagation terminates. As shown in Section 3.3.2, it plays an important role for the identification of when query answering is complete.

In their recent work, Halevy et al. [83] discuss in detail the issue of query rewriting, and, more generally, query answering in the Piazza relational P2P data management systems. A major contribution of this work is that it characterizes the complexity of query answering for the language used in Piazza for both cyclic and acyclic topologies. Similarly to reference [40], the authors avoid cyclic query propagation by restricting a duplicate use of peer mappings within the same transaction; also they do not provide details of how peers can decide *locally* when query answering is complete.

The Hyperion project [24] addresses the problem of incorporating of mapping tables (which are similar to correspondence rules) into query rewritings. It does not address the problem of cyclic topologies, nor the problem of query answering termination in general. Aberer et al. [18] support the definition of mappings between peer schemata, each consisting of a single relation. The main focus of their work is on analysis of cyclic query (and data) propagation paths with the purpose of identification of potential inconsistencies. Though it is not directly related to query answering, this work is complementary to all relational P2P data management systems which use mappings for query propagation.

The query answering algorithm proposed in this thesis has emerged as a result of continuous elaboration of the algorithm originally sketched out

in [68]. There, we introduced the idea that query propagation in a P2P database network can be limited to the scope of interest groups. Apart from this, we introduced the idea that queries and query results can be propagated *transitively* through chains of peers, and that this propagation can be driven by ECA-like coordination rules. In the same work, we suggested some basic techniques for dealing with cyclic topologies, which are mainly based on the use of (pseudo-)unique query identifiers. Later, (elaborated) versions of the algorithm appeared in [71, 69], and, partly, in [58]. In [69] we propose to assign *states* to acquaintance queries, which are used during query answering, and which signal if (more) data are expected to be imported using these acquaintance queries. We use these states to identify *locally* at each peer when query answering is complete. We used a similar idea in [58] but in the context of update propagation.

To the best of our knowledge, the most comprehensive query answering algorithm for heterogeneous relational P2P data management systems was reported in [59]. This work is a logical consequence of [55, 58, 56], to which the author of the present thesis provided his contribution. The authors present the idea of exploring the topology structure at the runtime of query answering, and using this information locally at peers for the identification of when query answering is complete. This idea has been largely reused in this thesis.

The present proposal differs from [59] in several respects. First, it allows queries to be propagated through the same mapping more than once within the same transaction, given that consequently propagated queries are *not contained* in the previously propagated ones (see Section 3.3.1 for details). Second, in our approach query results are propagated with timestamp information which allows it to avoid declaring user queries “open” after they have been “closed” as it is possible in [59]³ (see Section 3.3.3). Third,

³As from a private communication with the authors of [59], in the actual implementation of the

our algorithm allows it to accumulate query results before sending them, and this allows it to avoid a potential flooding effect (see Section 3.3.2). Fourth, the caching technique that we exploit, allows it to decouple peers' data from temporary query results and to address the issue of duplicates, which optimizes the network traffic. Finally, our approach foresees more execution autonomy for peers and load self-balancing by allowing peers to process queries and query results differently (see Sections 3.3.1 and 3.3.2).

6.5 Good Enough Answers

The problem that standard metrics for measuring quality of answers, such as correctness and completeness, cannot be directly applied in the P2P settings has been recognized by the research community [68, 115, 86, 124, 105, 164]. However, very little has been done so far for the development of methodologies to assess the quality of query answers in heterogeneous schema-based P2P systems.

As a step from centralized to distributed but still integrated systems, [116, 138, 115] discuss data quality issues related to query answering against a global schema. Particularly, [116] discusses how to answer a user query using only a subset of relevant relational sources based on predefined data quality criteria. The work reported in [138] discusses data quality issues in the context of cooperative information systems, and it improves on the work in [116] by allowing the association of meta-data to the quality values, which allows it to improve the quality of query answers. Finally, [115] discusses how the data quality criteria shift, when going from database integration solutions to the integration of autonomous information sources. Apart from this, the work presented in [115] is very close in its spirit to the notion of good-enough answers, in that it shows that in a large scale

algorithm this problem is solved using a similar technique.

and autonomous environment such as the web, users cannot expect correct and complete query answers, but they accept incomplete and partially incorrect answers.

According to [116, 138], they are the first of a few works which shift the focus of the data quality problem from single centralized systems to distributed integrated systems⁴. To the best of our knowledge, none of the existing works addresses the problem of the quality of query answers in P2P ISs in enough detail. However, some first preliminary works have been reported. For instance, [164] proposes to measure the quality of query results by introducing the measure of (user) satisfaction, as some minimal number of query results, and time to satisfaction, i.e. the time that elapsed from the moment of query submission to the moment when the minimal number of query results is computed. Löser et al. [105] propose a list of quality dimensions for semantic overlay P2P networks. The list of dimensions include completeness, accuracy, response time, and amount of data. While giving intuitive definitions of the dimensions, due to their application in the P2P settings, the authors do not discuss how these dimensions are different from those defined in the standard data quality literature.

The problem of the quality of mappings among schemata has been addressed in [18, 79, 86]. In [18] the authors discuss how the quality of mappings in a schema-based P2P system can be assessed when some query arrives in a loop to the same peer. The intuition is that the query in the loop should refer to the same attributes in the schema of the peer, as the original query, which initiated the loop. If this is not the case, then some of the mapping(s) in the loop are incorrect. However, in their analysis, the authors rely on the assumption that each peer's schema is represented with a *single* relational table. Such an assumption cannot be made in the autonomous P2P environment. In [79] the author introduces the idea that

⁴See [162] for a survey of data quality evaluation approaches for centralized ISs.

mappings can be differentiated on the base of how well they serve their primary purpose. For instance, among the various possible mappings from the schema of a product retailer to the schema of a product merchant, those mappings are better which ultimately “sell” more products. The authors of [86] introduce some basic quality metrics, such as completeness and relevance, in the context of a relational P2P IS, and show how certain properties of mappings, such as the presence or absence of projections and selections, may affect the completeness dimension of the quality.

To our best knowledge, the novel idea that the quality of query answers in a P2P IS can be evaluated in the context and can be purpose- and effort-driven, was introduced for the first time in [68], where it was conceptualized as the notion of good-enough answer. This notion was further used in [150] to characterize query answers in ontology-based P2P systems. For the first time an account of the relation of this notion to the standard data quality metrics is reported in this thesis. A lot of research needs to be done to model and quantify the notion of good-enough answer. Currently, it is one of the main research lines of the EU FP6 project OpenKnowledge [14].

Chapter 7

Conclusion

In this thesis we discussed the conventional state-of-the-art information management systems and showed their limitation to develop in the three dimensions of heterogeneity, dynamics, and scalability. We also showed that there is a new emerging class of applications, which require high performance of the IMS in all the three dimensions. We introduced the P2P IMS as a new kind of IMSs which has the potential to overcome the limitations of the conventional approaches.

We identified the main “bottleneck” that prevents the conventional IMSs from scaling in the three dimensions. Even if some of them possess high distribution and autonomy characteristics as well as operate in the point-to-point way, the conventional IMSs are built on the assumption of the global knowledge of the overall system. We showed that this assumption is central to the architecture of these systems and we demonstrated how it limits their capabilities.

We showed that the assumption of the global knowledge cannot be made in P2P ISs, and that the principle of locality is central to this kind of systems. This principle is reflected in the architecture of P2P IMSs, which allows it to define pairwise mappings between system components, and use them for the *transitive* propagation of queries and data between the

components. We showed that the transitivity allows P2P IS to scale while maintaining only local partial knowledge of the whole system at individual components. We concluded, that the two principles of locality and transitivity are the main conceptual and architectural difference that discriminate P2P and conventional IMSs.

We showed that even if they offer many benefits, P2P IMSs pose several research challenges. We identified these challenges and proposed a solution which addresses them. Particularly, we proposed a P2P information management model, which allows it to effectively cope with the complexity introduced by the scalability and heterogeneity dimensions. Apart from this, we proposed a flexible query answering algorithm which supports the processing of cyclic mapping definitions and which is robust in the presence of dynamic runtime changes.

We introduced the problem that standard data quality metrics cannot be used to measure the quality of query answers in a P2P IS. We showed how the distributed subjective nature of P2P ISs brings new dimensions to the quality assessment. To address this problem, we introduced the novel notion of good-enough answers. The notion takes the subjectivity factor into account and proposes quality metrics be parametric on the user efforts made in computing a query answer. Apart from this, it suggests that answers should be evaluated in the context of a single peer and that this evaluation should be parametric on the initial purpose of the query.

Our future work includes a detailed study of how the proposed query answering algorithm can be effectively implemented in lightweight-ontology-based P2P ISs. For these systems, we will also explore how user queries can be classified into local ontologies and how this can be used within the query propagation algorithm. Apart from this, we will investigate the ways for qualifying good-enough answers in the context of database- and ontology-based P2P ISs.

Bibliography

- [1] JXTA project. see <http://www.jxta.org>.
- [2] Groove Networks. see <http://www.groove.net>.
- [3] ICQ. see <http://web.icq.com>.
- [4] Kazaa. see <http://www.kazaa.com>.
- [5] Morpheus. see <http://morpheus.com>.
- [6] Napster. see <http://www.napster.com>.
- [7] SETI@Home project. see <http://setiathome.ssl.berkeley.edu>.
- [8] Skype. see <http://www.skype.com>.
- [9] Resource description framework (RDF). see <http://www.w3.org/RDF/>.
- [10] Semantic Web W3C initiative. <http://www.w3.org/2001/sw/>.
- [11] SPARQL query language for RDF. see <http://www.w3.org/TR/rdf-sparql-query/>.
- [12] DB2 Universal Database (DB2 UDB) v8.2, 2005. See <http://www-306.ibm.com/software/data/db2/udb/>.
- [13] Knowledge Web project (EU IST-FP6-507482). see <http://knowledgeweb.semanticweb.org/>, Jan 2004 – Dec 2007.

- [14] OpenKnowledge project (EU IST-27253). see <http://www.openk.org/>, Jan 2006 – Dec 2008.
- [15] P2P “common interests” search engine and collaboration platform (P2PEOPLE), IST FP5 project (IST-2001-38458), Jul 2002 – Jan 2004.
- [16] ISO/IEC 90759:2000. Information technology – Database languages – SQL – Part 9: Management of External Data (SQL/MED). International Organization for Standardization, 2000.
- [17] Karl Aberer. Guest editor’s introduction. *SIGMOD Record*, 32(3):21–22, 2003.
- [18] Karl Aberer, Philippe Cudré-Mauroux, and Manfred Hauswirth. The chatty web: emergent semantics through gossiping. In *WWW ’03: Proceedings of the 12th international conference on World Wide Web*, pages 197–206, New York, NY, USA, 2003. ACM Press.
- [19] Karl Aberer and Manfred Hauswirth. P2P information systems: concepts and models, state-of-the-art, and future systems. In *proceedings of ICDE*, 2002.
- [20] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [21] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.
- [22] Zahra Afrookhteh. Technical Comparison of Oracle Real Application Clusters 10g vs. IBM DB2 UDB v8.2. White paper, Oracle corp., August 2005.

- [23] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.
- [24] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, and J. Mylopoulos. The Hyperion project: From data integration to data coordination, 2003.
- [25] Yigal Arens, Chun-Nan Hsu, and Craig A. Knoblock. Query processing in the SIMS information mediator. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 82–90. Morgan Kaufmann, San Francisco, CA, USA, 1997.
- [26] C. Axton, R. Gear, N. Macehiter, and E. Woods. Peer-to-Peer computing: Applications and infrastructure. Technical report, Ovum, January 2002.
- [27] Isidro Laso Ballesteros. Collaboration @ work: 3rd wave of internet to foster collaboration between individuals on the seem. In *ICEIS (1)*, pages IS–17–IS–27, 2004.
- [28] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.
- [29] Fabio Bellifimine, Giovanni Caire, Agostino Poggi, and Giovanni Riomassa. Jade - A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.
- [30] Michael K. Bergman. The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, 7(1), August 2001.
- [31] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer

- computing: A vision. In *Fifth International Workshop on the Web and Databases (WebDB)*, Madison, Wisconsin, June 6-7 2002.
- [32] M. Bonifacio, P. Bouquet, and P. Traverso. Enabling distributed knowledge management. managerial and technological implications. *Novatica and Informatik/Informatique*, III(1), 2002.
- [33] M. Bonifacio, R. Cuel, G. Mameli, and M. Nori. A peer-to-peer architecture for distributed knowledge management, 2002.
- [34] Matteo Bonifacio, Fausto Giunchiglia, and Ilya Zaihrayeu. Peer-2-Peer knowledge management. In *the 5th International Conference on Knowledge Management (I-KNOW)*, Graz, 2005.
- [35] Y. Breitbart. Multidatabase interoperability. *SIGMOD Rec.*, 19(3):53–60, 1990.
- [36] M. W. Bright, A. R. Hurson, and Simin H. Pakzad. A taxonomy and current issues in multidatabase systems. *Computer*, 25(3):50–60, 1992.
- [37] Vineet Buch. Database architecture: Federated vs. Clustered. White paper, Oracle corp., February 2002.
- [38] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Query reformulation over ontology-based peers. In *Proc. of the 12th Italian Conf. on Database Systems (SEBD 2004)*, pages 418–425, 2004.
- [39] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

- [40] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 2004)*, pages 241–251, 2004.
- [41] L. Camarinha-Matos, H. Afsarmanesh, C. Garita, and C. Lima. Towards an architecture for virtual enterprises. *J. Intelligent Manufacturing*, 9(2), 1998.
- [42] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *VLDB'99*, pages 397–410, 1999.
- [43] Paul-Alexandru Chirita, Wolfgang Nejdl, Mario T. Schlosser, and Oana Scurtu. Personalized reputation management in p2p networks. In *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*, 2004.
- [44] Intel Corporation. Peer-to-Peer-enabled distributed computing - making the financial services enterprise more productive. Technical report, January 2001.
- [45] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA, 2002. IEEE Computer Society.
- [46] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University, October 2002.
- [47] U. Dayal, E. Hanson, and J. Widom. Active database systems. *Modern Database Systems*, pages 434–456, 1995.

- [48] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *DS-8: Proceedings of the IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics- Semantic Issues in Multimedia Systems*, pages 351–369, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.
- [49] M. Ehrig, C. Tempich, J. Broekstra, F. van Harmelen, M. Sabou, R. Siebes, S. Staab, and H. Stuckenschmidt. SWAP - ontology-based knowledge management with peer-to-peer technology, 2003.
- [50] Marc Ehrig, Steffen Staab, and Christoph Tempich. Platform selection. Deliverable of the SWAP project (EU IST-2001-34103), September 2002.
- [51] Ahmed Elmagarmid, Marek Rusinkiewicz, and Amith Sheth, editors. *Management of Heterogeneous and Autonomous Database Systems*. San Francisco: Morgan Kaufmann, 1999.
- [52] D. Fensel, S. Staab, R. Studer, and F. van Harmelen. Peer-2-Peer enabled semantic web for knowledge management. In J. Davis et al., editor, *Towards the Semantic Web: Ontology-Driven Knowledge Management*, Wiley, 2002.
- [53] Mary Fernandez, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. Xquery 1.0 and xpath 2.0 data model, April 2005. W3C Working Draft, <http://www.w3.org/TR/xpath-datamodel/>.
- [54] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160, New York, NY, USA, 2000. ACM Press.

- [55] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *EDBT International Workshop on Peer-to-peer Computing and Databases (P2P&DB)*, Heraklion, Crete, 2004.
- [56] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. Queries and Updates in the coDB Peer to Peer Database System. In *30th International Conference on Very Large Data Bases (VLDB)*, 2004.
- [57] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB Robust Peer-to-Peer Database System. In *Twelfth Italian Symposium on Advanced Database Systems (SEBD)*, 2004.
- [58] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB Robust Peer-to-Peer Database System. In *2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid)*, 2004.
- [59] Enrico Franconi, Gabriel Kuper, and Andrei Lopatenko. Efficient query processing in dynamic networks of autonomous sources. Technical report, Free University of Bozen-Bolzano, 2005.
- [60] F. Giunchiglia. Contextual reasoning. *Epistemologia*, 16, 1993.
- [61] F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Towards a theory of formal classification. In *Proceedings of the AAAI'05 International Workshop Contexts and Ontologies: Theory, Practice and Applications*, Pittsburgh, USA, July 2005. AAAI Press. ISBN 1-57735-237-8.
- [62] F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Encoding classifications into lightweigh ontologies. In *Proceedings of ESWC'06*, Budva, Montenegro, June 2006. Springer.
- [63] F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review Journal*, (18(3)):265–280, 2003.

- [64] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: An algorithm and an implementation of semantic matching. In *Proceedings of ESWS'04*, 2004.
- [65] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *Proceedings of CoopIS*, pages 347–365, 2005.
- [66] F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Proceedings of Meaning Coordination and Negotiation workshop at International Semantic Web Conference (ISWC)*, 2004.
- [67] F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of the 2nd european semantic web conference (ESWC'05)*, Heraklion, 29 May-1 June 2005.
- [68] F. Giunchiglia and I. Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *6th International Workshop on Cooperative Information Agents (CIA)*, Universidad Rey Juan Carlos, Madrid, Spain, September 18 -20 2002.
- [69] F. Giunchiglia and I. Zaihrayeu. Coordinating mobile databases. In *1st International Workshop on Peer-to-Peer Knowledge Management (P2PKM)*, 2004.
- [70] F. Giunchiglia and I. Zaihrayeu. Coordinating mobile databases: A system demonstration. Demonstration session of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), 2004.
- [71] F. Giunchiglia and I. Zaihrayeu. Implementing database coordination in p2p networks. In *2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid)*, 2004.

- [72] Cheng Hian Goh. *Representing and reasoning about semantic conflicts in heterogeneous information systems*. PhD thesis, 1997. Supervisor-Stuart E. Madnick.
- [73] Jim Gray and Richard Waymire. Sql server megaservers: Scalability, availability, manageability. Technical white paper, March 2003. available at www.microsoft.com.
- [74] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? *WebDB, Workshop on Databases and the Web*, June 2001.
- [75] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1993.
- [76] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.
- [77] N. Guarino. Formal ontology and information systems, 1998.
- [78] Laura Haas and Eileen Lin. IBM federated database technology. Technical survey paper, IBM corp., March 2002.
- [79] Alon Halevy. Why your data won't mix. *Queue*, 3(8):50–58, 2005.
- [80] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
- [81] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787, New York, NY, USA, 2005. ACM Press.

- [82] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.
- [83] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDB J.*, 14(1):68–83, 2005.
- [84] Joachim Hammer, Héctor García-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 532–535, New York, NY, USA, 1997. ACM Press.
- [85] Joachim Hammer and Dennis McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Journal for Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.
- [86] Ralf Heese, Sven Herschel, Felix Naumann, and Armin Roth. Self-extending peer data management. In *BTW*, pages 165–174, 2005.
- [87] Jeff Heflin, James A. Hendler, and Sean Luke. Shoe: A blueprint for the semantic web. In *Spinning the Semantic Web*, pages 29–63, 2003.
- [88] Dennis Heimbigner and Dennis McLeod. A federated architecture for information management. *ACM Trans. Inf. Syst.*, 3(3):253–278, 1985.
- [89] Richard Hull. Managing semantic heterogeneity in databases: a theoretical prospective. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 51–61, New York, NY, USA, 1997. ACM Press.

- [90] K.A. Hummel, G. Kotsis, and R. Kopecny. Peer profile driven group support for mobile learning teams. In *CATE/IASTED Conference*, Rhodes, Greece, June 2003.
- [91] D. Jones, T. Bench-Capon, and P. Visser. Methodologies for ontology development, 1998.
- [92] Sepandar Kamvar, Mario Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of WWW2003*. ACM, 2003.
- [93] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kementsietsidis, and M. Arenas. Coordinating peer databases using eca rules, 2003.
- [94] Vipul Kashyap and Amit Sheth. Semantic heterogeneity in global information systems: The role of metadata, context and ontologies. In Michael P. Papazoglou and Gunter Schlageter, editors, *Cooperative Information Systems*, pages 139–178. Academic Press, San Diego, 1998.
- [95] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: semantics and algorithmic issues. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 325–336, New York, NY, USA, 2003. ACM Press.
- [96] W. Kent. The many forms of a single fact. In *IEEE COMPCON*, San Francisco, 1989.
- [97] M. Khambatti, K.D.Ryu, and P. Dasgupta. Peer-to-peer communities: Formation and discovery. In *14th IASTED Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 166–173, Cambridge, Massachusetts, November 2002.

- [98] Mujtaba Khambatti, Kyung Dong Ryu, and Partha Dasgupta. Structuring peer-to-peer networks using interest-based communities. In *DBISP2P*, pages 48–63, 2003.
- [99] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.
- [100] Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.
- [101] Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.
- [102] W. Litwin, A. Abdellatif, A Zeroual, B. Nicolas, and Ph. Vigier. Msql: a multidatabase language. *Inf. Sci.*, 49(1-3):59–101, 1989.
- [103] Witold Litwin. An overview of the multidatabase system mrds. In *ACM '85: Proceedings of the 1985 ACM annual conference on The range of computing : mid-80's perspective*, pages 524–533, New York, NY, USA, 1985. ACM Press.
- [104] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Comput. Surv.*, 22(3):267–293, 1990.
- [105] Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. Semantic overlay clusters within super-peer networks. In *DBISP2P*, pages 33–47, 2003.
- [106] Barb Lundhild. Oracle real application clusters 10g. Technical white paper, Oracle corp., May 2005.

- [107] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *The VLDB Conference proceedings*, pages 49–58, 2001.
- [108] S. Marsh. Formalising trust as a computational concept, 1994.
- [109] Mark Maybury. Exploitation of digital artifacts and interactions to enable peer-to-peer knowledge management. In I. Zaihrayeu and M. Bonifacio, editors, *Peer-to-Peer Knowledge Management*, online CEUR-WS.org/Vol-108/1568938662.pdf, Aug 2004. CEUR Workshop Proceedings, ISSN 1613-0073.
- [110] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *International journal on Distributed And Parallel Databases (DAPD)*, 8(2):223–272, April 2000.
- [111] Sun Microsystems. JXTA v2.3.x: JavaTM Programmer’s Guide. JXTA project documentation, www.jxta.org, April 2005.
- [112] D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer computing. Technical report, HP Laboratories, Palo Alto, March 2002.
- [113] Paolo Missier and Marek Rusinkiewicz. Extending a multidatabase manipulation language to resolve schema and data conflicts. In *DS-6: Proceedings of the Sixth IFIP TC-2 Working Conference on Data Semantics*, pages 93–115, London, UK, UK, 1995. Chapman & Hall, Ltd.
- [114] H. Garcia Molina and B. Kogan. Node autonomy in distributed systems. In *DPDS ’88: Proceedings of the first international symposium*

- on Databases in parallel and distributed systems*, pages 158–166, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.
- [115] Felix Naumann. From databases to information systems - information quality makes the difference. In *Proceedings of the International Conference on Information Quality (IQ)*, Cambridge, MA, 2001.
- [116] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. Quality-driven integration of heterogenous information systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 447–458, Edinburgh, UK, 1999.
- [117] Wolfgang Nejdl, Wolf Siberski, and Michael Sintek. Design issues and challenges for rdf- and schema-based peer-to-peer systems. *SIGMOD Rec.*, 32(3):41–46, 2003.
- [118] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjørn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 604–615, New York, NY, USA, 2002. ACM Press.
- [119] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario T. Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-peer-based routing strategies for rdf-based peer-to-peer networks. *J. Web Sem.*, 1(2):177–186, 2004.
- [120] Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Peerdb: A p2p-based system for distributed data sharing, 2003.
- [121] Natalya F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, 2004.

- [122] Natalya Fridman Noy, Sandhya Kunnatur, Michel C. A. Klein, and Mark A. Musen. Tracking changes during ontology evolution. In *International Semantic Web Conference*, pages 259–273, 2004.
- [123] Bureau of Labor Statistics. Computer systems design and related services, March 2004. See <http://www.bls.gov/oco/cg/pdf/cgs033.pdf>.
- [124] Beng Chin Ooi, Yanfeng Shu, and Kian-Lee Tan. Relational data sharing in peer-based data management systems. *SIGMOD Rec.*, 32(3):59–64, 2003.
- [125] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O’Reilly, March 2001.
- [126] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, second edition, 1999.
- [127] Alun D. Preece, Kit ying Hui, W. A. Gray, P. Marti, Trevor J. M. Bench-Capon, D. M. Jones, and Zhan Cui. The KRAFT architecture for knowledge fusion and transformation. *Knowledge Based Systems*, 13(2-3):113–120, 2000.
- [128] Wang R. and Strong D. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–34, 1996.
- [129] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
- [130] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM ’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

- [131] Thomas C. Redman. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA, 1997. Foreword By-A. Blanton Godfrey.
- [132] Patricia Rodríguez-Gianolli, Maddalena Garzetti, Lei Jiang, Anastasios Kementsietsidis, Iluju Kiringa, Mehedi Masud, Renée J. Miller, and John Mylopoulos. Data sharing in the Hyperion peer database system. In *VLDB*, pages 1291–1294, 2005.
- [133] Arnon Rosenthal, Len Seligman, and Scott Renner. From semantic integration to semantics management: case studies and a way forward. *SIGMOD Rec.*, 33(4):44–50, 2004.
- [134] Mary Tork Roth and Peter M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 266–275, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [135] Marie-Christine Rousset. Small can be beautiful in the semantic web. In *International Semantic Web Conference*, pages 6–16, 2004.
- [136] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.
- [137] M. Satyanarayanan. Pervasive computing: Vision and challenges, August 2001.
- [138] Monica Scannapieco, Antonino Virgillito, Carlo Marchetti, Massimo Mecella, and Roberto Baldoni. The DaQuinCIS architecture: a plat-

- form for exchanging and improving data quality in cooperative information systems. *Inf. Syst.*, 29(7):551–582, 2004.
- [139] Don Schlichting. MS SQL Server Distributed Partitioned Views. Database journal, www.databasejournal.com, March 2004. Available at <http://www.databasejournal.com/features/mssql/article.php/3319481>.
- [140] Christoph Schmitz. Self-organization of a small world by topic. In Ilya Zaihrayeu and Matteo Bonifacio, editors, *Proc. 1st International Workshop on Peer-to-Peer Knowledge Management*, Boston, MA, AUG 2005.
- [141] Amit P. Sheth. Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In R. Fegeas M. F. Goodchild, M. J. Egenhofer and C. A. Kottman, editors, *Interoperating Geographic Information Systems*, pages 5–30. Kluwer, Academic Publishers, 1999.
- [142] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.
- [143] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, (IV):146–171, 2005.
- [144] John Miles Smith and Diane C. P. Smith. Database abstractions: aggregation and generalization. *ACM Trans. Database Syst.*, 2(2):105–133, 1977.
- [145] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

- [146] D. Stenmark. Information vs. knowledge: The role of intranets in knowledge management. In *HICSS-35*, Hawaii, January 7-10 2002.
- [147] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
- [148] H. Stuckenschmidt, H. Wache, T. ogele, and U. Visser. Enabling technologies for interoperability, 2000.
- [149] Heiner Stuckenschmidt. Approximate information filtering with multiple classification hierarchies. *International Journal of Computational Intelligence and Applications*, 2(3):295–302, 2002.
- [150] Heiner Stuckenschmidt, Fausto Giunchiglia, and Frank van Harmelen. Query processing in ontology-based peer-to-peer systems. In V. Tamma, S. Cranefield, T. Finin, and S. Willmott, editors, *Ontologies for Agents: Theory and Experiences*. Birkhuser, 2005.
- [151] B. Traversat. JXTA: Beyond P2P file sharing, the emergence of knowledge addressable networks. In I. Zaihrayeu and D. Robertson, editors, *Second International Workshop on Peer-to-Peer Knowledge Management (P2PKM'05)*, San Diego, CA, USA, July 2005. Sun Microsystems. Invited talk, www.p2pkm.org.
- [152] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [153] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

- [154] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [155] Michael Uschold and Michael Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64, 2004.
- [156] Mike Uschold and Michael Grninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.
- [157] V. Ventrone and S. Heiler. Semantic heterogeneity as a result of domain evolution. *SIGMOD Rec.*, 20(4):16–20, 1991.
- [158] H. Wache, L. Serafini, and R. García-Castro. Survey of scalability techniques for reasoning with ontologies. The NoE Knowledge Web project (FP6-507482), deliverable D2.1.1, June 2004. available at: <http://knowledgeweb.semanticweb.org/>.
- [159] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In H. Stuckenschmidt, editor, *IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.
- [160] Yair Wand and Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996.
- [161] Richard Y. Wang. A product perspective on total data quality management. *Commun. ACM*, 41(2):58–65, 1998.
- [162] Richard Y. Wang, Veda C. Storey, and Christopher P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):623–640, 1995.

-
- [163] A. White, K. Peterson, and B. Lheureux. New P2P solutions will redefine the B2B supply chain. Technical report, Gartner, February 2003.
- [164] Beverly Yang and Hector Garcia-Molina. Efficient search in peer-to-peer networks. In *ICDCS*, 2002.
- [165] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *IEEE International Conference on Data Engineering*, 2003.
- [166] Ilya Zaihrayeu, Paulo Pinheiro da Silva, and Deborah L. McGuinness. IWTrust: Improving user trust in answers from the web. In *iTrust*, pages 384–392, 2005.