Emergent Schema Management for P2P-based Applications

Felix Heine

2006
Emergent Schema Management for P2P-based Applications

Challenges, Approaches, and Open Issues

Felix Heine

Paderborn Center for Parallel Computing, Paderborn University

Abstract. In p2p based data management applications, it is unrealistic to rely upon a centralized schema or ontology. The p2p paradigm is more than a new underlying infrastructure. It supports an emergent approach to data management where the data is generated and inserted into the network in a decentralized fashion. Thus, each peer or group of peers will have its own schema to store the data. Moreover, the user querying the data will use yet another schema to formulate the request.

The vision of emergent schema management is to resolve these heterogeneities automatically in a self-organizing, emergent way by taking advantage of overlaps and mediators scattered over the network. The emerging schema information can be used in various ways, i.e. to drive the construction of an overlay network, and to route queries through the network.

In this article, we start by explaining the various challenges. We look at the problem both from the viewpoint of the database community describing schemas as entity-relationship models, and from the viewpoint of the knowledge representation community using logic-based formalisms. We then survey existing p2p based approaches dealing with semantics, schemas, and mediation. After describing our own approach to p2p schema management, we conclude with an outlook to open problems in the field.

Keywords. Schema Management, Ontologies, Information Integration, P2P, Data Management, Emergence

1. Introduction

Large scale data management is a challenging task. Through modern information technologies, more and more data and information is available online. However, it is often difficult to find the relevant information, and to efficiently combine the pieces found in various data sources. The problems lie both in the sheer amount of data and in syntactic and semantic heterogeneities.

Even within single organizations, sometimes a large number of different data sources is available. The need to process this data collection as a whole and to draw conclusions from the aggregated information residing in the sources lead to the field of data warehousing. The typical approach is to copy every relevant piece of information into a large,
centrally managed data warehouse (DWH), which is then used for evaluating queries, e.g. in so called decision support systems. See [35] for a deeper discussion of DWH concepts.

However, in many cases the relevant data sources span multiple organizations. In the collaboration of multiple companies, or when mining the information contained in the world wide web, it is necessary to combine multiple heterogeneous information sources which are decentrally controlled. The Semantic Web initiative [11] aims at this goal.

To face these challenges, a system needs to be capable of integrating a huge number of information sources which are syntactically and semantically heterogenous, decentrally managed, and which may be highly dynamic. Two important aspects have to be regarded. First, the underlying infrastructure must be scalable and flexible. Second, the system must handle the heterogeneities of the data sources.

With respect to infrastructure, p2p systems [53] have gained much attention of the research community in recent years. They provide a good basis for large-scale data management systems. Compared to traditional approaches, p2p systems offer good scalability features, combined with decentralized control and flexibility. Within this chapter, we focus on p2p based systems. The integration aspect has attracted researchers both from the database community and from the knowledge representation community, with results ranging from approaches based on entity-relationship models [39] to Ontology-based systems [17] using formal logic. Recently, the strong connection between these issues has been realized and systems turned up [43,30,4,13] which apply the methods of information integration on top of a p2p based infrastructure.

An important idea behind modern p2p systems is the notion of emergence [38]. Typical p2p systems consist of a huge number of relatively small entities, which might be unreliable. They are neither centrally controlled, nor do they have to be professionally managed and supervised. However, the network as a whole provides reliability and other properties which emerge from the collective behavior of the peers. Following this notion of emergence, we expect from such a system that global knowledge emerges from the individual pieces of information which are contributed to the network by the peers.

From the lack of central control we can deduce that there will not be a lingua franca which each peer uses to describe its own data. Even if the p2p network is restricted to a certain domain of knowledge, we cannot assume a fixed standard which suffices to describe any data a peer might want to insert into the network. Additionally, data already stored in legacy databases or knowledge representation systems has to be integrated into p2p systems. Thus the networks require efficient methods to manage heterogeneities in the description of information. Efficiency comprises both low consumption of technical as well as human resources. We consider this aspect crucial for the success of data oriented p2p systems.

Throughout this chapter, we use the term schema for a collection of meta-data used to describe the structure of data stored at some point. We use it in a broad manner subsuming aspects ranging from entity-relationship models to Ontologies based on formal logics [27]. By schema management, we mean any activity related to the storage, exchange, comparison, translation, querying, or other uses of schema information. We further use the terms data, information, and knowledge interchangeably.

Even before the advent of p2p systems, integration issues where important in the fields of knowledge management, data warehousing, and distributed databases; see e.g.
However, the existing approaches are not directly applicable to p2p schema management, as p2p systems pose additional challenges. In the next section, we dive into the problem details of p2p schema management and develop criteria to distinguish and categorize existing approaches. In the following section, we survey and classify the work already published in this area. Anticipating the summary, no currently existing approach is a full-fledged solution to all problems in p2p schema management. Finally, we describe a new, scalable way to schema management based on Semantic Web technologies. We end the chapter with a summary and conclusion.

2. Challenges for Emergent Schema Management

Before heading towards the details, we define the term schema in the context of this chapter, and explain schema management by describing the various usages of schemas in data management systems.

First of all, a schema defines the structure of valid data. It defines which entities, attributes, relations, etc. can be used to insert data, and it may impose integrity constraints on the values of certain instances. However, there are two more important use-cases for schema information. The user needs schema information in order to be able to browse the data or to formulate meaningful queries. Furthermore, the schema information enables query optimization in the query processor [36]. Especially for the query processor, also summary information about existing data like histograms is highly useful. Although not schema information in the strict sense, we also regard this type of meta information within this chapter.

We can envision multiple levels of schema-handling p2p systems. A basic schema awareness requires each peer to commit to a global schema. The next level allows multiple schemas. Each peer’s schema is registered and communicated. However, queries are only routed to peers which share the query’s schema. Further elaborated systems allow to specify mediators which are used to translate either queries or the data in order to answer queries with differing schemas. The final step is the integration of mechanisms for automatic schema matching, in order to reduce human interaction during the design of mediators.

In this section, we describe the basic design choices for emergent semantic systems, followed by a discussion of the types of heterogeneities to be resolved. Different schema integration approaches are mentioned in the subsection 2.3. The origin of the schema and the mediators is discussed in the following subsection, and finally information quality and updates are reviewed.

2.1. Basic Design Choices

When talking about schema management, we first have to talk about the data model. In current p2p solutions, a broad variety of data models is used. We classify the data models in the categories relational, object-oriented, and semi-structured [25]. We further call a data model if it is built using logic-based knowledge representation systems deductive. Examples of the latter are description logics (DL) systems [5]. Relational and object-oriented data models have their origins in traditional database research. Semi-structured
approaches like XML do not expect the schema to be fixed before storing data. Deductive approaches are mostly motivated from the recent development of the Semantic Web which is based on knowledge representation frameworks using formal logic and deduction. However, also the database community has exploited the use of logics for data storage and querying, which has lead to the field of deductive databases. There are no sharp boundaries between these categories, and hybrid models like the object-relational model [25] are possible.

When integrating a web-scale number of different data sources, one also has to think of heterogeneities in the data model. However, no current approach takes this fully into account, as it leads to an enormous complexity. Typically, a system assumes to use an a priori fixed data model. If there are multiple data models allowed, they typically have to be wrapped locally, resulting in a system-wide homogenous data model. Enhancements in the data model are typically out of scope. So it is unclear how to develop the capabilities of a system which is deployed in large scales.

Closely related to the data model is the system model. By this, we mean the degree up to which a system integrates the various data sources. Systems can have individual units consisting of a single data source or a collection of data sources. Each unit has a schema subsuming the individual schemas of the sources. In these systems, individual query results are typically generated by the data stored on a single unit. We coin this model the local data system model, cf. figure 1. The system knows some rules, which allow to translate a query from the schema of one unit to the schema of another unit. With these translations, the query can also be answered by other data sources.

Other approaches look at the entire data scattered over the whole system, and try to answer queries by reasoning with all entries. This approach is shown in figure 2. It mirrors the behavior of a centralized system where each source contributes its local data. We thus call it global data system model. However, the query processing is done in a decentralized fashion. An intermediate approaches is to treat schema information globally, while the instance data is managed locally.
A system supports one or more **query semantics**. Typical semantics are relational algebra, datalog semantics [25], RDF queries [46], RDF-Schema [14] aware queries, or instance checking in description logics systems. Each of these query semantics can be combined with a specific system model. In the local data system model, it is relatively easy to support different query semantics as the system can rely on the local query processors. In the global data system model, more sophisticated query semantics are difficult to achieve as it is hard to identify the data relevant to the current query.

When talking about the data model and the system model, we have to discuss the **closed world assumption** (CWA) and the **open world assumption** (OWA), i.e. the question whether missing information is assumed to be negative, or whether they it is assumed to be unknown. The OWA is a typical characteristic of description logics systems, while relational databases typically use the CWA [5]. For the global data system model, the OWA seems to be appropriate. For the local data system model, we can envision either an OWA or a CWA local to a specific data source.

The system model is closely related to the underlying **infrastructure**. Roughly speaking, there are currently two competing approaches: structured p2p networks and unstructured p2p networks [54]. Unstructured p2p networks follow the idea of the gnutella network. They have been enhanced in various ways, typically by switching from a flat topology to an hierarchical approach like super-peer networks (e.g. [45]), or by introducing semantic overlay networks (e.g. [18]). However, a central problem in these networks remains the question how to find peers which have relevant information for the query without flooding the whole network.

Structured p2p networks are mostly based on the abstraction of distributed hash tables (DHTs) [7]. They are designed to find key-based entries in an efficient and scalable way. Properly used, they can enhance the efficiency and flexibility of query answering. However, as the indices are founded on fixed keys, it is difficult to answer range queries, although there are approaches to solve this problem [3,44]. Also the load balancing might be a problem if there are highly popular keys which either generate a huge storage load or a huge query load, see e.g. [16]. A further disadvantage is a constant network load needed to maintain the DHT itself and the entries in the hash table.
2.2. Heterogeneities

Heterogeneity is a very broad term. Different systems support various types of heterogeneities. In this section, we classify and define the types of heterogeneities relevant for this discussion.

First of all, data sources can have syntactic heterogeneities, e.g. different RDF [41] representations like RDF/XML [9] or N3 [10], or syntactic variants of a query language with the same query semantics. Although important in practice, we ignore these heterogeneities as they are easily resolvable by appropriate conversion tools. Data heterogeneities are differently assigned values for attributes which are otherwise identical. Examples are different keys or different scales for numeric values. The next type are schema heterogeneities, which means that – e.g. in the relational data model – the data is organized in different tables, the attributes have different names, etc. In knowledge representation systems, this means that the knowledge bases use different ontologies. The worst kind of differences of the data sources are heterogeneities in the semantics of data storage or querying. An example is a user who expects the query evaluator to respect transitivity of certain properties, while the answering peer does not support transitive semantics.

2.3. Types of Schema Integration

The possible types of schema management are tightly connected to the different system models. The database community has developed data integration systems [39]. These systems use a mediated schema to provide a uniform interface to various data sources. Additionally, translation rules have to be stored which encode the relationship between the source schemas and the mediated schema.

The central component is the query reformulation algorithm. It takes a query formulated using the mediated schema and translates it into a query using the schemas of the data sources. We call this approach mediator based, as each set of translation rules serves as a mediator between two schemas.

Various formalisms exist to encode the mediators. The basic approaches are LAV (local as view), GAV (global as view), or a combined approach coined GLAV. In the GAV approach, the mediated schema is represented as a set of views over the data sources. The LAV approach is opposite: the contents of the data sources are described as views over the mediated schema.

This type of integration is especially well suited for unstructured p2p networks where each peer can maintain mappings to the schemas used by its direct neighbors in the network. When routing a query via multiple hops through such a network, we build implicitly chains of mappings, thereby generating new mappings which exploit the transitivity of mappings. Piazza [30] and Hyperion [4] are examples for such systems. However, in general a mapping between two schemas is lossy. Thus in long chains of mappings the losses will add up and lead to a highly reduced view of the original data. If we take the possibility of errors in the mediators into account, also the errors might add up in long chains. The Chatty Web approach [1] tries to deal with these problems.

Another way to integrate the data is more relevant to the deductive data model. Here, no individual mappings between two schemas are described. The schemas as well as the instance data is considered to be knowledge, to which deductive algorithms can be applied.
plied. These algorithms infer new knowledge which follows logically from the existing, exploiting so-called intermodel assertions [17]. Thus the schema information is broken down into individual pieces of knowledge, and the mappings themselves are small pieces of mapping information for individual entities. For each situation, the logical calculus tries to find the relevant pieces to construct an answer.

This approach leads to a much higher flexibility in describing, obtaining, and applying mapping information. However, reasoning procedures are typically computationally expensive and difficult to apply in a p2p environment, when combined with the global data system model.

An important aspect when talking about the type of integration is the underlying formalism used to describe the mappings, which determines both the flexibility and expressiveness of the mappings, and the complexity of query answering. Query answering might even be undecidable in general for some formalisms [49,39].

2.4. Origin of the Schema

Within a relational or an object oriented data model, the existence of a schema is typically a mandatory prerequisite to store data. The schema might include integrity constraints which define which kind of data is valid. We can directly access this schema and publish it to the network.

The other data models, notably the semi-structured and the deductive, do not necessarily require a schema. Thus there can be data without a schema. However, due to the data model, this data is always up to some extend self-describing. Thus there is a possibility to generate a schema from the data. However, in contrast to an a priori existing schema, this schema will not impose integrity constraints over the data, it will merely describe which types of data are available. Such a generated schema is known as a Data Guide [26] and serves as a structural summary of the information contained in a database.

In the context of p2p networks, a combination of both approaches can be useful. In case a schema exists, this schema is used and published, else the Data Guide can replace the schema up to a certain extend. Both the existing schema as well as the Data Guide can finally be annotated with statistical information about the data to support query routing and answering.

2.5. Origin of the Mediators

Today, the mediators are typically handcrafted by human experts who have to understand both the semantics of the mediator language and the schema domains. As this is an error-prone and time-consuming procedure, the automatic or semi-automatic generation of mediators is desirable.

There are approaches to automatic schema matching or ontology matching [47,22]. These approaches exploit various techniques like text mining using natural language descriptions of the relevant concepts, structural comparison, key-word detection, etc. Another way is to use the existing mediators to generate new ones, e.g. by chaining, see [30].
2.6. Information Quality

In p2p networks, the maintenance of information quality is a huge challenge. As there is no central authority or control, any peer can push arbitrary information to the network. Thus the network has to cope both with unintentional errors and with malicious attacks and spamming.

This problem exists for the data as well as for schema information. However, malicious schema information might be a much worse problem, because it can lead to wrong interpretation of correct data. Thus a relative small amount of wrong information might cause a huge number of queries to return faulty results.

If the expressivity of the data model is high enough, the network could try to detect contradictions. By this, it is not clear which information is faulty, however it can be detected that there is a problem.

2.7. Updates

Updates in general are an important challenge for p2p data management. In the presence of schema heterogeneities, this becomes even more difficult. First of all, we have to look at the system model and decide who is allowed to update which data. Second, we have to decide whether data can be updated only via their original schema or via arbitrary views.

A simple solution is to leave the update process to the individual peers. So each peer is only responsible for its own data, and nobody else is allowed to update this peer’s data. However, according to the application it might be desirable to allow peers to update also foreign data. Besides consistency problems (see e.g. [8]), this opens new challenges in the schema management field.

We then have to assume that the updating peer uses a different schema than the updated peer. Thus we have to apply mappings during the update process. As with the query process, this can involve multiple steps via different peers.

3. Survey of existing Approaches

Within this section, we survey existing approaches related to the field of p2p based data management which have schema management components. The survey does not aim to be exhaustive; we rather describe selected works which represent different approaches to the problem.

We start with two systems having their roots in traditional database research: Hype- rion [4] and Piazza [31,30]. Their basic assumption is quite similar: each peer holds a collection of physical relations, and associated schema information. It has furthermore a mediated schema, which represents a homogenous view including the peer’s own relations and the mediated schema of the neighbors of this peer. Both systems assume the existence of mapping information. They differ in the types of supported mappings.

Next, we describe the Chatty-Web approach [1], which is a specific solution to the problem of errors and losses during the mediation between different schemas.

Furthermore, we describe some of the systems stemming from the Semantic Web research. GridVine [2], the ICS/Forth RDF Suite [37], and Edutella [43,42,15] are systems based on RDF which employ RDF Schema information. Finally, we mention Bibster [29] as an example of a domain-specific application of schema based p2p systems focussing on the exchange of bibliographic meta-data.
3.1. Hyperion and Piazza

The origins of Hyperion and Piazza are stand-alone data integration systems for relational or XML-based data. Hyperion and Piazza can be seen as a natural evolution step of these systems, moving from a single, centralized mediated schema towards an arbitrary number of peers where each peer runs a local data integration system which integrates both its own data and data from the other peers.

The basic architecture of both systems is similar, see figure 3. Each system consists of peers holding stored relations, which are connected via an unstructured p2p network. In Hyperion, the peer schema is the schema of the stored relations, while in Piazza the peer schema is a mediated schema which spans both over its own stored relations and the peer schemas of other peers.

The relationships between the different schemas are represented through mappings. In Hyperion, the mappings use a GLAV [39] formalism at the schema level, which is supplemented with data mappings at the instance level. Thus, also the relationship between individual entities can be expressed.

![Figure 3. Piazza architecture.](image)

Piazza focuses on the schema level. Two types of mappings are used. First, mappings between the stored relations and the peer relations; second, between the peer schemas. In Piazza, mappings are defined in a language called PPL (Piazza Peer Language). It allows for the following mapping descriptions:

Storage descriptions relate the stored relations of a peer $A$ to its peer schema. They are of the form $A : R = Q$, where $Q$ is a conjunctive query over the peer schema, and $R$ is a relation stored at peer $A$. Thus the content of the relation $R$ is formulated in terms of a view definition over the peer schema, resulting in a LAV formalism. In case the peer relation does not contain every tuple expected in the view, Piazza allows descriptions using inclusion: $A : R \subseteq Q$.

The second type of mappings are peer mappings, which link the schemas of different peers. They can be of different forms. The first form expresses facts about results of queries. They are of the form $Q_1(A_1) = Q_2(A_2)$ or $Q_1(A_1) \subseteq Q_2(A_2)$, where $Q_1$ and $Q_2$ are queries over the peer schemas from a set of peers $A_1$ or $A_2$, respectively. Their meaning is that the evaluation of these queries will always have the same result, or the result of the evaluation of $Q_1$ over $A_1$ will always be a subset of the result of $Q_2$ over $A_2$. Thus they are called equality and inclusion mappings. The second form of peer mappings is a definitional mapping which is a datalog rule over the peer relations.

Both systems use an unstructured p2p network. Thus each peer has a limited set of connections to neighboring peers, and might store some peer mappings relating its data.
to the neighbors data. A peer which receives a query formulated using its peer schema, will reformulate the query using the available mappings, and forward it to the other peers.

In Piazza, a global system catalog containing all mappings is assumed during the query reformulation. The authors have identified this flaw and are working towards a distributed version of the query reformulation algorithm. They plan to use DHT techniques to store a distributed system catalogue.

3.2. Chatty Web

As said in section 2.3, new mappings between different schemas can be generated by exploiting a transitive relationship between existing mappings. However, as translations may be lossy or may have errors, the quality of these new mappings might be poor.

The Chatty Web approach [1] is a way to detect these problems, and to steer the routing decisions in the network according to the observed quality. The underlying system model is the local data system model, typically combined with an unstructured p2p network. Thus each peer or group of peers has its own schema and maintains links together with schema translations to other peers.

The whole network can be seen as a graph, where each group of peers with a common schema are represented by a vertex, while the known translations are the edges. A query which is routed through this graph is continuously translated in order to match the target schema.

An important feature of this graph is the existence of cycles. This means that a node, which routes a query toward another semantic domain, might receive the same query again later. However, the query will have been modified multiple times through various mappings. As the original peer knows also the original version of the query, it is able to compare these two versions and draw conclusions about the quality of the mappings on the used path.

Consider a source node S and a target node T. The source node receives a query using its schema, and forwards the query towards T. Node T expects to receive a query in its own schema. The mappings are on the attribute level and specify how the attributes of S can be expressed as functions over the attributes of T. These mappings are applied to the query, so that it can be evaluated over T’s database.

When a peer receives a query, it detects whether the query has passed a cycle. In this case, the similarity between the original version and the new version is measured with different indicators:

- **Syntactic Similarity**: Here, attributes are counted which are missing in the target query. However, as not all attributes share the same importance for a query, they are weighted with both a user-defined weight and a system-supplied weight in the case of selection-relevant attributes. The system weight reflects the selectivity of the attribute.

- **Cycle analysis**: Here, the correctness of the resulting query is measured. Each attribute may be preserved (positive score), may have vanished (neutral score), or it may have been replaced by a wrong attribute (negative score). Furthermore, the probability of compensating errors is calculated and taken into account.

- **Result analysis**: This analysis checks to which degree a known functional dependency is respected by the query results returned from peers on the cycle that don’t share the same schema.
Subsequently, these measures are used in routing decisions. The user is requested to supply lower bounds for the similarities. Queries are only routed to neighbors where the iteratively updated measures are above these bounds. This ensures that queries will be routed into domains where they are likely to produce valid answers, and prevents flooding.

3.3. GridVine and ICS/Forth RDF Suite

GridVine [2] is an DHT based RDF [41] repository. RDF triples in the network are indexed by subject, predicate, and object. Thus, GridVine can retrieve matches for a triple pattern with a single lookup as long as there is at least one element of the triple known. Based upon this basic lookup primitive, more complex queries are available. GridVine supports the RDQL query language [51].

RDFS [14] is the schema language for RDF and provides the basis for semantic interoperability of RDF based knowledge bases. In RDFS, properties and classes can be defined, and hierarchies can be built. However, as RDFS lacks a way to describe equality between concepts, GridVine borrows the semantics for equivalent properties from OWL [52] to translate queries into other semantic domains. This translation information is also stored using the DHT.

For each query which is received by a peer, this peer looks up translations in the network and can thus reformulate the query to other semantic domains. A translated query is either forwarded to a target peer which then executes the translated query and recursively translates and forwards the query, or the original node stays responsible for the query and iteratively applies multiple translations.

To steer the translation decision and to measure the quality of multiple chained translations, GridVine applies methods from the Chatty Web approach.

The ICS/Forth RDF Suite [37] is also a p2p based RDF store which is aware of RDF Schema. In contrast to GridVine, the RDF triples themselves are not distributed but rather stay at the original node. Thus for an exhaustive answer to a query the system needs to execute the query on each peer which might have answers to the query.

RDF Schema information is used to identify peers which may hold matches for the given query. For this, so-called RVL views are defined which represent parts of the schema graph. A query graph is broken down into multiple sub-graphs; for each sub-graph peers are searched which have matching RVL views. By this, both the RDF Schema inheritance (called vertical subsumption) and sub-graph relationships (called horizontal subsumption) are respected. The main focus of the work is the query planning and execution, driving the breakdown of the query graph and the forwarding of the sub-queries to the target nodes.

3.4. Edutella

The Edutella project [42,15] implements a schema-based p2p system. By this, the authors mean a system which is aware of schema information and uses it in query optimization and routing. Multiple schemas are allowed, but currently no translation is applied to mediate between different schemas. It is up to the user to produce queries which match the schema used to describe the data. However, Edutella is work in progress and aims

\footnote{RDF View Language}
to fill this gap. In the Edutella white paper [43] a mediatior architecture is described in which so-called *query hubs* present views spanning over the data of multiple peers which can be queried using the hub’s schema.

Edutella is based on a super-peer architecture. A small number well-equipped nodes are selected to form the super-peer network, while the other peers connect in a star-like fashion to the super-peers. The super-peers are connected in a so-called HyperCuP \(^2\) topology. In this structure, each super-peer can be seen as the root of a spanning tree which is used for query routing, updates of indices, and broadcasting.

Each peer sends indices of its data to its super-peer. This information can be stored in different granularities, like schema, property, property value range, or property value. However, the index never points to individual data entries but to peers. This kind of index is called an SP-P (super-peer - peer) index.

The super-peers share their index information along the spanning tree structures. Thus each SP also holds several SP-SP (super-peer - super-peer) indices which guide the peer when forwarding the query or parts of it to other super-peers.

Based upon this infrastructure, a query processor and optimizer tries to split the query into multiple parts and to ship the query to target peers which are likely to have results for these parts. The same part of a query might also be shipped to multiple super-peers and peers. The queries may carry code for user-defined operators, so that the operators can be executed on the peer which holds the relevant data.

In order to cope with different query semantics, Edutella defines the *Edutella Common Data and Query Exchange Model* (ECDM). It is based on RDF and is used internally to represent queries and their results. The query language is called RDF query exchange language (RDF-QEL)\(^3\). In the most general form, RDF-QEL queries are datalog queries, with several built-in predicates suited for the evaluation of RDF-based data. In order to cope with peers with limited query processors the query language has different levels, from rule-less queries over conjunctive, disjunctive, up to linear recursive and finally general recursive queries.

### 3.5. Bibster

The Bibster system [29] is an example application of the SWAP\(^4\) project [21]. It is an unstructured p2p system based on JXTA [40] which targets the exchange of bibliographic meta-data like (e.g. BibTeX entries) in academic communities. It is schema aware in the sense that two different schemas are supported. One is the ACM topic hierarchy, the other one is Semantic Web for Research Communities (SWRC) \(^5\).

The information sources are integrated into these schemas a priori by a local component which supports a fixed set of mappings. Thus, no query reformulation is necessary. The classification of the database entries according to the ACM topic hierarchy is used to measure the expertise of a peer. A similarity measure between a query and the expertise of a peer is calculated and used in the routing decisions of the network.

An important aspect of Bibster is the removal of duplicate results. In the field of bibliographic databases, it is likely that numerous peers have overlapping sets of entries.

---

\(^2\)Hypercube P2P

\(^3\)The specification can be found under [http://edutella.jxta.org/spec/qel.html](http://edutella.jxta.org/spec/qel.html)

\(^4\)Semantic Web and Peer-to-Peer, [http://swap.semanticweb.org](http://swap.semanticweb.org)

\(^5\) [http://ontoware.org/projects/swrc](http://ontoware.org/projects/swrc)
However, these entries are not 100 percent identical, but rather similar. Thus, an important task of Bibster is to detect similar entries in the result set of a query and to remove them.

4. RDF Schema based P2P Data Management

Within this section, we describe our own approach, which is motivated from Grid research [24]. In large Grids, resource discovery is a challenging problem. First of all, the size of Grids will grow making scalability an issue. Second, Grids evolve beyond pure networks providing CPU cycles and storage space towards ubiquitous devices integrating any kind of services [6]. Thus the heterogeneity of resource descriptions will increase. No standard will be able to follow the rapid development of new hardware, software, and services and their integration into the Grid.

Thus, we assume that providers will need to extend the resource description schema to suit their needs. Further, resource matchers for Grids need additional information about the resources like compatibility information and dependencies between e.g. a certain program and a licence server.

Although the origins of this work lie in Grid computing, we stress that the developed system is general purpose and can be used also in other scenarios like Semantic Web or management of networks.

Our goal is to realize the global data system model. From the users point of view, the system behaves the same way as a centralized system. It integrates knowledge from all providers and reasons about it. As a small example, we would like the system to integrate compatibility information from node X (Itanium is compatible to Pentium) with resource descriptions from node Y (I have a cluster with Itanium CPUs) to answer a query (I need a cluster with Pentium CPUs or compatible).

In the following subsection, we introduce basic design decision of the system and underlying assumptions. After that, we define the problem in a formal way. The last two parts of this section consider the knowledge distribution in the network and the query processing algorithm.

4.1. Introduction

We chose to build our system upon DHTs as they provide a scalable and efficient way to realize the global data system model. Additionally, we chose RDF and RDF Schema as a knowledge representation framework. We think that more sophisticated Ontology languages like OWL would also be attractive for our work; however, the scalability of OWL reasoners is limited [5].

The important point in the previous section was that the schema knowledge and the knowledge about the resources can be located on different nodes, so that the query answer can only be computed by combining these distributed RDF graphs.

In general, we assume to have \( n \) nodes participating in the p2p network. All of them have some local knowledge stored as RDF triples. They also have local schema knowledge stored as RDF Schema triples. The schema knowledge does not need to be the same for every node. In fact, we are convinced that it is impossible to ensure synchronization of schema knowledge in large world-wide distributed environments or to restrict the
schema to a single common standard. Moreover, it is desirable to allow each node to add locally needed schema information on the fly. If new entities need to be described, new classifications may become necessary. Waiting for a new version of some standard schema does not solve this problem.

However, we assume that there is an ontology which serves as a common schema, at least for some subsets of the nodes. This ontology will be the basis which can be extended locally. Additional schema knowledge may be stored to allow translation from one ontology to the other. Without such common understanding, no interoperability would be possible.

![Figure 4. Virtual pool of knowledge.](image)

Our desired result is to put all this knowledge from all the nodes virtually in one pool, apply RDFS entailment rules to this pool and evaluate queries with respect to the union of the knowledge, see figure 4. This approach is very beneficial, as overlaps in the schema knowledge are used to build bridges between different schemas used by different nodes. A query is formulated as a pattern consisting of multiple triples where parts of the URI references and labels are replaced by variables. We call the RDF graph resulting from the union of the local knowledge and appliance of the entailment rules the model graph, while we call the query pattern the query graph.

### 4.2. Formal Problem Definition

Now we define the foundations more formally. Both the model and the query graph are directed graphs. The labels of the model graph can be URI references or XML literal values, or blank node labels. For the following discussion, we do not have to differentiate between URI references and XML literals, so we define the set of labels to be $\mathcal{L}$ which contains both types of entities. The set of blank node labels is denoted by $\mathcal{B}$. Thus each vertex is labelled with an element of either $\mathcal{L}$ or $\mathcal{B}$. An edge of an RDF graph cannot have a blank label, so only elements of $\mathcal{L}$ are allowed here. RDF permits multi-edges, i.e. more than one edge between a pair of nodes, and no stand-alone vertices are allowed, so the graph can be described as a triple set
The query graph is defined analogously. However, instead of blank node labels, we use variables from a set $V$ of variables, and we allow edges to be labelled with variables. Thus the query graph in triple representation is

$$T_Q \subseteq (L \cup V) \times (L \cup L) \times (L \cup V)$$

For convenience, we denote the set of variables occurring in $T_Q$ by $V_Q$, and the set of literals occurring in $T_Q$ by $L_Q$. The sets $L_M$ and $B_M$ are defined analogously.

The desired semantics for our query evaluation are as follows: given a model graph $T_M$ and a query graph $T_Q$, find every mapping for the variables occurring in $T_Q$ to the set of blank nodes and literals occurring in $T_M$, such that for each triple in $T_Q$ there is a matching triple in $T_M$. Thus we search for mappings

$$R : V_Q \rightarrow L_M \cup B_M$$

such that for every triple $(s, p, o) \in T_Q$ there is a triple $(s', p', o') \in T_M$ such that:

$$s \in V_Q \Rightarrow s' = R(s) \quad s \in L_Q \Rightarrow s' = s$$

$$p \in V_Q \Rightarrow p' = R(p) \quad p \in L_Q \Rightarrow p' = p$$

$$o \in V_Q \Rightarrow o' = R(o) \quad o \in L_Q \Rightarrow o' = o$$

Note that this definition includes the possibility to match two different variables to the same value, as we do not insist on $R$ being an injective function, which makes the problem a bit different to the subgraph isomorphism problem (see [56]). We impose two restrictions upon $T_Q$. First, we expect it to be connected (not strongly connected). This is natural, as we can break the query evaluation of an unconnected query graph in multiple evaluations of the connected components. The result set for the whole query is determined by enumerating every combination of the results for the connected components. Second, we expect that there exists at least one triple in $T_Q$ having at least one labelled element. One of these labels will serve as a starting point for the query evaluation.

4.3. Knowledge Distribution

In order to query the knowledge, we have to pre-distribute the RDF triples to well-defined nodes in the network to prevent flooding. We also pre-evaluate the RDFS rules and distribute the resulting triples. By this, the query evaluation does not have to regard the reasoning any more. In this section, we describe these mechanisms.

4.3.1. Triple Distribution

The general architecture is shown in figure 5. Each node $i$ has initially stored a set of RDF triples, which contains both schema knowledge $SK_i$ and local knowledge $LK_i$.

In order to be able to query the knowledge, we have to have a way to find relevant triples for the query, as we do not want to query every node in the triple. For this purpose, we connect the nodes via a structured peer-2-peer network which implements a distributed hash table [7].
There are different DHT approaches available like Chord [55], Pastry [50], or CAN [48]. All have some kind of lookup mechanism in common. This lookup mechanism enables the user to determine a specific node which is responsible to store data for a certain key.

In our scenario, we use the URI references respectively the XML literals as keys. We store each triple three times, indexing by the subject, predicate, and object. Thus each node sends out its own triples to the responsible nodes. The target nodes store the triples for later retrieval. Note that we assume blank node labels to be unique in the network. This can simply be achieved by adding a node identifier to the label. Thus we can assure that we can join the triple-sets without caring about the blank node labels.

Thus, after finishing this process, the whole model graph is accessible in a well-defined way over the DHT network. There are several ways to retrieve triples from the network. To retrieve a set of triples, at least one part of the triples must be fixed. We use this part as a key to the DHT network, retrieving all triples with this value.

We define three functions, $\text{getBySubject}$, $\text{getByPredicate}$ and $\text{getByObject}$, which we use to retrieve sets of triples. As an example we describe the $\text{getBySubject}$ function. It takes a label as input and retrieves all triples from the network where the subject equals this label. It calls the lookup operation of the DHT network to retrieve the network node which stores these triples. So the execution time of these functions is determined by the time the lookup operation takes plus the transfer time of the result set. It will be a central goal of the query algorithm to minimize both the number of calls to these functions and the size of the returned triple sets.

### 4.3.2. RDFS rules

The RDF semantics document [32] describes how RDFS entailment can be viewed as a set of rules which generate new RDF triples from existing ones. For our scenario, the taxonomy-related rules are most important. First, they ensure that the subClassOf and subPropertyOf relationships are transitive. Second, they propagate instances of classes and properties towards more generic classes and properties. As an example, the class-related rule states:

\[ \text{If} \ X \ \text{is a sub-class of} \ Y, \]
and A is an instance of X, then A is also an instance of Y.

The pre-conditions of all rules share at least one URI in common. Thus, there is always at least one node were all triples are locally known. This means that all RDFS rules can be evaluated locally without network interaction. However, the resulting triples have to be further distributed to the responsible nodes.

See [33] for a discussion about the length of this process and the message load generated by it. Although this discussion focussed on taxonomies generated from DL reasoners, it applies as well to the RDFS taxonomies we are using here.

The transitivity of the subClassOf and subPropertyOf predicates is supported implicitly. The triples are not generated, but the taxonomy rules implicitly propagate instances or pairs of instances to every class / property in the transitive closure of the subClassOf / subPropertyOf relation.

4.4. Query evaluation

Our algorithm for evaluating the queries works in two phases. In the first phase, we determine candidate sets for each of the triples in the query graph, as well as for the variables. The candidate sets for the variables and for the triples are mutually dependent, thus we have a refinement procedure which successively removes candidates from both sets which are not suitable. In the second phase, matching combinations of triple candidates are searched locally.

Thus, the first phase collects a subgraph of the model graph distributed over the network which is large enough to contain every result for the query. The second phase is a subgraph matching in this smaller, local model graph which reveals the final results of the query.

4.4.1. Determination of Candidate Sets

The task of this phase is to identify parts of the model graph which are relevant for the query. The main focus of the algorithm is to reduce network load during this phase. This means, that we want to contact as less nodes as possible, and to transfer a minimal amount of data. We present different strategies, which are compared in the following section.

The main idea behind the algorithm is to determine how much candidates are expected for each triple and to iteratively choose the triple with the smallest expected candidate set. At each time, the algorithm maintains a set of candidates for each triple, denoted \( C_T(t), t \in T_Q \) and a set of candidates for each variable denoted \( C_V(v), v \in V_Q \). Candidate sets may be undefined. As a short-cut, we will write \( C_V(v) = \Delta \) iff the candidate set for \( v \) is not defined. We furthermore define \( |C_V(v)| := \infty \) iff \( C_V(v) = \Delta \).

As it will simplify the algorithms presented later, we further define the candidate set of a fixed value (either literal or URI reference) to be the one-element set containing that value: \( C_V(x) = \{ x \} \) iff \( x \in L \).

Then some network communication will be used to retrieve the candidate sets, leading to new estimates for the other triples. We use the notion of the specification grade of a triple to see where we expect the smallest communication overhead. If you look at the way we distribute the triples, we can either use the subject, the predicate, or the
object to retrieve the candidates. Each of these can either be a variable or a fixed value. If it is a fixed value, we have to do a single lookup to retrieve the candidate set. If it is a variable, the number of lookups is determined by the current number of candidates for this variable. If there are no candidates for the variable so far, we cannot use it to retrieve a candidate set.

Thus we define the specification grade of a triple’s element as follows:

\[
sgI(x) = \begin{cases} |C_V(x)| & : C_V(x) \neq \Delta \\
\infty & : C_V(x) = \Delta 
\end{cases}
\]

Due to our above definition, this can be written short-hand as \(sgI(x) = |C_V(x)|\). The specification grade for a triple is the minimum specification grade of its elements:

\[
sgI(\langle s, p, o \rangle) = \min(sgI(s), sgI(p), sgI(o))
\]

The idea behind this definition is that the specification grade determines the number of lookup operations needed. Thus we can write down the algorithm:

**function** candidates\((T_Q, T_M)\)

set each \(C_T(t)\) and \(C_V(v)\) to \(\Delta\)

while there is an undefined \(C_T(t)\)

- determine a triple \(t = \langle s, p, o \rangle\) where
  - \(C_T(t) = \Delta\), and
  - \(sgI(t) \leq sgI(t') \forall t'\) with \(C_T(t') = \Delta\)

  if \(sgI(t) = sgI(s)\)
    \[C_T(t) := \bigcup_{x \in C_V(s)} \text{getBySubject}(x)\]
    \[C_T(t) := \{ \langle s, p, o \rangle \in C_T(t) : p \in C_V(p), o \in C_V(o) \}\]
  else
    similar code for predicate and object
  end if

if refine\((C_T, C_V, \{t\}, \emptyset)\) = error
  return error
end if
end while
return ok
end function

The heart of the algorithm is the refinement procedure. There are two ways of refinement. First, we can look at a variable’s candidate set. We compare it with the candidate sets for each triple where this variable occurs. If a candidate does not occur within the triple candidate set, it has to be removed from the variable candidate set. The other way around, we look at the candidate set for a triple and remove any candidates where there is some value not within the matching variable’s candidate set. We always keep track of the set of changed variables \(V\) and changed triples \(T\), so that we do not have to check every set.

**function** refine\((C_T, C_V, T, V)\)

while \(V \neq \emptyset\) or \(T \neq \emptyset\)

for each \(t = \langle s, p, o \rangle \in T\)
if $s \in V$
  $C_V(s) := C_V(s) \cap \text{subject}(C_T(t))$
  if $C_V(s)$ has been changed
    $V := V \cup \{s\}$
  end if
end if

similar code for predicate and object

$T := T - \{t\}$

end for

for each $v \in V$
  for each $t \in T_Q$
    if $\text{subject}(t) = v$
      $C_T(t) := \{(s', p', o') \in C_T(t) : s' \in C_V(v)\}$
      if $C_T(t)$ has been changed
        $T := T \cup \{t\}$
      end if
    end if
  end for
end for

$V := V - \{v\}$

end for

if some $C_V(v)$ or $C_T(t)$ is empty
  return error
end if

end while

return ok

end function

The crucial question is how we can further reduce the network load. The refinement procedure is uncritical, as it works completely local. Thus we have to look at the order in which the triple candidates are retrieved from the network. The definition of the specification grade as given above ensures a minimal number of lookup operations in the current step. However, it can lead to a large number of candidates for the triple, leading to both a high bandwidth consumption and a large number of lookups in further steps. Furthermore, if we already have candidates for other variables in the triple, we can use these candidates to reduce the size of the returned candidate set. In the following two subsections, we introduce methods to benefit from these ideas.

4.4.2. Look-Ahead

The first enhancement is to introduce a look-ahead for the candidate set size in order to choose the next triple during the first phase of the query evaluation. We implement this look-ahead by summing up the result set sizes for each lookup instead of only counting the number of lookups. This is a trade-off, as it results in further lookups during the calculation of the sg value, however, it might lead to a better path through the query graph with fewer candidates to transfer.
We define the following functions to retrieve the needed statistical information: \(\text{cntBySubject} \), \(\text{cntByPredicate} \), and \(\text{cntByObject} \), respectively. They work similar to the \(\text{getBySubject} \) etc. functions, however, instead of returning the triple set, they only return its size.

The new definition of the specification grade is as follows. We have to define three different functions, referencing to the three elements of a triple. We only describe the subject function \(\text{sgs} \); \(\text{sgp} \) and \(\text{sgo} \) are analogous.

\[
\text{sgs}(x) = \left\{ \begin{array}{ll}
\sum_{s \in C_V(x)} \text{cntBySubject}(s) & : C_V(x) \neq \Delta \\
\infty & : C_V(x) = \Delta
\end{array} \right.
\]

The specification grade of a triple is now defined as

\[
\text{sg2}(\langle s, p, o \rangle) = \min(\text{sgs}(s), \text{sgp}(p), \text{sgo}(o))
\]

During the collection of the candidate sets, we retrieve the sg of a triple multiple times. Thus we defined a further version \(\text{sg3} \) which implements a cache which is valid during the evaluation of a single query. Thereby, we can reduce the overhead introduced by the additional lookup-operations.

4.4.3. Bloom Filters

When we retrieve candidates for a triple \(\langle v_1, v_2, v_3 \rangle \), we have to choose an element whose candidates we use as a key for the DHT lookup. Assume we choose \(v_1 \) which has the candidate set \(\{x, y, z\} \). Then we contact three nodes by using \(\text{lookup}(x)\), \(\text{lookup}(y)\), and \(\text{lookup}(z)\). These nodes return all triples where the subject is either \(x\), \(y\), or \(z\).

We might also have candidates for the other two variables. If we further transfer the known candidates for \(v_2\) and \(v_3\) during the \(\text{getBySubject} \) function, the target nodes could reduce the result sets. However, also the candidate sets for the other variables might be large, so that the reduction of the result set is outweighed by the additional transfer of the candidate sets.

Bloom filters [12] are ideal for this situation. A bloom filter is a compact representation of a set using an array of bits of a fixed size. Each element which is stored in the filter is hashed multiple times using different hash functions. The bits corresponding with the hash values are set in the filter. Membership test is done in the same way. Thus, each element of the set is reliably detected. However, so-called false positives are possible. This means, that an element is detected to be a member of the set which is in fact a non-member.

Thus we can encode the candidate sets for the other variables as bloom filters and send these filters to the target node. This node locally sorts out non-matching results and sends back the reduced candidate set. Due to the false positives of the bloom filter, there may be too many candidates, however, they are removed by the refinement procedure. As each set member is reliably detected, no candidate will be lost, which ensures the correctness of the query results.

The bloom filters can be used both in the \(\text{getBySubject} \) etc. functions and the \(\text{cntBySubject} \) functions. The former can be combined with all versions of the specification grade; the latter results in a new definition of \(\text{sg4} \), which results in a better look-ahead as the already known candidates for the other elements of the triple are included. We combine the \(\text{sg4} \) version with the caching mechanism of \(\text{sg3} \). The cache is filled as soon
as a $sg4$ value is calculated. This means, that we might later loose better estimates when we have received candidates for more variables. However, the caching effect reduces the number of lookups dramatically.

4.4.4. Final Evaluation

After having retrieved candidate sets for all triples and variables, we have to do the final evaluation to retrieve matches for the query. This is done completely local. However, it can be computationally expensive. In general, every combination of candidates for the triples has to be considered and tested, which are exponentially many. In fact, the query complexity version of RDF querying is NP complete. However, the data complexity version is in P, see [28].

We employ a backtracking algorithm. In each step, it chooses a triple which has more than one candidate, and fixes each of the candidates in a loop. By fixing a triple candidate, we also fix values for the variables. Triple candidates are only chosen if they don’t contradict with previous variable assignments. For a full description of the algorithm, see [34].

5. Conclusion and Open Problems

In this chapter, we have given an overview of the field of Emergent Schema Management. With an ever increasing number of online-available information sources, the need for integration of these sources rises. The main challenge in this integration is to automatically overcome heterogeneities on different levels while maintaining scalability.

We have provided a classification of the basic design choices, of the types of heterogeneities, and of the types of schema integration. We have further discussed the origins of schema information and mediators, and briefly touched the topics of information quality and distributed updates to the information sources.

A number of case studies shed some light on a selection of current research in this field, including our own work. Each of these projects focusses on a specific subset of the problem; none of them is a full fledged solution resolving all main challenges.

In the remainder of this section, we list what we feel are the most challenging open problems to be resolved by future research.

- **Sophisticated logics**: To integrate arbitrary pieces of information scattered over the globe, expressive logics such as the various flavors of Description Logics are desirable. They are expected to play an important role in the forthcoming Semantic Web. However, complex reasoning procedures hinder scalability. In such scenarios, it is difficult to identify the pieces of information relevant for a given query.

- **Global data system model**: To detect arbitrary connections between seemingly unrelated information sources, and to take full advantage of the stored datasets, we feel that the global data system model is superior to any local approach. However, it typically needs a kind of virtual global catalog, which is then distributed over a DHT network. The maintenance of this catalog becomes increasingly difficult in the light of an highly dynamic environment.
Automated construction of schema and mappings: The success of any integration system relies on the existence of schema information and mappings between these schemas. A typical assumption is that this information is human supplied. However, the larger the whole system grows, the more important it will be to generate reliable mappings automatically, either by combining existent mappings or by automated approaches of schema or Ontology mapping.

Trust and Security: The assumption that each peer in the network behaves well will not hold in larger, open scenarios. Malicious peers will try to influence the behavior of the system to gain advantages. Thus we think that trust and security will be important aspects of practical systems which are deployed beyond small research communities.

Dynamics: Information is not static. Thus also the systems integrating various sources need to respect the dynamics of the underlying data. A system will be the more useful the more recent the results are. However, as caching and replication are basic elements of most systems, this poses additional challenges resolving the pay-off between freshness and performance.

Concluding, we believe that the research activity in this area already has achieved highly useful results forming a stable basis for further work to resolve the remaining issues.

Acknowledgements

Partially supported by the EU within the 6th Framework Programme under contract 001907 “Dynamically Evolving, Large Scale Information Systems” (DELIS).

References


24

F. Heine / Emergent Schema Management


[48] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A


