Semantic Query Routing and Processing in P2P Database Systems: The ICS-FORTH SQPeer Middleware *

George Kokkinidis and Vassilis Christophides
Institute of Computer Science - FORTH
Vassilika Vouton, P.O. 1385, GR 71110
Heraklion, Greece
and
Department of Computer Science
University of Crete, GR 71409
Heraklion, Greece
{kokkinid, christop}@ics.forth.gr

ABSTRACT

Peer-to-peer (P2P) computing is currently attracting enormous attention. In P2P systems a very large number of autonomous computing nodes (the peers) pool together their resources and rely on each other for data and services. More and more P2P data management systems employ nowadays intensional (i.e., schema) information for integrating and querying peer bases in, so-called, Semantic Overlay Networks (SONs). Such information can be easily captured by emerging Semantic Web languages such as RDF/S. However, a fully-fledged framework for evaluating semantic queries over peer RDF/S bases (materialized or virtual) is missing. In this paper, we present the ICS-FORTH SQPeer middleware for routing and processing RQL queries and RVL views taking into account the data distribution (e.g., vertical, horizontal) involved in peer bases. To the best of our knowledge, SQPeer is the first middleware exploiting the full-power of RDF/S-based SONs for both hybrid and adhoc P2P database systems.

Categories and Subject Descriptors

H.2.4 [**Database Management Systems**]: Distributed Databases, Query Processing; C.2 [**Network Protocols**]: Routing protocols; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.4 [**Knowledge Representations Formalisms and Methods**]: Semantic Networks

*This work was partially supported by the EU projects SeLeNe (IST-2001-39045) and Delos (NoE-6038-507618). An earlier version of this work appears in the Proceedings of the International Workshop on Peer-to-peer Computing & DataBases, Heraklion, Crete, Greece, 2004.

HDMS '04 Athens, Greece

1. INTRODUCTION

Peer-to-peer (P2P) computing is currently attracting enormous attention, spurred by the popularity of file sharing systems such as Napster [23], Gnutella [13], Freenet [10], Morpheus [22] and Kazaa [18]. In P2P systems a very large number of autonomous computing nodes (the peers) pool together their resources and rely on each other for data and services. P2P computing introduces an interesting paradigm of decentralization going hand in hand with an increasing self-organization of highly autonomous peers. This new paradigm bears the potential to realize computing systems that scale to very large numbers of participating nodes while ensuring fault-tolerance.

However, current P2P systems offer very limited data management facilities. In most of the cases, searching information relies on simple selection on a predefined set of index attributes or IR-style string matching. These limitations are acceptable for file-sharing applications, but in order to support highly dynamic, ever-changing, autonomous social organizations (e.g., scientific or educational communities) we need richer facilities in exchanging, querying and integrating structured and semi-structured data. To build such net-centric information systems we essentially need to adapt the P2P computing paradigm to a distributed data and knowledge management setting. More precisely, we would like to support loosely coupled communities of databases where each peer base can join and leave the network at will.

The importance of intensional (i.e., schema) information for integrating and querying peer bases has been highlighted by a number of recent projects [3, 24, 14, 1]. In particular, the notion of Semantic Overlay Networks (SONs) [11, 28], appears to be an intuitive way to cluster together peers sharing the same schema information about a community domain or application model. Thus, peers employing one or more topics (or concepts) of a community schema, are semantically related and belong to the same SON. This approach facilitates query routing, since a peer can easily identify relevant peers instead of broadcasting (flooding) query requests on the network.

A natural candidate for representing such descriptive schemas (ranging from simple structured vocabularies to complex

reference models [6]) is the Resource Description Framework/Schema Language (RDF/S). RDF/S (a) enables a *modular design* of descriptive schemas based on the mechanism of *namespaces*; (b) allows easy *reuse* or *refinement* of existing schemas through *subsumption* of both class and property definitions; (c) supports partial descriptions since *properties* associated with a resource are by default *optional and repeated* and (d) permits *super-imposed descriptions* in the sense that a resource may be multiply classified under several classes from one or several schemas. These modelling primitives are crucial for P2P databases where monolithic RDF/S schemas and resource descriptions cannot be constructed in advance and peers may have only incomplete descriptions about the available resources. In this context, several declarative languages for querying and defining views over RDF/S description bases have been proposed in the literature such as RQL [17] and RVL [21]. However, a fully-fledged framework for evaluating semantic queries over peer RDF/S bases (materialized or virtual) is still missing.

In this paper, we present the ongoing SQPeer middleware for routing and processing semantic queries in P2P database systems. More precisely, we make the following contributions:

- In Section 2.1 we illustrate how conjunctive RQL queries expressed against a SON RDF/S schema can be represented in our middleware as *semantic query patterns*.
- In Section 2.2 we introduce a novel technique for advertise peer RDF/S bases using intentional information. In particular, we are employing *active-schemas* for declaring the parts of a SON RDF/S schema which are actually populated (or can be) in a peer base. Active-schemas are essentially RVL (materialized or virtual) views of peer bases.
- In Section 2.3 we sketch a semantic query routing algorithm which matches a given RQL query against a set of active-schemas in order to determine relevant peers. More precisely, this algorithm relies on query/view subsumption techniques to produce *semantic query patterns annotated with routing information*.
- In Section 2.4 we describe how SQPeer *query plans* are generated from annotated semantic query patterns taking into account the involved data distribution (e.g., vertical, horizontal). Then, a query plan is executed with the deployment of appropriate communication *channels* between the relevant peers.
- In Section 2.5 we discuss several *compile* or *run-time optimization opportunities* for SQPeer query plans.
- In Section 3 we present how the SQPeer semantic query routing and processing algorithms can be actually used in order to deploy both *hybrid* and *ad-hoc* P2P database systems.

Finally, Section 4 discusses related work and Section 5 summarizes our contributions and presents our future work.

2. THE SQPEER MIDDLEWARE

In order to design an effective query routing and processing middleware for peer RDF/S bases, we need to address the following issues:

1. How peer nodes formulate queries?
2. How peer nodes advertise their bases?
3. How peer nodes route a query?

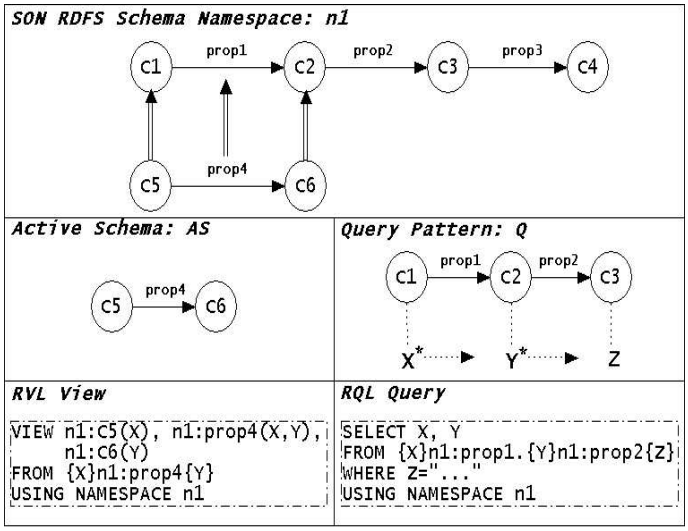

Figure 1: An RDF/S schema of a SON, an RVL peer active-schema and an RQL query pattern

4. How peer nodes process a query?
5. How distributed query plans are optimized?

In the following subsections, we will present the main design choices for SQPeer in response to the above fundamental issues.

2.1 RQL Queries of Peers

Each peer node in SONs provides RDF/S descriptions about information resources available in the network that conform to a number of community RDF/S schemas (e.g., for e-learning). Peer nodes employing the same schema to construct such descriptions can be considered to belong to the same SON. In the upper part of Figure 1 we can see an example of the schema defined in a specific namespace (i.e., `n1`) with four classes, `C1`, `C2`, `C3` and `C4`, that are connected with three properties, `prop1`, `prop2` and `prop3`. There are also two subclasses, `C5` and `C6`, of classes `C1` and `C2` respectively, which are related with the subproperty `prop4` of the property `prop1`.

Queries in SQPeer are formulated by client-peers in RQL, according to a community RDF/S schema as for example the schema identified by the namespace `n1`. A graphical end-user interface[1] may be used to assist RQL query formulation. For instance, in the bottom right part of Figure 1 we can see an RQL query **Q** returning all the resources binded by the variables `X` and `Y`. The path expressions in the from-clause imply a join on `Y` between the target resources of the property `prop1` and the origin resources of the property `prop2`. The where-clause filters, as usual, the binded resources according to the provided boolean conditions (e.g., on variable `Z`).

Since peer nodes in a SON can use one or more community RDF/S schemas either to actually populate their description bases or alternatively to define virtual views over their legacy (XML or relational) databases, we need logic support to reason on the intension of both query requests and peer base contents. To this end, we rely on the notion of *query patterns* for capturing the schema information employed by

[1]See for instance the RQL interactive demo at http://139.91.183.30:8999/RQLdemo/

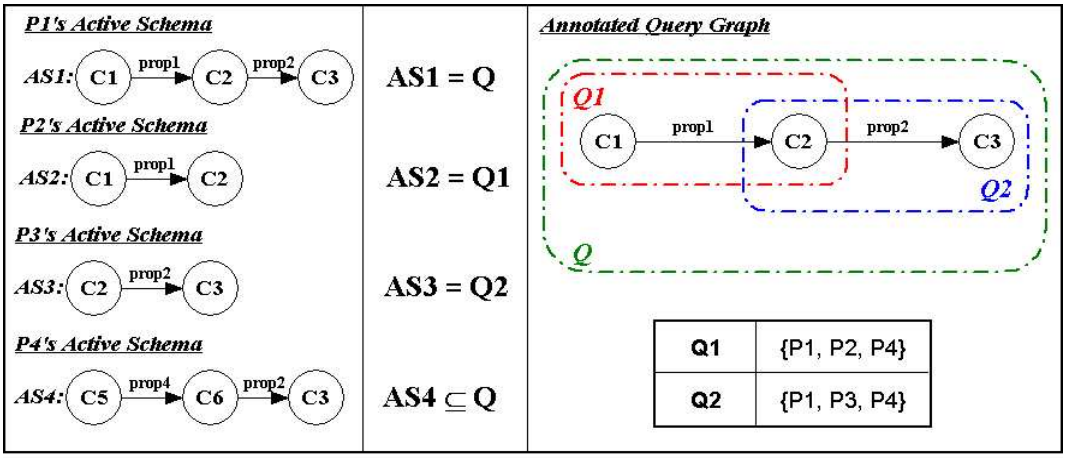

Figure 2: An annotated RQL query pattern

an RQL query. Query patterns are extracted from the path expressions appearing in the from clause of an RQL query in conjunction with the employed schema namespaces. The right middle part of Figure 1 illustrates the query pattern of query **Q**, where `X` and `Y` resource variables are marked with "*" to denote projections. Note that the end-point classes `C1`, `C2` and `C3` of properties `prop1` and `prop2` are obtained from their corresponding definitions in the namespace `n1`. In the rest of this paper, we are focusing on conjunctive query patterns formed only by RQL path expressions and projections (filtering conditions are ignored).

2.2 RVL Advertisements of Peer Bases

In the context of a SON, each peer node should be able to advertise its base to other peers. Peer base advertisement in SQPeer relies on materialized or virtual RDF/S schema(s). In the former case, a peer RDF/S base actually holds resource descriptions created according to the employed community schema(s), while in the latter, schema(s) can be populated on demand with data residing in a relational or an XML peer base. In both cases, community RDF/S schemas may contain numerous classes and properties not necessarily populated with data in a peer base. Therefore, we need a fine-grained definition of schema-based advertisements. We employ the notion of *active-schemas* to denote essentially the subset of a community RDF/S schema(s) for which all classes and properties are (in the materialized scenario) or can be (in the virtual scenario) populated in a peer base. The active-schema may be broadcasted to (or requested by) other peer nodes, thus informing the rest of the P2P system of what is actually available inside the peer bases.

The bottom left part of Figure 1 illustrates the RVL statement employed to advertise a peer base according to the community RDF/S schema identified by the namespace `n1`. This statement populates the classes `C5` and `C6` and the property `prop4` (in the view-clause) with appropriate instances from the peer's base (in the from-clause). In the middle left part of Figure 1 we can see the corresponding active-schema obtained by this RVL view. A more complex example is illustrated in the left part of Figure 2, comprising the active-schemas of four peers. Peer P1 contains resources related through the properties `prop1` and `prop2`, while peer P4 contains resources related through the properties `prop4` and `prop2`. Peer P2 contains resources related by `prop1`, while peer P3 contains resources related by `prop2`.

We can note the similarity in the intensional representation of peer base advertisements and query requests, respectively, as active-schemas and query patterns. This representation provides a uniform logical framework to route queries through the distributed peer bases, while also yielding significant performance gains. First, by representing in the same way what is queried by a peer and what is contained in a peer database, we can reuse the RQL query/RVL view (sound and complete) subsumption techniques, proposed in the Semantic Web Integration Middleware (SWIM [9]). Second, compared to global schema-based advertisements [24], we expect that the load of queries processed by each peer is smaller, since a peer receives only relevant to its base queries. This also affects the amount of network bandwidth consumed by the P2P system.

2.3 Semantic Query Routing

Query routing is responsible for finding the relevant to a query peers by taking into account data distribution (vertical, horizontal and mixed) of peer bases committing to a community RDF/S schema. The query-routing algorithm takes as input a query pattern and annotates each involved path pattern with the peers that can actually answer it, thus outputting an annotated query pattern. The query/view subsumption techniques of [9], are employed to determine which part of a query can be answered by an active-schema and rewrite accordingly the query sent to a peer. A pseudocode description on how this algorithm works is given in the next page.

Figure 2 illustrates an example of how SQPeer routing algorithm works given an RQL query **Q** composed of two path patterns, namely **Q1** and **Q2**, as well as the active-schemas of four peers. The middle part of the figure depicts how each query pattern matches one of the four active-schemas. P1's active-schema is equal to the path patterns **Q1** and **Q2**, so both path patterns are annotated with P1. P2's active-schema is equal to path pattern **Q1** and P3's active-schema is equal to **Q2**, so **Q1** and **Q2** are annotated with P2 and P3 respectively. Finally, P4's active-schema is subsumed by path patterns **Q1** and **Q2**, since `prop4` is a subproperty of `prop1`. Similarly to P1, **Q1** and **Q2** are annotated with P4. In the right part of the figure we can see the annotated query pattern returned by the SQPeer routing algorithm.

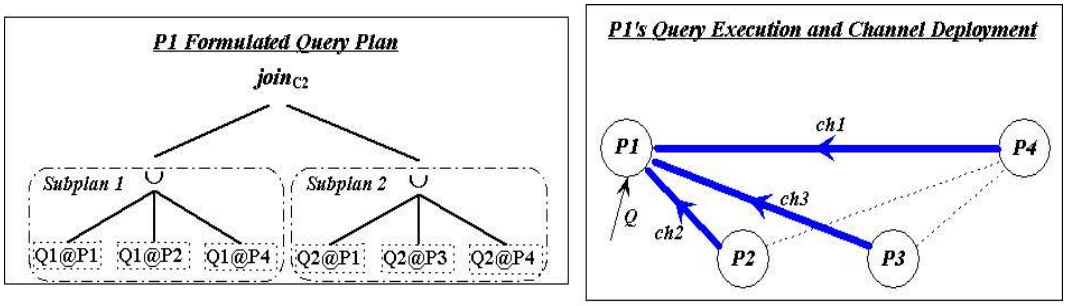

Figure 3: Query plan generation and channel deployment in SQPeer

Query-Routing Algorithm:

```
Input:  A query pattern AQ.
Output:  An annotated query pattern AQ'.

1. AQ':=Construct empty annotations for query
   pattern AQ
2. for all query path patterns AQ_i∈AQ, i=1 ... n do
    for all active schemas AS_j, j=1 ... m do
     for all active schemas path patterns
     AS_jk∈AS_j, k=1 ... l do
         if isSubsumed(AS_jk, AQ_i) then
            Annotate AQ'_i with peer P_j
         end if
     end for
    end for
   end for
3. return AQ'
```

2.4 Semantic Query Processing

Query processing in SQPeer is responsible for generating distributed query plans according to the information returned by the routing algorithm. In order to create the necessary foundation for executing distributed query (sub)plans, as well as for exchanging data between the involved peers, SQPeer relies on appropriate communication *channels* [26].

Through channels, peers are able to route (sub)plans and exchange the intermediary results according to the queries requested by client-peers. In addition, channels allows each peer to further route and process autonomously the received queries, by contacting peers independently of the previous routing operations. Finally, channel deployment can be adapted during query execution in order to response to network failures or peer nodes processing limitations. Each channel has a root and a destination node. The root node of a channel is responsible for the management of the channel using its local unique id. Data packets are sent through each channel from the destination to the root node. Beside query results, these packets can also contain "changing plan" and failure information or even statistics useful for query optimization. The channel construct and operations of ubQL [26] are employed to implement the above functionality in the SQPeer middleware.

The query-processing algorithm receives as input an annotated query pattern and outputs its corresponding query plan. A pseudocode description on how this algorithm works is also given below.

Figure 3 illustrates an example of how the RQL query **Q** shown in Figure 1 can be executed over the peer bases presented in Figure 2, given the already produced annotated query pattern. In this example, we assume that P1 runs

Query-Processing Algorithm:

```
Input:  An annotated query pattern AQ and current
        path pattern PP (initially the root).
Output:  A query plan QP corresponding to the
        annotated query pattern AQ.

1. QP:=∅
2. P:={P_1 ... P_n}, set of peers obtained by the
   annotation of PP in AQ
3. if P=∅
      QP:=PP@?
   else
      for all peers P_x∈P do
          QP:=QP⋃PP@P_x
          --Horizontal Distribution--
      end for
4. for all PP_i∈children(PP)
       TP_i:=Query-Processing-Algorithm(PP_i, AQ)
   end for
   QP:=⋈_Cp(QP, TP_1, ..., TP_n)
   --Vertical Distribution--
5. return QP
```

the query-processing algorithm in order to generate a corresponding query plan. The algorithm initially starts from the root of the annotated query pattern, i.e., the path pattern **Q1**, for which it runs the horizontal distribution algorithm in order to create `Subplan 1`, shown in Figure 3 (i.e., P1, P2 and P4 can effectively answer the subquery). The partial results obtained by these peers should be "unioned" (horizontal distribution). Next, the children of **Q1** are examined, i.e., **Q2**. The query-processing algorithm is executed with input the path pattern **Q2** and `Subplan 2` is formulated and returned (i.e., P1, P3 and P4 can effectively answer the subquery). Then, the returned query plan concerning **Q2** is "joined" (vertical distribution) with `Subplan 1`, thus producing the final plan shown in Figure 3. This is the final query plan produced by the algorithm, since the whole query **Q** has been processed. As we can easily observe from our example, taking into account the vertical distribution ensures correctness of query results (i.e., produce a valid answer), while considering horizontal distribution in query plans favours completeness (i.e., produce several valid answers).

In this context, a natural question that arises is how the SQPeer query processing algorithm behaves in cases where there is no sufficient routing information to produce a valid query answer (e.g., when there are no peers that can answer subquery **Q2**). In this case, SQPeer produces *partial query plans* with "holes" about relevant peers (denoted by

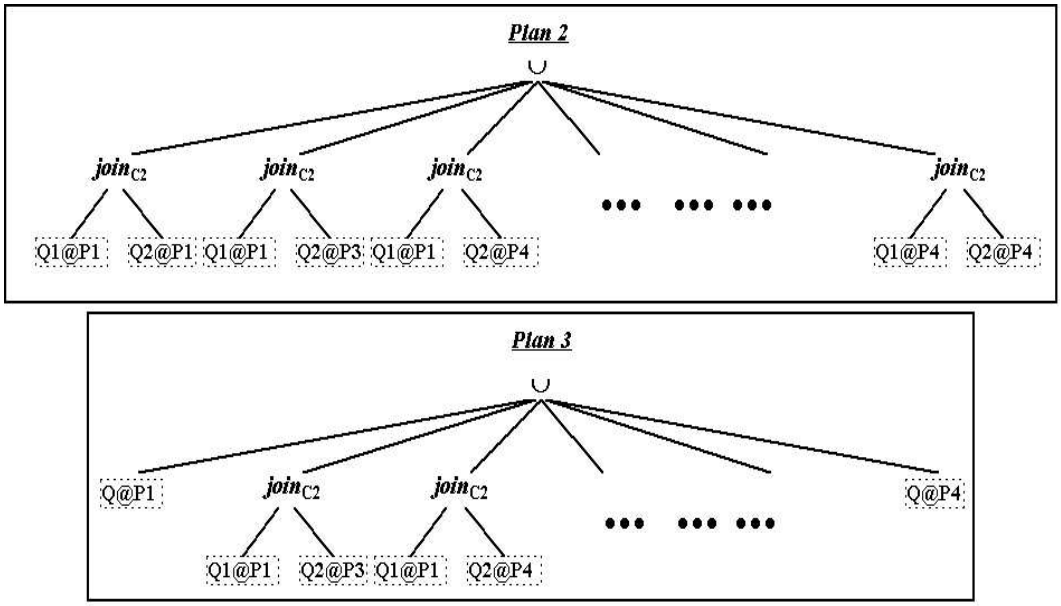

Figure 4: Optimizing query plans by applying algebraic equivalences and transformation rules

?) which need to be filled by other peers receiving the plan. This interleaved query routing and processing depends on the particular architectural setting of the P2P system and it will be detailed in Section 3.

It should be also stressed that SQPeer is capable to reformulate queries expressed against a SON RDF/S schema in terms of heterogeneous descriptive schemas employed by remote peers. This functionality is supported by SWIM [9] powerful mappings to RDF/S of both structured relational and semistructured XML peer bases.

Once a peer creates a query plan, it is responsible for its execution by deploying the necessary channels in the system (Figure 3). A channel is created having as root the peer launching the execution of the query and as destination one of the peers that need to be contacted each time according to the plan. Although each of these peers may contribute in the execution of the plan by answering to more than one subqueries, only one channel is of course created. This is one of the objectives of the optimization techniques presented in the sequel.

2.5 Query Optimization

In SQPeer we distinguish two possible optimization strategies of distributed query plans. First, compile-based optimization rely on algebraic equivalences (e.g., distribution of joins and unions) allowing us to push as much as possible query evaluation to the same peers, as well as on statistics held by each peer enabling us to choose between different execution policies for the query plans (e.g., data versus query shipping).

As we have seen in Figure 3, the query plan produced by the query processing algorithm contains unions only at the bottom of the plan tree. We can use the following algebraic equivalence to push unions on the top of our query plan, since pushing joins below the unions produces smaller intermediate results.

Distribution of joins and unions:
Given a subquery $\bowtie(\bigcup(Q_{11},\ldots,Q_{1n}),\bigcup(Q_{21},\ldots,Q_{2m}))$, *and rewriting it into* $\bigcup(\bowtie(Q_{11},Q_{21}),\bowtie(Q_{11},Q_{22}),\ldots,\bowtie(Q_{1n},Q_{2m}))$ *is beneficial, if the expected size of the join result is smaller than any of the inputs.*

According to the above algebraic equivalence, the query plan of Figure 3 is transformed using heuristics into the equivalent query `Plan 2` of Figure 4. This plan decreases the data transferring cost between the peers due to smaller intermediate results, as well as offers the ability to evaluate this plan in a pipeline way.

One can easily observe that query `Plan 2` does not take into account the fact that one peer (e.g., P4) can answer more than one successive path patterns. To this end, we are applying the two following transformation rules for identifying those subplans that can be answered by the same peer.

Transformation Rule 1:
Given a subquery $\bowtie(Q_1@P_i,\ldots,Q_n@P_i)$ *rewrite it into* $Q@P_i$, *where* $Q = Q_1 \bigcup \ldots \bigcup Q_n$.

Transformation Rule 2:
Given a subquery $\bowtie(\bowtie(QP,Q_1@P_i),Q_2@P_i)$ *rewrite it into* $\bowtie(QP,Q@P_i)$, *where* $Q = Q_1 \bigcup Q_2$.

By applying these two transformation rules on query `Plan 2`, query `Plan 3` will be produced reducing the number of subplans that need to be sent to the relevant peers (i.e., pushing the join on property `prop1` and `prop2` to peer P1 and P4).

Furthermore, statistics about the communication cost between peers (e.g., measured by the speed of their connection) can be used to decide between different channel deployments. Additionally, the expected size of peers' query results can be considered for the optimization choice. The processing load of the peers should also be taken into account, since a peer that processes fewer queries, even if its connection is slow, may offer a better execution time. This processing load can be handled by the existence of slots in each peer, which show the amount of queries that can be handled simultaneously. Having these statistics in hand, a peer node can decide at compile-time between *data, query or hybrid shipping* execution policies. In the example of Figure 5 we can see two alternatives on how P1 handles the

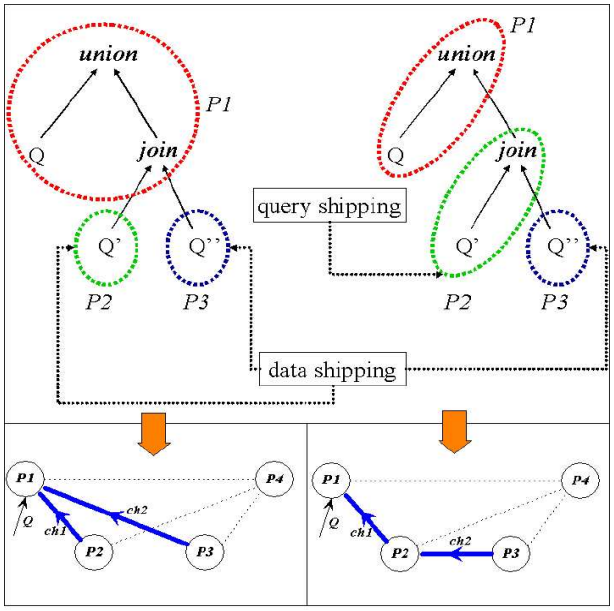

Figure 5: Data and Query Shipping Example

generated query plan. In the left part of the figure we can see the data shipping alternative, since P1 sends queries Q2 and Q3 to peers P2 and P3 and joins their results locally. In the right part of the query we can see the query shipping alternative, since P1 decides to forward the join operation down to P2, which in turn receives the results from P3 and executes the join locally before sending the full answer to P1 for further processing. At the bottom of the figure, we can see the deployment of the channels in SQPeer for each of these two alternative policies. In a scenario where the communication cost between peers P1 and P3 is greater than the cost between peers P2 and P3, query-shipping is preferable, since it exploits the fastest peer connection. In the case where peer P2 has a heavy processing load, data-shipping should be chosen, since P1 will process both the union and the join of the plan. Alternatively, if peer's P2 intermediate results of subquery **Q2** are large, query-shipping is the most beneficial.

On the other hand, run-time adaptability of query plans is an essential characteristic of query processing when peer bases join and leave the system at free will or more in general when system resources are exhausted. For example, the optimizer may alter a running query plan by observing the throughput of a certain channel. This throughput can be measured by the number of incoming or outgoing tuples (i.e., resources related through a property). Changing query plans may alter an already installed channel, as well as the query plans of the root and destination node of the channel. These changes include deciding at execution time between data or query shipping or discovering alternative peers for answering a certain subplan. The root node of each channel is responsible for identifying possible problems caused by environmental changes and for handling them accordingly. It should also inform all the involved nodes that are affected by the alteration of the plan. Since the alteration is done on a subplan and not on the whole query plan, only the nodes related to this subplan should be informed and possibly a few other nodes that contain partial results of the execution of the failed plan. Finally, the root node should create a new query plan by re-executing the routing and processing algorithm and not taking into consideration those peers that became obsolete.

We should keep in mind that switching to a different query plan in the middle of the query execution may cause some problems. Previous results, which were already created by the execution of the query to possible multiple peer nodes, have to be handled, since the new query plan will produce new results. Two are the possible solutions to this issue. The ubQL approach [26] proposes to discard previous intermediate results and all on-going computations are terminated. Alternatively [15] proposes a phased query execution, in which each time the query plan is changed, the system enters into a new phase. The final phase, which is called the cleanup phase, is responsible for combining the sub-results from the other phases in order to obtain a full answer. In SQPeer middleware, we have adopted the ubQL approach.

3. P2P ARCHITECTURES AND SQPEER

SQPeer can be used in different P2P architectural settings. Even though the P2P architecture affects the peers behavior, our proposed query processing and routing algorithms work independently of the particular architectural setting. Before going more on details regarding these issues, we explicitate the possible roles that peers may play in each case and their corresponding computing capabilities.

On the one hand, we have *client-peers*, which may frequently join or leave the system. These peers have only the ability to pose RQL queries to the rest of the P2P system. Since these peers usually have limited computing capabilities and they are connected to the system for short period of time, they do not participate in the query routing and processing in the way the other peers do.

On the other hand, we may have *simple-peers* that also act autonomously by joining or leaving the system, maybe not so frequently as client-peers. Their corresponding description bases can be shared by others during their connection to the P2P system. When they join the system, simple-peers can broadcast their active-schema information or alternatively request the active-schema of their known neighbors. Thus, a simple-peer identifies and connects physically with the SON(s) it belongs to and becomes known to its new neighborhood. Simple-peers have also the ability to pose queries as client-peers, but with the extra functionality of executing these queries against their own local bases.

Additionally, a small percentage of the peers may play the role of *super-peers*. A super-peer acts as a centralized server for a subset of simple-peers. Super-peers are mainly responsible for routing queries through the system and for managing a cluster of simple-peers which are responsible for. In this scenario, queries received and processed by a simple-peer are first send to its corresponding super-peer, which undertakes the routing process by replying to the simple-peer an annotated query pattern for further processing. Super-peers are usually highly-available nodes offering high computing capabilities.

In this context, we consider two architectural alternatives distinguished according to the distribution of knowledge on a P2P system regarding peer base advertisements. The first scenario corresponds to a *hybrid* P2P architecture based on the notion of Super-Peer Nodes (like Morpheus or Kazaa) while the second one is closer to an *ad-hoc* P2P architecture (like Freenet or Gnutella).

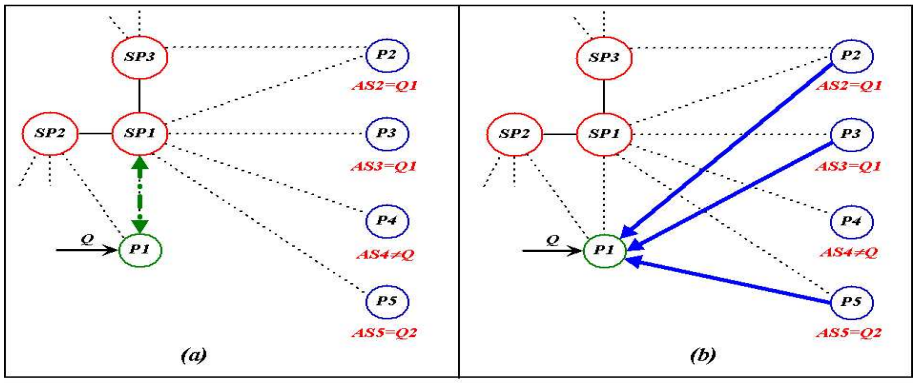

Figure 6: SQPeer query processing in a hybrid P2P system

3.1 Hybrid P2P SONs

In a hybrid P2P system [29] [24] each peer is connected with at least one super-peer, who is responsible for collecting the active-schemas (materialized or virtual) of all its simple-peers. The peers that provide RDF descriptions for the same community RDF/S schema are clustered under the same super-peer. Thus, each peer implicitly knows the active-schemas of all its semantic neighbors in the sense that they employ the same community RDF/S schema. When a peer connects to a super-peer, it forwards its corresponding active-schema (push). All super-peers are aware of each other, in order to be able to answer queries expressed in terms of different RDF/S schemas, while a simple-peer can be connected to multiple super-peers when it provides descriptions conforming to more than one schema. The topology of the super-peer network depends on the number of the super-peers, which in turn depends on the number of simple-peers and the processing capability of each super-peer. A multi-layered hierarchical organization of the super-peers network can be employed by using appropriate articulations (aka mappings) of the classes and properties defined in each super-peer RDF/S schema.

A client-peer can connect with a simple-peer and send a query request for further processing to the system. The simple-peer forwards the query to the appropriate super-peer according to the schema employed by the query. If this schema is unknown to the simple-peer, it sends the query randomly to one of its known super-peers, which will consecutively discover the appropriate super-peer through the super-peers backbone. In this scenario, we distinguish two sequential query evaluation phases: the first corresponds to query routing, which is done exclusively at super-peers, and the second to query processing and execution, which is performed by the simple-peers.

Figure 6, illustrates an example of how this scenario works. We consider a super-peer backbone containing three super-peers, SP1, SP2 and SP3, and a set of client peers, P1 to P5. All the simple-peers are connected with at least SP1, since they contain data conforming to the schema that SP1 is responsible for. When P1 receives a query **Q**, it initially contacts SP1, which is the super-peer responsible for the SON on which the query is addressed (Figure 6a). SP1 examines the active-schemas of all its simple-peers and creates an annotated query pattern containing the information that P2 and P3 can answer only the **Q1** path pattern, while P5 can answer the **Q2** path pattern. SP1 sends this annotated pattern to P1 in order to generate the appropriate query plan and create the two channels with P2 and P3 for gathering the results (Figure 6b). P2, P3 and P5 send their results back to P1, who joins them locally in order to produce the final answer. We should point out that since super-peers contain all the active-schema information of a SON, the annotated query pattern for a relevant query contains essentially the routing information for producing not only a correct but also a complete query plan, in the sense that it will not contain any holes and thus no further broadcasting and processing of the query is necessary.

3.2 Ad-hoc P2P SONs

Alternatively, we can consider an ad-hoc P2P architecture [7] [8]. In this alternative, when a peer first joins the system, it becomes aware only of its physically close neighbors. The peer should identify, by sending appropriate requests, at least one other related peer for each of its RDF/S community schemas. The related peers identified in this way, form the semantic neighborhood of the peer. In the next step, the peer explicitly requests the active-schemas of its neighbor peers (pull). Then, when a peer receives a relevant query, it applies locally the query routing algorithm and create a query plan. When the peer receives a query, whose schema is unknown or which cannot be answered by the semantic neighbors of the peer, it could request the active-schema information of a 2-depth, 3-depth, etc. neighborhood[2], until a relevant peer is found and thus constructing progressively *self-adaptive SONs*.

Unlike super-peers, in the ad-hoc scenario there are no real guarantees that a peer can actually generate a complete query plan. The query processing algorithm produces a query plan according to its local knowledge about relevant peers and therefore the query plan may contain "holes", i.e. subplans with incomplete peer information (denoted by **?**). In order to discover these peers and fill the holes, the query plan is forwarded to other peers which are known to be able to answer at least a part of the initial query plan. By establishing appropriate channels, the peers receiving a partial plan can in turn interleave query processing and routing using their local knowledge of the P2P system. The first peer that is able to fill all the "holes" and generate a complete query plan, holds also the responsibility of executing it and send the results back to the root peer through the already deployed channels.

Figure 7 depicts an example where peers P1 to P5 are

[2]More elaborated techniques based on DHT for RDF/S schemas can be used in this respect.

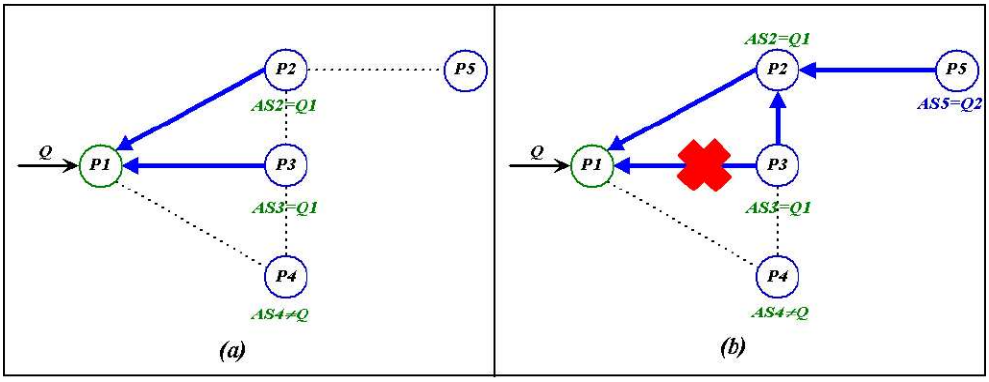

Figure 7: SQPeer query processing mechanism in an ad-hoc P2P system

connected in a self adaptive SON. P1 is aware of the active-schema of its neighbor peers, i.e., P2, P3 and P4. When P1 receives the query **Q**, it uses the already gathered active-schemas in order to apply locally the routing algorithm (Figure 7a). Since P2 and P3 can answer the **Q1** part of **Q** and since no known neighbor peer can answer the **Q2** part, P1 creates the query plan `Plan 1 =` $\bigcup(\bowtie(Q1@P2, Q2@?), \bowtie(Q1@P3, Q2@?))$. P1 establish two channels with those peers and forwards them the corresponding partial plans. Since P3 is not connected with any new peer, the channel between P1 and P3 fails. On the other hand, when P2 receives the query plan from P1, executes the query routing algorithm and identifies that P5 can answer the Q2 part of the query (Figure 7b). P2 creates the query plan `Plan 2 =` $\bigcup(\bowtie(Q1@P2, Q2@P5), \bowtie(Q1@P3, Q2@P5))$ and executes the plan by deploying the necessary channels between the involved peers. The appropriate subplans are sent to peers P3 and P5 and the results are returned to P2, where there are "joined" and "unioned" locally before they are sent back to P1 as a complete answer.

These two architectural alternatives exhibit different behavior in the routing, processing and execution of a query. In the ad-hoc architecture, SONs are created in a self-adaptive way, while in the super-peer architecture SONs are created in a more static way, since each super-peer is responsible for the creation and further management of SONs. The existence of SONs leads to minimizing the broadcasting (flooding) in the P2P system, since a query is received and processed only by the relevant peers in both architectures. It should be stressed that while in the ad-hoc architecture, peers handle both the query routing and processing load, super-peers are only responsible for routing and simple-peers for processing of the queries. Additionally, super-peers contain a global knowledge of the active-schemas of the peers in a SON and therefore can create a query plan offering completeness in the results. In the ad-hoc alternative, peers are aware only of a small number of active-schemas in the SON, and thus they can't guarantee completeness of query plans. Finally, super-peers may handle the role of a mediator in a scenario where a query expressed in terms of a global-known schema needs to be reformulated in terms of the schemas employed by the local bases of the simple-peers by using appropriate mapping rules.

4. RELATED WORK

Several projects address query processing issues in P2P database systems. Query Flow [19] is a system offering dynamic and distributed query processing using the notion of HyperQueries. HyperQueries are essentially subplans that exist in peer nodes and guide the processing of a query through the network. Furthermore, ubQL [26] provides a suite of process manipulation primitives that can be added on top of any traditional query language to support distributed query optimization. UbQL distinguishes the deployment from the execution phase of a query and supports adaptability of query plans during the execution phase. Compared to these projects, SQPeer does not require an a priori knowledge of the relevant to a query peers.

Mutant Query Plans (MQPs) [25] are logical query plans, where leaf nodes may consist of URN/URL references, or of materialized XML data. The references to resource locations (URLs) refer to peers where the actual data reside, while the abstract resource names (URNs) can be seen as the thematic topics of the requested data in a SON. MQPs are themselves serialized as XML elements and are exchanged among the peers. When a peer N receives a MQP M, N can resolve URN references, materialize URL references, evaluate or re-optimize MQP subplans, or just route M to another peer. When a MQP is fully evaluated, i.e. is reduced to XML code only, the result is returned to the *target* peer, which has initiated the query. The efficient routing of MQPs is preserved by information derived from multi-hierarchic topic namespaces (e.g., for educational material on computer science and for geographical information) organized by assigning different roles to certain peers. This approach is similar to a super-peer architecture, with the difference of distributing the super-peer role (i.e., the routing of the query) to more than one peers. Unlike SQPeer, this approach reduces the optimization opportunities of MQP by simply migrating possibly big XML fragments of query plans along with partial results of subqueries. In addition, it is not clear how subtopics can be exploited by query routing.

AmbientDB [7] addresses P2P data management issues in an digital environment, i.e. audio players exchanging music collections. AmbientDB assumes the existence of a common global schema, although client-peers may contain their own schemas (mappings are used in this case). AmbientDB relies on the relational data model and algebra. The query processing mechanism is based on a three-level translation of an "abstract global algebra" into multi-wave stream processing plans, distributed over an ad-hoc and self-organizing P2P network. Initially, a query is translated into standard relational operators for selection, join, aggregation and sort over "abstract table types". Then, this abstract query plan

becomes concrete by instantiating the abstract table types with concrete ones, i.e., the local or distributed tables that exist in the peer bases. Finally at the execution level, the concrete query plan is executed by selecting between different query execution strategies. AmbientDB P2P protocol is responsible for the query routing and relies on temporary (logical) routing trees, which are created on-the-fly as subgraphs of the Chord network. Chord can also be used to implement clustered indices of distributed tables in AmbientDB as DHTs. Each AmbientDB peer contains the index table partition that corresponds to it after hashing the key-values of all tuples in the distributed table. The user decides for the use of such DHTs, thus accelerating relevant lookup queries. Compared to AmbientDB, SQPeer provides a richer data framework, as well as exhibits a run-time adaptability of generated query plans.

Other projects address mainly query routing issues in SONs. In [12] indices are used to identify peers that can handle containment queries (e.g., in XML). For each keyword in the query, a peer node searches its indices and returns a set of nodes that can answer it. According to the operators used to connect these keywords, the peer node decides whether to union or intersect the sets of relevant peers. In this approach, queries are directly sent to the set of peers returned by the routing algorithm with no further details on how a set of semantically related peers can actually execute a complex query involving vertical and horizontal distribution.

RDFPeers [8] is a scalable distributed RDF/S repository based on an ad-hoc P2P architecture. RDF-triple-storing nodes are used to store RDF/S triples at three peers in the network according to the subject, the predicate or the object value. An extension to Chord, called Multi-Attribute Addressable Network (MAAN), is used to store these triples and to route disjunctive, range and conjunctive multi- predicate queries to the appropriate peers. This approach ignores RDF/S schema information during query routing, while distributed query processing and execution policies are not addressed.

In [27], a super-peer like P2P architecture is introduced, which relies on the extent of an existing RDF/S store. Authors propose an index structure for all the property paths that can be specified given an RDF/S schema. The paths in the index are organized hierarchically according to their length (simple properties appear as leaves of the tree). For each path in the tree, the index maintains information about the peers that can answer it, as well as the size of path instantiations. A query answering algorithm that determines all possible combinations of the subpaths of the given query path and determines the sources to answer it is described. The proposed index structure, which is considered to be controlled by a mediator, is difficult to be updated and handled in a situation where peers frequently enter and leave the system. The routing information concerning different paths is held in a centralized way, in contrast to SQPeer, where this knowledge is distributed and obtained throughout the routing phase. Although schema information is used for indexing, RDF/S subsumption is not considered. Finally, optimization (based on a cost model) is focused only on join re-orderings, which is a subset of the optimizations considered in SQPeer.

The Edutella project [24] explores the design and implementation of a schema-based P2P infrastructure for the Semantic Web. In Edutella, peer content is described by different and extensible RDF/S schemas. Super-peers are responsible for message routing and integration/mediation of peer bases. The routing mechanism is based on appropriate indices to route a query initially within the super-peer backbone and then between super-peers and their respective peers. A query processing mechanism in such a schema-based P2P system is presented in [5]. Query evaluation plans (QEPs) containing selection predicates, compression functions, joins, etc., are pushed from clients to simple or super-peers where they are executed. Super-peers dispose an optimizer for generating partial query plans determining the parts of the query to be sent to the next (super-)peers and the operators to be locally executed for combining the results. The proposed query processing facility does not take into account the possible existence of subsumption relationships of RDF/S classes and properties. Additionally, this approach does not consider run-time adaptability of query plans.

Finally, although the use of indices and super-peer topologies facilitate query routing, the cost of maintaining (XML or RDF) indices of entire peer bases is important compared to the cost of maintaining peer active-schemas (i.e., views), as in the case of SQPeer. Last but not least, SQPeer can be used to deploy both hybid and ad-hoc P2P systems.

5. SUMMARY AND FUTURE WORK

In this paper, we have presented the ICS-FORTH SQPeer middleware offering sophisticated query routing and processing middleware in P2P data management systems. We presented how (conjunctive) RQL path queries expressed against a SON RDF/S schema can be represented as semantic query patterns and how peers can advertise their bases using active-schemas expressed in the same formalism. We sketched a semantic query routing algorithm, which relies on query/view subsumption techniques to annotate semantic query patterns with information concerning relevant peers. We also presented how SQPeer query plans are created and executed by taking into account the data distribution in peer bases. Finally, we have discussed several compile and run-time optimization opportunities for SQPeer query plans, as well as possible architectural alternatives for static or self-adaptive RDF/S-based SONs.

Several issues remain open with respect to the optimization of distributed and adaptive queries in SQPeer borrowing ideas from related works [2] [4] [16]. We plan to study the trade-off between result completeness and processing load using the concepts of Top N (or Bottom N) queries [20]. In the same direction, we can use constraints regarding the number of peer nodes that each query is broadcasted and further processed. Finally, we want to investigate the possible use of Distributed Hash Tables [28] for RDF/S schemas with subsumption information, used in the query routing process.

Acknowledgments

We would like to thank Val Tannen for fruitful discussions on peer channels.

6. REFERENCES

[1] Aberer, K., Cudre-Mauroux, P., Hauswirth, M.: The Chatty Web: Emergent semantics through gossiping.

In Proceedings of the 12th International World Wide Web Conference (WWW), Budapest, Hungary, 2003.

[2] Avnur, R., Hellerstein, J.M.: Eddies:Continuously Adaptive Query Processing. ACM SIGMOD, p.261-272, Dallas, TX, May 2000.

[3] Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In Proceedings of the 5th International Workshop on the Web and Databases (WebDB), Madison, Wisconsin, 2002.

[4] Braumandl, R., Keidl, M., Kemper, A., Kossmann, D., Kreutz, A., Seltzsam, S., Stocker, K.: ObjectGlobe: Ubiquitous query processing on the Internet. In VLDB Journal 10, pp.48-71, 2001.

[5] Brunkhorst, I., Dhraief, H., Kemper, A., Nejdl, W., Wiesner, C.: Distributed Queries and Query Optimization in Schema-Based P2P-Systems. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), Berlin, Germany, 2003.

[6] Maganaraki, A., Alexaki, S., Christophides, V., Plexousakis, D.: Benchmarking RDF Schemas for the Semantic Web. In Proceedings of the 1st International Semantic Web Conference (ISWC'02), 2002.

[7] Boncz, P., Treijtel, C.: AmbientDB: relational query processing in a P2P network. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), LNCS 2788, Springer Verlag, 2003.

[8] Cai, M., Frank, M.: RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In 13th International World Wide Web Conference (WWW), New York, 2004.

[9] Christophides, V., Karvounarakis, G., Koffina, I., Kokkinidis, G., Magkanaraki, A., Plexousakis, D., Serfiotis, G., Tannen, V.,: The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware. In Proceedings of the First International Workshop on Semantic Web and Databases (SWDB), Co-located with VLDB 2003, Humboldt-Universitat, Berlin, Germany, 2003.

[10] Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, volume 2009 of LNCS, Springer-Verlag, 2001.

[11] Crespo, A., Garcia-Molina H.: Semantic Overlay Networks for P2P Systems. Stanford Technical Report, 2003.

[12] Galanis, L., Wang, Y., Jeffery, S.R., DeWitt, D.J.: Processing Queries in a Large P2P System. In Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE), 2003.

[13] The Gnutella file-sharing protocol. Available at : http://gnutella.wego.com

[14] Halevy, A. Y., Ives, Z. G., Mork, P., Tatarinov, I.: Piazza: Data Management Infrastructure for Semantic Web Applications. In Proceedings of the 12th International World Wide Web Conference (WWW), 2003.

[15] Ives, Z. G.: Efficient Query Processing for Data Integration. phD Thesis, University of Washington, 2002.

[16] Ives, Z. G., Levy, A. Y., Weld, D. S., Florescu, D., Friedman, M.: Adaptive Query Processing for Internet Applications. IEEE Data Engineering Bulletin, Vol.23, No.2, pp.19-26, 2000.

[17] Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In Proceedings of the 11th International World Wide Web Conference (WWW), Honolulu, Hawaii, USA, 2002.

[18] The Kazaa file-sharing system. Available at : http://www.kazaa.com

[19] Kemper, A., Wiesner, C.: HyperQueries: Dynamic Distributed Query Processing on the Internet. In Proceedings of the International Conference on Very Large Data Bases (VLDB), Rome, Italy, 2001.

[20] Kossmann, D.: The State of the Art in Distributed Query Processing. ACM Computer Surveys, Vol.32, No.4, pp.422-469, 2000.

[21] Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the Semantic Web Through RVL Lenses. In Proceedings of the 2nd International Semantic Web Conference (ISWC), 2003.

[22] The Morpheus file-sharing system. Available at: http://www.musiccity.com

[23] The Napster file-sharing system. Available at : http://www.napster.com

[24] Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Loser, A.: Super-Peer-Based Routing and Clustering Strategies for RDF-Based P2P Networks. In Proceedings of the 12th International World Wide Web Conference (WWW), Budapest, Hungary 2003.

[25] Papadimos, V., Maier, D., Tufte, K.: Distributed Query Processing and Catalogs for P2P Systems. In Proceedings of the 2003 CIDR Conference, 2003.

[26] Sahuguet, A.: ubQL: A Distributed Query Language to Program Distributed Query Systems. phD Thesis, University of Pennsylvania, 2002.

[27] Stuckenschmidt, H., Vdovjak, R., Houben, G., Broekstra, J.: Index Structures and Algorithms for Querying Distributed RDF Repositories. In Proceedings of the International World Wide Web Conference (WWW), New York, USA, 2004.

[28] Triantafillou, P., Pitoura, T.: Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In Proceedings of the Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P), Collocated with VLDB '03, 2003.

[29] Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. In Proceedings of the 19th International Conference Data Engineering (ICDE), IEEE Computer Society Press, Los Alamitos, CA, 2003.