Accessing XML Documents using Semantic Meta Data in a P2P Environment

Dominic Battré and **Felix Heine** and **André Höing**
University of Paderborn
Paderborn Center for Parallel Computing
Fürstenallee 11, 33102 Paderborn, Germany
{battre, fh, andrehoe}@upb.de

Giovanni Cortese
Interplay Software
C.so III Novembre 166 Trento Italy
g.cortese@ipsoft.it

Abstract

XGR (XML Data Grid) and BabelPeers are both data management systems based on distributed hash tables (DHT) that use the Pastry DHT to store data and meta data. XGR is based on the XML data model; BabelPeers uses the Resource Description Framework (RDF) for its data. XGR and BabelPeers have different but complementary functionality. On the one hand, XGR focuses on document-based storage of XML data and publish/subscribe mechanisms, on the other hand, BabelPeers focuses on query strategies that combine pieces of information originating from various sources and provides reasoning about the information. Thus it is valuable to research how the two concepts can be merged to get the best of both worlds.[1]

Introduction

In the time of the Semantic Web and Web 2.0, the meaning of semantically rich meta data grows every day. The Resource Description Framework, defined by the W3C, established itself to the de facto standard to represent information in the semantic web. But it is not limited to represent meta data for web content. It can also help sharing information between all kind of organizations, from private persons to global players.

XML is a flexible data format, which is still very important for data exchange and storage, e.g. Open Office documents or SOAP1.2 (Gudgin *et al.* 2003) data are represented by XML. This paper combines two technologies: XGR, a distributed XML database, and BabelPeers, a distributed RDF database and inference system. We create a scalable distributed P2P-based XML data storage system, which supports arbitrary meta data information about the content. This meta data helps users to find the desired XML documents, and additionally they do not need to care about the underlying XML structures.

We use XML as data model for the pure data and RDF as data model for the meta data. Thus RDF can be used to encode schema information on a higher level than e.g. XML Schema. XGR itself uses its own set of meta data where types and indices are described. This information can be augmented when RDF is used in conjunction with the reasoning capabilities of BabelPeers.

XGR can save a lot of information like large XML documents. These documents are saved locally on a node and can be accessed by an index structure, so the network traffic is not very high. Furthermore, the client can formulate precise queries as XPath expressions to receive just a small fragment of the complete XML document which also reduces network traffic. The big disadvantage is that a client has to know the XML structure of the document to formulate these precise queries.

BabelPeers stores all information as RDF triples and has to distribute the complete knowledge over the network. This produces a lot of network traffic and is probably inefficient if we need information which are naturally connected with other knowledge, e.g. an address always consists of a name, street etc. But BabelPeers can manage semantically rich data. Thereby concepts can be set in relation to other concepts and this provides new possibilities for the XGR application.

As an example, consider the piece of RDF[2] encoded meta data about addresses given in Figure 1. It encodes via XPath expressions how an address can be found for various XML schemas. The addresses themselves are classified via RDF-Schema (RDFS), as private addresses or company addresses. Through the reasoning, both of the addresses are classified as general addresses, visualized via the dotted arrows.

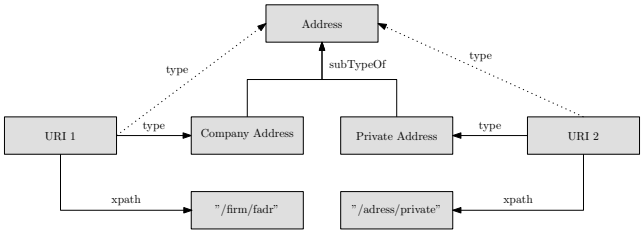

Figure 1: RDF Meta Data for XML.

A query over this meta data is given in Figure 2. The query asks for XPath expressions for an address in any avail-

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

[2]For an easier understanding, we use a simplified RDF notation with shortened URIs

able XML document. Through the RDFS reasoning explained in the section "BabelPeers", the query will yield both "/firm/fadr" and "/address/private".

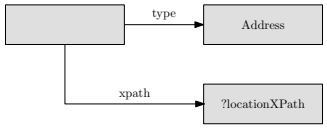

Figure 2: Query for RDF Meta Data.

By managing meta data in RDF triples we gain the possibility to generalize queries and to hide the possibly complex XPath structure behind clearly defined terms.

This paper is structured as follows. The next section gives an overview about the related work. The following sections, "DHT Networks", "BabelPeers", and "XGR", introduce our researches, which this paper combines. The description about the integration and how the different applications are connected is given in section "Integrating XGR and BabelPeers". That section also shows the advantages resulting from the integration. The last section "Conclusion" summarizes the results and gives ideas for further use cases.

Related Work

The two main themes of this paper are P2P-based data management and the possibilities, which we gain by using a semantic description language as RDF. So the related work limits to these.

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 want to introduce the projects Hyperion (Arenas *et al.* 2003) and Piazza (Halevy *et al.* 2003), which represent different approaches to the problem.

The origins of both 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. 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.

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.

RDFPeers (Cai *et al.* 2004) and Atlas (Koubarakis *et al.* 2006) are based on structured P2P networks, in particular distributed hash tables, the same kind of network BabelPeers uses. They also work with RDF and distribute triples in the same way. The big disadvantage is that they do not support RDF Schema reasoning.

DHT Networks

To understand the employed technologies, BabelPeers and XGR, we give a short introduction to DHT-networks.

A distributed hash table (DHT) is comparable to a regular hash table with efficient insert and lookup operations, except that the ID space of a DHT is distributed between the nodes of a peer-to-peer network. This network is able to route insert and lookup requests to those nodes which are responsible for the respective fractions of the ID space in $\mathcal{O}(\log N)$ routing steps with high probability, where N is the number of nodes in the network.

Each node of the peer-to-peer network is assigned a unique position in the ID space. In case of Pastry (Rowstron & Druschel 2001), the DHT used by BabelPeers and XGR, the ID space is a ring $0 \ldots 2^{128} - 1$. The concrete distribution of IDs to nodes depends on the DHT implementation. In Pastry, the IDs of the ring that are closest to a node's position are mapped to this node. In Chord (Stoica *et al.* 2003), those IDs that follow a node's position on the ring up to the following node are mapped to this node. Additionally, Pastry provides some other services, e.g. replication.

Like in a regular hash table, we insert attribute value pairs. The network derives the hash value of the attribute and sends both, attribute and value, to the node which is responsible for the hash value. The lookup operation works similarly. We hash the attribute and send the lookup to the node which is responsible for the hash value. Figure 3 presents an example, how we insert an RDF triple in BabelPeers. This figure is described below in detail.

BabelPeers

Generally, BabelPeers can not only manage meta data, but also other information, given as RDF triples. An RDF triple can be regarded as a sentence, which consists of subject, predicate, and object. The knowledge can be seen as a directed graph. The subjects and objects are the vertices of the graph and for every triple, there is one edge, labeled with the predicate, directed from the subject to the object.

BabelPeers is peer-to-peer based and every peer shares his knowledge with all other nodes. The network fits all these parts together into one big knowledge base which is available for every peer. The following paragraphs describe how this knowledge sharing works.

After the distribution of the RDF triples, each triple is saved on three different peers, identified by the hash values of the triple's elements: $hash(Subject)$, $hash(Predicate)$ and $hash(Object)$. Figure 3 shows a five node Pastry network and how we insert an RDF-Triple.

By distributing the RDF triples that way, it is guaranteed that all triples, that share a common URI at arbitrary positions merge on a single node in the network. This is a precondition for the RDFS inference rule system, described below. Furthermore every triple is accessible already, if we only know one of the three elements.

Figure 4 presents the view, which each node has to the network. The peers do not know where triples are stored, even the triples, which they have inserted. They only calculate the hash value and send a lookup request to the network.

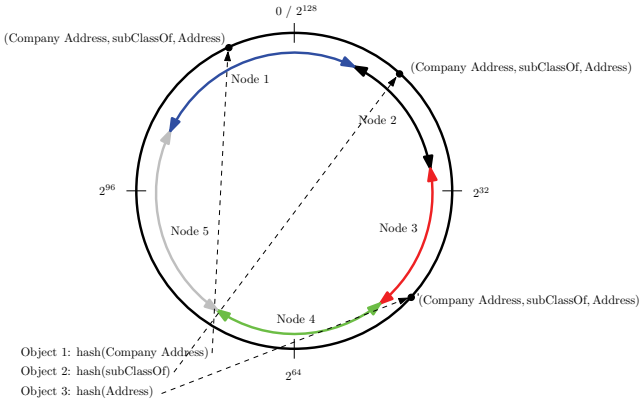

Figure 3: Inserting RDF-Triples in a Pastry Network

So each node has the same way to query knowledge and so the network seems to provide one big knowledge base that incorporates the complete knowledge of each peer. So it is possible to infer new triples (information) which was not possible without sharing information. In Figure 4, this is shown by the dotted edges.

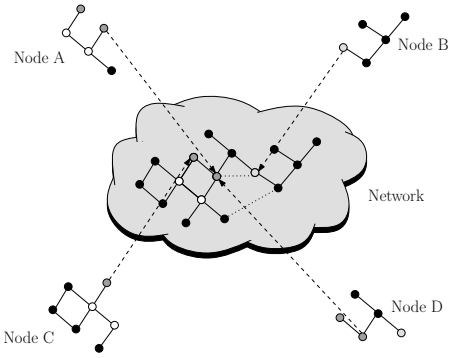

Figure 4: Sharing knowledge with BabelPeers

The RDFS specification gives us a set of inference rules to extend the explicit RDF knowledge. Each node executes a forward chaining algorithm to generate all derivable knowledge by using this RDFS rule set. Most of the RDFS rules map a single triple to a new one, but some rules contain two triples in the precondition. This is no problem, because of the dissemination described above. There is always a single node which handles all triples of a specific URI. A closer look to the rules shows that this dissemination suffices to fulfill all rule preconditions.

Figure 1 shows an RDF reasoning example. We derive a new triple, which states that the "URI 1" which is a "Company Address" is also an "Address". This new triple will be distributed over the network and so the derived information is available for all other nodes. So the inference system works without generating a lot of network traffic. Our paper (Battré *et al.* 2006) describes this in detail.

Clients can connect to any peer and pose questions formulated in SPARQL. They connect via a TCP connection. Until now, just a subset of SPARQL is available, but it fulfills our needs. A query consists of a set of triples including variables. The peer tries to find a binding for these variables by collecting all possible candidates for every query triple and processing a pattern matching as the second step. Finally the node sends all valid bindings back to the client. Figure 5 shows the query, seen in Figure 2, as SPARQL code. More detailed information about the query evaluation can be found in the paper (Heine 2006).

```
SELECT ?indexPath WHERE {
  ?var type Address .
  ?var xpath ?indexPath .
}
```

Figure 5: Example – SPARQL query

We have shown that BabelPeers combines the semantic knowledge of every single peer to one knowledge base, accessible for all nodes. Additionally, it generates the complete derivable knowledge and makes it available network-wide. Altogether it is the right choice to manage a huge number of RDF triples and to use this information in a semantic context.

XGR

XGR is peer-to-peer based, too, but aims for different goals than BabelPeers. It realizes a distributed database for XML documents, relying on storage resources provided by a network of collaborating peers. Depending on the application scenario, XML data can be stored in a local storage at the peer which performed the insert operation, and only indices are stored in the DHT-based storage, but it is also possible to store both, data and indices, in the network, e.g., for replication purposes.

An XGR datatype defines one kind of document with a specific root element (set by the `xgr:id`-tag). It furthermore specifies a primary key (set by the `xgr:pkey`-tag) that can be used to address one specific XML document. Further indices can be defined by the `xgr:index`-tag. Indices allow efficient lookups of XML nodes (addressed by XPaths) that store certain values. Figure 6 shows an example datatype definition of an address.

```
<xgr:datatype xmlns:xgr="XGRSchema">
  <xgr:id>ns:address</xgr:id>
  <xgr:pkey>ns:id</xgr:pkey>
  <xgr:desc>Address</xgr:desc>
  <xgr:index>
   <xgr:name>name</xgr:name>
   <xgr:path>
     /ns:address/ns:name
   </xgr:path>
  </xgr:index>
</xgr:datatype>
```

Figure 6: Example – datatype definition of an address

In this example, every address data item has to start with an `ns:address`-tag. The `xgr:pkey`-tag specifies the primary key, `ns:id`, used to build the primary index. Furthermore we can define further indices by using XPath expressions inside the `xgr:index`-tag. In this example, the name is a second index.

After registering the datatype above, the system is ready for inserting addresses. Figure 7 shows an example address XML document, which represents the address of our institute.

```
<ns:address xmlns:ns="AddressSchema">
   <ns:id>1</ns:id>
   <ns:name>PC2</ns:name>
   <ns:street>
      Fuerstenallee 11
   </ns:street>
   <ns:city>Paderborn</ns:city>
   <ns:zip>33102</ns:zip>
</ns:address>
```

Figure 7: Example – address insertion

XGR provides means to formulate queries (e.g. for addresses of named entities) as simple XPath expression and returns the appropriate XML fragments. We defined two indices above, the primary key `ns:id` and a second index `ns:name`. For fast querying, we can use these two indices, which are hold distributed. A DHT-lookup gives us the nodes where we find relevant XML fragments. XGR provides also the possibility to look for XPath expressions, which are not indexed. However, the network has to ask every node for adequate information and so this is very expensive and slow.

XGR offers different possibilities for data retrieval. The user can simply use XPath expressions for describing the desired information (see paragraph above). For our running example, the query XPath expression `/ns:address[ns:name="PC2"]` returns the PC2 address we added above, and alternatively `/ns:address` returns all known addresses. Furthermore, XGR implements a publishing/subscriber system (Emiliano Casalicchio, Federico Morabito, Giovanni Cortese, Fabrizio Davide 2005). The user will receive an update, if a certain event happens on the observed elements, also defined by a XPath expression. So the user is always informed about the current status of an entry or about new interesting information added to the database. This is not very interesting in our example, but when using XGR, e.g., for observing the progress in a workflow or following a product through its fabrication process, it becomes more relevant.

Integrating XGR and BabelPeers

The aim of this paper is to create a distributed XML database which uses semantic meta data. The combination of XGR and BabelPeers offers a solution to this problem. The first subsection describes how we combined the two applications on the DHT network and presents the advantages, we gain by this integration. The second subsection then shows how a new client uses the potential of the XGR-BabelPeers application.

Combining XGR and BabelPeers

Both applications are based on the Pastry DHT network. So it is pretty easy to combine them without causing great overhead. Every peer has to start a single Pastry (see section "DHT Networks") node and connect to the network over an already integrated bootstrap node. Then BabelPeers and XGR simply register themselves at this node. From then on, they can exchange messages with other XGR-BabelPeers applications. The message routing is handled by Pastry. The new client application can connect to both applications and can send queries in SPARQL for BabelPeers and XPath for XGR. Figure 8 shows the new combined network. In this example, there are three companies, that want to share their address books and perhaps several other XML documents for better collaboration. Each company has its own XML format for its address books and none wants to change its XML-Schema. XGR-BabelPeers combines all information with the help of a smart meta data structure without any changes in the XML documents.

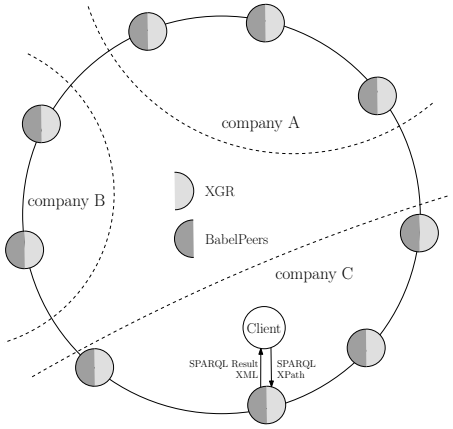

Figure 8: XGR and BabelPeers integration

First every company has to insert its meta data as RDF into the network. Therefore all companies have to agree on a generally accepted namespace for different concepts. In our introductory example (see Figure 1) the concepts "Company Address", "Private Address" and "Address" are given. With these concepts, the structure of the inserted address XML documents can be described by every company itself. Assume that company A saves all information about a firm inside one big XML document with root tag `firm`. Figure 9 shows an extract of the meta data company A adds to XGR-BabelPeers. It defines the semantics for company addresses, which can be found in the database at the path `/firm/fadr`. In addition, addresses naturally contain the city, zip, street, etc. and it should be possible to use the information for finding a particular firm. Again, no company needs to change its own XML format, it only has to define where it saves information about the concepts. Therefore

the inserted meta data describes where to find this additional concepts relative to the "Company Address" XPath.

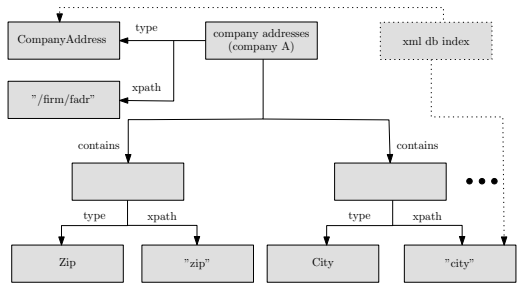

Figure 9: Extract of the address meta data (company A)

Now assume, Company B has a completely different XML structure for saving addresses. E.g. it has a lot of XML documents and every document contains just one address. By inserting their own XPaths to the semantic concepts, the information can be queried very easily for all participants without knowing the XML structure.

After designing the meta data and adding it into the BabelPeers part, the second step is to insert the XML documents into the XGR part. Therefore, every company has to map metadata and the datatypes with each other. The datatypes include, as described above, the information about the indexing structure, which will be distributed, and only indexed paths can be found efficiently. It would be an advantage if all companies defined the same indexing structure, and accordingly use indices which match with the usually performed queries. E.g. if we often look for addresses in a given city, it makes sense, if we have an index on the addresses or even a second index directly to the cities inside the addresses. If we describe these indices by defining RDF triples, which mark the desired concepts as an index, instead of XPath expressions, we can use the meta data to generate the datatype definitions even automatically (see dotted bordered concept in Figure 9). After this, we just insert all XML documents into the database and are ready to query all data in the same way.

Querying XGR-BabelPeers

In the next few paragraphs, we show how we design the new queries for hiding the two systems from the user.

We assume that normally the queries consist of a part which is looking for a XML fragment identified by an index and additionally one or several filters which select the desired data records. Figure 10 presents an example query which asks for all company addresses located in "Paderborn".

First we have to compose a SPARQL query to find the relevant concepts and appropriate XPath expressions. In detail, we search for a company address which contains information about the city. As result, we get a list of valid bindings for the variables `index` and `filter`.

For the example shown in Figure 9, the bindings would be as follows:

```
?indexPath = "/firm/faddr"
```

```
SELECT ?indexPath ?filter WHERE {
  ?addr type CompanyAddress .
  ?addr xpath ?indexPath .
  ?addr contains ?city .
  ?city type City .
  ?city xpath ?filter .
}
```

XPathPattern: `?indexPath[?filter="Paderborn"]`

Figure 10: Example – query

```
?filter = "city"
```

The second part of the query is an XPath expression, which describes how to use the variable bindings. We simply substitute the variables and get the following XPath expression:

```
/firm/faddr[city="Paderborn"]
```

Now the client uses this to query XGR and gets the desired information. For another company, the path can look completly different. Perhaps they have sorted their addresses by cities (the city tag is a parent of the address tag). The SPARQL query returns another result for the company XML structure and the new path expression probably looks like the following (the address tag is defined as an index):

```
/addresses/city/address/
  [../@name="Paderborn"]
```

The client poses this query to the XGR system, too, and merges the two results before returning one complete answer to the user.

Additionally, the combination of these two technologies also offers the possibility to query more general concepts, which can not be found directly in the XML structure. Figure 1 and Figure 2 introduced this. Imagine a database including different kinds of addresses with different datatypes: employee addresses, customer addresses, distributor addresses and so on. Now you want to inform everyone per eMail that something very important will happen. Without using meta data, you had to look at each datatype definition and create special queries for every kind of addresses. Now you can do this by defining a single query, because the meta data states that every kind of an address is a subclass of the concept "Address". BabelPeers uses this information to derive that every "Company Address" is an "Address", too. The query is shown in Figure 11. This query filters all addresses, which contain information about email addresses and returns this information.

Certainly, other query options like the publish and subscriber system can also be used after fetching meta data.

A different use case of XGR-BabelPeers is, if several companies share parts of their data, sometimes it could be hard to know what kinds of information are accessible. If the meta data information are up to date, a query for all concepts or even a graphical interface which presents the class hierarchy in a clear way, can help to solve this problem. Later, such a GUI can be extended, so that queries can be formulated by drag and drop or other simple interfaces.

```
SELECT ?indexPath ?rpath WHERE {
  ?addr type Address .
  ?addr xpath ?indexPath .
  ?addr contains ?email .
  ?email type EMail .
  ?email xpath ?rpath .
}
```

XPathPattern: `?indexPath/?rpath`

Figure 11: Example – general query

Conclusion

The goal of creating a system for managing XML documents with the help of semantic meta data is realized by the combination of BabelPeers and XGR. Both are working on a scalable P2P network and so the system is ready to save a high number of documents and the corresponding meta data.

We presented how we can use the advantages of XML and RDF to combine our local documents with the distributed knowledge base, available in the network. Every user is able to access the data without reading big XML DTDs or similar documents. Instead, he formulates the queries by using the meta data and gets direct access to all available documents containing relevant information.

The usage of the concepts presented in this paper creates several advantages:

- Easy joining of completely differently structured XML documents containing the similar information or parts of them.
- Querying the information of the combined database by posting simple semantically rich queries and not by writing difficult XPath expressions.
- Getting an overview over the saved information by analysing the meta data.

This is just a beginning to use the advantages of both applications, so there are some unsolved problems. It is possible to define which semantic concept should be used as an index for the datatype by using the meta data, but it is not clearly defined whether the XPath is given as an absolute path or a relative one. We can declare that all indices must be given as absolute and all other concepts as relative paths. Then we can build the query XPath expression by searching the matching index concept for a non index query in the RDF graph, but we think that there can be better solutions.

The XGR-BabelPeers system is not only usable for companies sharing addresses. We can imagine many scenarios, where it can help managing data. We only list a few of our ideas:

- In the introduction, we mentioned that Open Office documents are written in XML. Thus, sharing Open Office documents in a working community and tagging the documents with semantic meta data is possible.
- Combined with adequate access control mechanism it is possible to build a great semantic network between users, workgroups, or companies and sharing their knowledge.
- In connection with the Semantic Desktop idea the system provides the possibility to save arbitrary information and the related semantic meta data. With the help of an adequate API, this data can be accessed from different desktop applications.

At the moment, we implemented a prototype which combines both applications on the same pastry node. A very simple client has been created to test the integration but no evaluation has been done yet. This will be future work.

References

Arenas, M.; Kantere, V.; Kementsietsidis, A.; Kiringa, I.; Miller, R. J.; and Mylopoulos, J. 2003. The Hyperion Project: From Data Integration to Data Coordination. *SIGMOD Record* 32(3):53–58.

Battré, D.; Heine, F.; Höing, A.; and Kao, O. 2006. On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT based RDF stores. In *Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2006)*.

Cai, M.; Frank, M.; Pan, B.; and MacGregor, R. 2004. A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 2(2):109–130.

Emiliano Casalicchio, Federico Morabito, Giovanni Cortese, Fabrizio Davide. 2005. A novel adaptive content-based subscription management system. Technical report, DELIS – Dynamically Evolving, Large-Scale Information Systems.

Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.-J.; and Nielsen, H. F. 2003. Soap version 1.2. `http://www.w3.org/TR/soap12`.

Halevy, A. Y.; Ives, Z. G.; Suciu, D.; and Tatarinov, I. 2003. Schema Mediation in Peer Data Management Systems. In *19th International Conference on Data Engineering (ICDE)*.

Heine, F. 2006. Scalable P2P based RDF Querying. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, 17. New York, NY, USA: ACM Press.

Koubarakis, M.; Miliaraki, I.; Kaoudi, Z.; Magiridou, M.; and Papadakis-Pesaresi, A. 2006. Semantic Grid Resource Discovery using DHTs in Atlas. In *3rd GGF Semantic Grid Workshop*.

Rowstron, A., and Druschel, P. 2001. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science* 2218:329+.

Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.; Kaashoek, M. F.; Dabek, F.; and Balakrishnan, H. 2003. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *IEEE Transactions on Networking* 11.