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, PO Box 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 rely nowadays on intensional (i.e. schema) information for integrating and querying peer bases. 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. The novelty of SQPeer lies on the use of intensional active schemas for determining relevant peer bases, as well as, constructing distributed query plans. In this context, we consider optimization opportunities for SQPeer query plans.

1 Introduction

Peer-to-peer (P2P) computing is currently attracting enormous attention, spurred by the popularity of file sharing systems such as Napster [19], Gnutella [10], Freenet [7], Morpheus [18] and Kazaa [14]. 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

* This work was partially supported by the EU project SeLeNe (IST-2001-39045).

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. In particular, 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 [20] [11]. In particular, the notion of Semantic Overlay Networks (SONs) [8], appears to be an intuitive way to group together peers sharing the same schema information. Thus, peers employing one or more topics (or concepts) of the same thematic hierarchy, 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 topic hierarchies (or more complex domain models) 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 users may have only incomplete descriptions about the available resources. In this context, several declarative languages for querying and defining views over RDF description bases have been proposed in the literature such as RQL [13] and RVL [17]. 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 advertising 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 (or can be) populated 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, we show how 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 the generated SQPeer query plans. Finally, Section 3 summarizes our contributions and presents our future work.

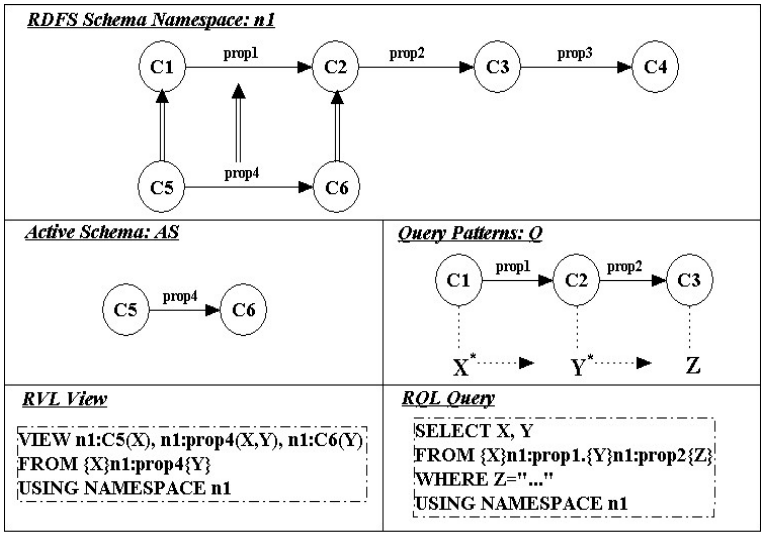


Fig. 1. RDF/S schema namespace, peer active-schema and query pattern graph

2 Semantic Query Routing and Processing in SQPeer

In order to design an effective query routing and processing middleware for peer RDF/S bases, we need to address issues, such as how peer nodes *formulate* queries and *advertise* their bases over a SON, as well as how they *route* and *process* queries and how the resulting 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 Peer Queries

Each peer node in SQPeer provides RDF descriptions that conform to a number of RDF schemas. Peer nodes with the same schema can be considered to belong to the same SON[1]. In the upper part of Figure 1 we can see an example of the schema graph of 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 sub-property `prop4` of the property `prop1`.

Queries in SQPeer are formulated by client-peers in RQL, according to the RDF schemas they use to create their description bases or to define virtual views over their legacy (XML or relational) databases. In this context, we need to reason about query/view containment in order to guide query routing through the peer bases of the system. To this end, we introduce the notion of ***query patterns*** capturing the schema information

[1] In a more sophisticated scenario, peer nodes contributing RDF descriptions specified by an RVL view are members of the same SON

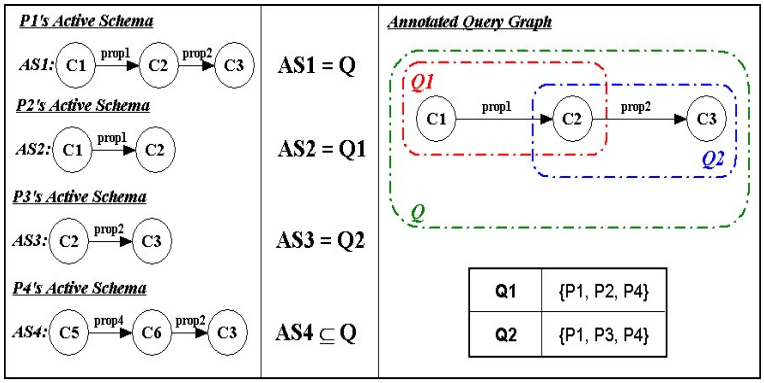


Fig. 2. An annotated RQL query graph

employed by an RQL query. This information is mainly extracted from the path expressions appearing in the from clause.

In the bottom right part of Figure 1 we can see an RQL query returning all the resources binded by the variables `X` and `Y`. In the from-clause, the employed path expressions imply a join on the `Y` resource variable between the target of the property `prop1` and the origin of the property `prop2`. The where-clause filters the returned resources according to the value of variable `Z`. Filtering conditions are not taken into account by RQL query patterns. The right middle part of Figure 1 illustrates the query pattern graph of query Q, where `X` and `Y` resource variables are marked with "*" to denote projections. A graphical end-user interface[2] may be used to create and visualize such query pattern graphs. Since the query pattern graph involves only path expressions and projections, we focus only on conjunctive RQL queries.

2.2 Peer Base Advertisement

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 virtual or materialized RDF schema(s). Since these schemas contain numerous RDF classes and properties not necessarily populated with data in a peer base, we need a fine-grained notion of schema-based advertisements. The ***active-schema*** of a peer node is essentially a subset of the employed RDF schema(s) for which all RDF classes and properties are (in the materialized scenario) or can be (in the virtual view scenario) populated. 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 of a peer active-schema. 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 the figure we can see the corresponding active-schema graph obtained by this view. This view can be a materialized RDF/S schema with actual resource

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

descriptions or can be a virtual one populated with data from a relational or an XML peer base. 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 representation of active-schemas and query pattern graphs.

Representing active-schemas and query pattern graphs in an intensional way makes easier to maintain a distributed knowledge of the P2P system, while 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 subsumption techniques, as proposed in the Semantic Web Integration Middleware (SWIM [6]). Second, compared to global schema-based advertisements [20], we expect that the load of queries processed by each peer is smaller, since a peer receives only relevant to its content 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 SON RDF/S schema. The query/view subsumption techniques of [6], 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.

The query-routing algorithm takes as input a query graph and annotates each involved path pattern with the peers that can actually answer it, thus outputting an annotated query graph. A pseudocode description on how this algorithm works is given below.

Query-Routing Algorithm:

```
1. A peer P receives an RQL query Q.
2. Peer P parses the query Q and creates the corresponding query
   pattern graph by obtaining the involved paths.
3. For each pattern, the matching algorithm is performed.
  (a) Compare the path pattern with all known active-schemas.
  (b) If the active-schema graph is subsumed by the selected path
      pattern, then it is annotated with the name of the peer
      owning the active-schema.
4. Output annotated query graph.
```

An example of a query graph, which is composed of two path patterns, Q1 and Q2, is illustrated in Figure 2. In the middle part of Figure 2 we can also see how the matching is performed with the active-schemas of our example peers. 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 sub-property of `prop1`. Similarly to P1, Q1 and Q2 are annotated with P4. In the right part of Figure 2 we can see the annotated graph created by this matching.

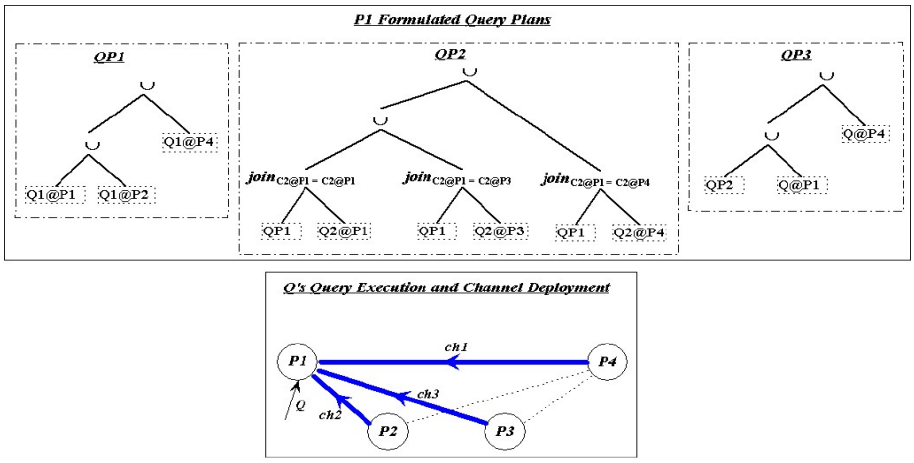


Fig. 3. A SQPeer query execution example

2.4 Semantic Query Processing

Query processing in SQPeer takes the responsibility of generating distributed query plans according to the information returned by the routing algorithm. These query plans are then executed by the relevant peers. Communication ***channels*** [21] enable this distributed execution and create the necessary foundation for exchanging appropriate data between the appropriate peers.

Through channels, peers are able to route query plans and exchange the corresponding results according to the queries requested by client-peers. In addition, channels permit each peer to further route and process the queries received, since it can be connected with more 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 and for creating a locally unique id for it. 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 other statistics useful for run-time query adaptability.

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

Query-Processing Algorithm:

```
1. Peer P receives an annotated query graph AQ for the RQL query Q.
2. If P doesn't answer any part of the query Q
      Send query to a neighbor peer (according to the
      architectural alternative);
   Else
      Create a new query plan QP;
```

```
3. Starting from a root of the graph, check every query path
   pattern PP in a sequential order, until AQ is fully processed;
4. If PP the root
      Execute horizontal data distribution algorithm with input
      (QP, PP, AQ);
   Else
      Execute vertical data distribution algorithm with input
      (QP, PP, AQ);
      Execute horizontal data distribution algorithm with input
      (QP, PP, EQ ⋃ AQ), where EQ the previously processed query;
5. Output formulated query plan QP.
```

Vertical data distribution algorithm with input (QP, PP, AQ):

```
1. Obtain set of peers, e.g. P'={P1,...,Pn}, from the annotated
   query graph that can answer PP.
2. For each peer of the set, e.g. Px, create query plan
   QPx = QP join_{Cp@P=Cq@Px} AQ@Px, where Cp and Cq are the classes
   on which the join of the two queries is executed.
3. Create query plan QP = QP1 ⋃ QP2 ⋃ ... ⋃ QPn
```

Horizontal data distribution algorithm with input (QP, PP, AQ):

```
1. Obtain set of peers, e.g. P"={P1,...,Pn}, from the annotated
   query graph that can answer PP.
2. For each peer of the set, e.g. Px, expand query plan as
   QP = QP ⋃ AQ@Px
```

Figure 3 illustrates an example of how the RQL query Q shown in Figure 1, can be executed over the P2P database system described in Section 2.2. The query is first sent to peer P1 which initially executes the query-routing algorithm in order to obtain the annotated query graph of Figure 2. P1 runs the query-processing algorithm and since it can answer a part of the query, creates a new query plan. The algorithm selects as a root of the annotated query graph, the path pattern Q1, for which it runs the horizontal distribution algorithm. This algorithm outputs query plan QP1, shown in Figure 3, since P1, P2 and P4 can execute query path pattern Q1. Next, path pattern Q2 is selected and the vertical data distribution algorithm executes and returns the query plan QP2. For each of the peers that can process Q2, we join the sub-queries sent with the results obtained from query plan QP1. P1 additionally follows the horizontal data distribution for obtaining more complete results. Since P1 and P4 can answer the whole query Q, P1's query plan will evolve to QP3: QP2 ⋃ Q@P1 ⋃ Q@P4. The final query plan and the deployment of the channels in SQPeer can also be seen in Figure 3. As seen from the example, taking into account the vertical distribution ensures correctness of query results, while considering horizontal distribution in query plans favours completeness.

Possible rewritings of the queries sent to remote peers in terms of different descriptive schemas may be necessary. This functionality can be implemented in SWIM [6], which supports powerful mappings to RDF/S of both structured relational and semistructured XML databases.

After the creation of the query plan, the peer holds the responsibility of executing this plan and deploy the necessary channels in the system. From the query plan the peer obtains the set of peers that need to be conducted for executing the query. For each of

these peers, a channel is created with the root being the peer executing the algorithm and the destination being the peer examined. Although each of these peers may contribute in the execution of the plan by replying to more than one sub-queries, only one channel is of course necessary.

2.5 Query Optimization

In SQPeer we distinguish two possible optimizations of distributed query plans. First, compile-based optimization depends on statistics held by each peer and allows to choose between different execution policies, i.e. data or query shipping [16]. These statistics involve response times from previously contacted peers or result sizes from previous executions of the same query. The speed of the connection between the peers can be used to decide between different channel deployments. The processing load of the peers can be also considered, 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.

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). 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. 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.

3 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.

SQPeer's query processing and routing algorithms can be used independently of the particular P2P architectural setting. We consider two such architectural alternatives. The first alternative corresponds to an ad-hoc P2P architecture (like Freenet or Gnutella), where SONs are formed in a self-adaptive way. More precisely, when a peer initially

joins the system, it identifies and connects to other peers committing to the same RDF/S schema of the SON. The second alternative is closer to a hybrid P2P architecture based on the notion of Super-Peer Nodes (like Morpheus or Kazaa), where SONs are created in a more static way. In particular, each super-peer node is responsible for the creation and further management of a SON. 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 their sub-peers for processing the queries. Additionally, super-peers contain a global knowledge of the active-schemas of the peers in a SON and therefore they can create a query plan offering completeness in the results. Finally, super-peers may possibly play 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 peer bases by using appropriate mapping rules.

Recent projects address query processing issues in P2P database systems, such as Query Flow [15], ubQL [21], Mutant Query Plans (MQPs) [22] and AmbientDB [4]. Compared to these projects, SQPeer does not require an a priori knowledge of the relevant to a query peers and exhibits more optimization opportunities for distributed query plans. Other projects address mainly query routing issues in SONs, such as the Edutella project [20] [3], RDFPeers [5] and [9]. Unlike these projects, SQPeer's query routing and processing algorithms considers both vertical and horizontal distribution, while exploiting the full power of RDF/S-based SONs and in particular the subsumption of classes or properties. Finally, although the use of indices and super-peer topologies facilitate query routing, the cost of updating (XML or RDF) distributed indices when peers join and leave is important compared to the cost of maintaining active-schemas (i.e. views), as in the case of SQPeer.

Several issues remain open with respect to the optimization of distributed and adaptive queries in SQPeer borrowing ideas from related works [1] [2] [12]. We plan to study the trade-off between result completeness and processing load using the concepts of Top N (or Bottom N) queries [16]. In the same direction, we can use constraints regarding the number of peer nodes that each query is broadcasted and further processed.

Acknowledgment

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

References

1. Avnur, R., Hellerstein, J.M.: Eddies:Continuously Adaptive Query Processing. ACM SIGMOD, p.261-272, Dallas, TX, May 2000.
2. 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.
3. 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.

4. 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.
5. 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.
6. 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.
7. 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.
8. Crespo, A., Garcia-Molina H.: Semantic Overlay Networks for P2P Systems. Stanford Technical Report, 2003.
9. 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.
10. The Gnutella file-sharing protocol. Available at : http://gnutella.wego.com
11. 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.
12. 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.
13. 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.
14. The Kazaa file-sharing system. Available at : http://www.kazaa.com
15. 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.
16. Kossmann, D.: The State of the Art in Distributed Query Processing. ACM Computer Surveys, Vol.32, No.4, pp.422-469, 2000.
17. 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.
18. The Morpheus file-sharing system. Available at: http://www.musiccity.com
19. The Napster file-sharing system. Available at : http://www.napster.com
20. 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.
21. Sahuguet, A.: ubQL: A Distributed Query Language to Program Distributed Query Systems. phD Thesis, University of Pennsylvania, 2002.
22. Papadimos, V., Maier, D., Tufte, K.: Distributed Query Processing and Catalogs for P2P Systems. In Proceedings of the 2003 CIDR Conference, 2003.