PUBLISHING, DISCOVERING AND UPDATING SEMANTIC GRID RESOURCES USING DHTS*

Zoi Kaoudi, Manolis Koubarakis, Kostis Kyzirakos, Matoula Magiridou, Iris Miliaraki†, and Antonios Papadakis-Pesaresi
Dept. of Informatics and Telecommunications
National and Kapodistrian University of Athens
Athens
Greece
{zoi, koubarak, kkyzir, matoula, iris, apapadak}@di.uoa.gr

Abstract We present *Atlas*, a P2P system for publishing, discovering and updating Semantic Grid resources described using the RDF data model. Atlas has been developed in the context of project OntoGrid funded by FP6. Atlas is built on top of the distributed hash table Bamboo and extends Bamboo's protocols for storing, querying and updating RDF(S) data. Atlas is being used currently to realize the metadata service of S-OGSA architecture in a fully distributed and scalable way. In this paper, we describe the operations of publishing and updating RDF information and answering one-time queries in Atlas.

Keywords: peer-to-peer networks, DHT, RDF, query processing, Semantic Grid

*This work is partially funded by FP6/IST project OntoGrid.
†Iris Miliaraki is supported by Microsoft Research through its European PhD Scholarship Programme

1. Introduction

A crucial operation in a Semantic Grid environment is the efficient publication and discovery of semantically annotated Grid services and resources. One of the goals of project OntoGrid[1] is to provide a scalable, robust and efficient way of providing such an operation in the Semantic Grid. To achieve this, we develop Atlas[2], a P2P system for the distributed storage, querying and updating of metadata expressed in RDF(S) describing Semantic Grid resources.

Our basic assumption in this paper is that Semantic Grid resources (e.g., machines, services or ontologies) will be annotated by RDF(S) metadata. The Resource Description Framework (RDF) and RDF Schema (RDFS) are frameworks for representing information about Web resources. RDF(S) consists of W3C recommendations that enable the encoding, exchange and reuse of structured metadata, providing the means for publishing both human-readable and machine-processable information and vocabularies for semantically describing things on the Web. Although RDF(S) was originally proposed in the context of the Semantic Web, it is also a very natural framework for representing information about Grid resources. As a result, it is used heavily in various Semantic Grid projects e.g., ^{my}Grid[3] or OntoGrid.

Atlas is built on top of the Bamboo DHT[4] and is currently deployed on the Everlab cluster (a private PlanetLab developed by the Evergrow project[5]). In OntoGrid, Atlas is used for implementing a fully distributed metadata service [11, 9]. The metadata service uses Atlas as a distributed RDF(S) database to store, query and update resource descriptions produced and consumed by other OntoGrid components. The metadata service is considered to be a core component of the S-OGSA architecture as described in [7].

In previous work [9], we have presented in detail the functionality provided by Atlas implementation v0.6. In this paper, we will present the latest implementation of Atlas (Atlas v0.8) and describe in detail the new features offered. The interested reader can also find more implementation details about Atlas v0.8 in [8].

The rest of the paper is organized as follows. Section 2 gives an overview of Atlas and a description of the data model and query language supported by Atlas. Sections 3, 4 and 5 present in detail the operations of publishing, updating and discovering a Semantic Grid resource respectively. Section 6 briefly discusses related work. Finally, Section 7 concludes the paper.

[1] http://www.ontogrid.net
[2] http://atlas.di.uoa.gr
[3] http://www.mygrid.org.uk
[4] http://bamboo-dht.org
[5] http://www.evergrow.org

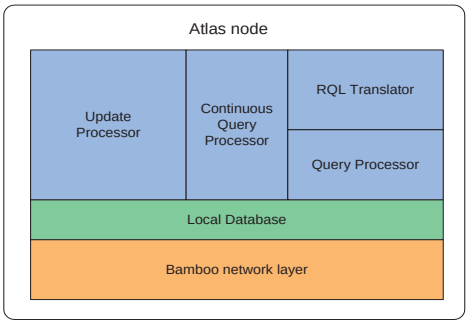

Figure 1. Architecture of an Atlas node

2. The Atlas system

Atlas is built on top of the Bamboo DHT [18]. Bamboo is a popular DHT that offers a simple interface consisting of the two operations: `put(ID, item)` and `get(ID)`. The `put` operation inserts an item with key `ID` and value `item` in the DHT. The `get` operation returns a pointer to the DHT node responsible for key `ID`. Also, in recent Bamboo releases, a `remove(ID, secret)` operation is also provided for removing the item with key `ID` and also needs a `secret` parameter for authorizations purposes. For the same reason, Bamboo offers an extended `put(ID, item, secret)` operation where one can also define a `secret` to ensure that remove operation is permitted only for an authorized user.

Our operations for storing, updating and querying RDF descriptions in Atlas, as described below, are based and implemented on top of these simple operations offered by the Bamboo DHT.

In the current implementation of the Atlas system, as described in detail in [8], the operations supported are publishing an RDF(S) description, submitting a query, subscribing with a continuous query and updating an RDF(S) description.

Figure 1 describes the architecture of an Atlas node. We distinguish between six components: the *Bamboo network layer*, which is responsible for routing and handling the network messages, the *update processor*, which is responsible for storing and updating RDF documents, the *continuous query processor*, which is responsible for evaluating continuous queries, the *query processor*, which is responsible for answering one-time queries, the *local database*, which is used for data storage locally in each node and the *RQL translator*, which is responsible for parsing RQL queries and transforming them to TQPL, which is the internal representation of queries handled by the *query processor*.

In this paper, we will describe in detail the indexing of RDF descriptions in Atlas and the one-time query processing algorithms. Algorithms for indexing

continuous queries are described in [13, 12] and will not be presented in this paper.

2.1 Data model and query language

We assume that Semantic Grid resources are annotated using the RDF data model. Resource descriptions are RDF documents written either in the RDF/XML format or the N3 format. Each RDF description is decomposed in a collection of RDF triples that are indexed in the Atlas network in a distributed way. A *triple* has the form $(subject, predicate, object)$ where $subject$ and $predicate$ are URIs and $object$ is a URI or a literal and represents some relationship, indicated by the predicate, that holds between the things denoted by subject and object of the triple.

Atlas supports one-time and continuous queries expressed in the RDF Query language (RQL) [10]. Each RQL query is parsed and transformed to an equivalent TPQL query, which is the language used for an internal representation of queries in Atlas. The *RQL translator* is the component that takes as input the RQL queries posed by the node. The details of how the translator works can be found in [9].

Let us describe the syntax of the TPQL query language as it is used in Atlas v0.8. A *TPQL conjunctive query* has form:

$$?x_1, \ldots, ?x_k : (s_1, p_1, o_1) \wedge (s_2, p_2, o_2) \wedge \cdots \wedge (s_n, p_n, o_n) \\ \wedge (?y_1\ op_1\ c_1) \wedge \cdots \wedge (?y_m\ op_m\ c_m)$$

where $?x_1, \ldots, ?x_k$, $?y_1, \ldots, ?y_m$ are variables, $s_1, \ldots, s_n$ are variables or URIs, $p_1, \ldots, p_n$ are variables or URIs and $o_1, \ldots, o_n$ are variables, URIs or literals. The formulas $(s_1, p_1, o_1), \ldots, (s_n, p_n, o_n)$ are called *triple patterns* and each variable $?x_i$ or $?y_i$ appears in at least one triple pattern. Each $(?y_i\ op_i\ c_i)$ is a constraint that restricts the range of the variable $?y_i$ based on the operator op_i and the constant c_i. Variables will always start with the '?' character. Operator op_i is one of the following operators: $>, <, =, \geq, \leq$ and LIKE. The variables $?x_1, \ldots, ?x_k$ are called *answer variables*.

Another class of TPQL queries supported by Atlas v0.8 results from adding the ability to express disjunctions of the form:

$$\phi_1 \vee \phi_2 \vee \ldots \vee \phi_m$$

where $\phi_1, \phi_2, \ldots, \phi_m$ are conjunctive queries.

3. Publishing Semantic Grid resources

In the previous implementation of Atlas (v0.6) described in [9], we presented algorithms for the evaluation of conjunctive TPQL queries without constraints of the form $(?y\ op\ c)$. In this paper we extend the algorithms of [9] in order

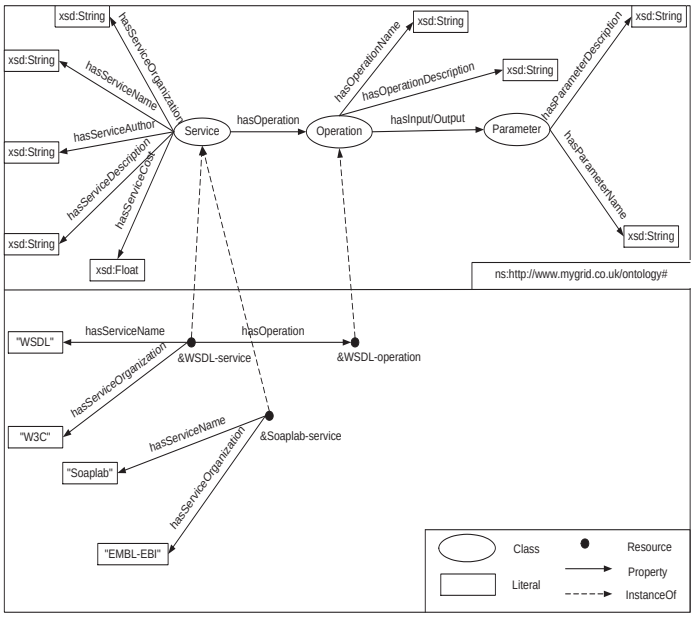

Figure 2. RDFS schema for Web Services

to deal with *range predicates* (i.e., constraints where c is an integer or a float and op is one of the following operators: $>, <, =, \geq$ and $\leq$). In order to support the distributed evaluation of queries with range predicates on attributes with numerical values, we should be able to keep ordering information for the numerical values stored in the DHT. A well understood solution from the literature, as presented in [6], [17] and [19], is to use a *locality preserving hash function* to hash the numeric values to the identifier space. Then, consecutive attribute values will be stored at nodes with consecutive identifiers.

Suppose that $DA = [low, high]$ is the domain for a numeric attribute. We define a *locality preserving hash function* $H(u) : DA \rightarrow \{0, 1, 2, ..., 2^m - 1\}$ such that if $u_i < u_j$ then $H(u_i) < H(u_j)$ for all $i, j \in \{0, 1, 2, ..., 2^m - 1\}$, where m is the identifier length. As proposed in [6], we can use the following locality-preserving hash function:

$$H(u) = (u - u_{min}) \times \frac{2^m - 1}{u_{max} - u_{min}}, \text{ where } u \in [u_{min}, u_{max}].$$

Let us assume that node x wants to store a triple $t = (s, p, o)$ in the network. t will be indexed on the DHT *three times*, once for its subject, once for its predicate and once for its object. To do this, node x computes the identifiers of the nodes that will store t as follows: $I_1 = Hash(s)$ and $I_2 = Hash(p)$, where $Hash()$ is the SHA-1 function [16]. If the object is numeric, we use

the locality preserving hashing function $H(u)$ to compute the third identifier based on the object's value, so that $I_3 = H(o)$. Otherwise, the third identifier is computed as follows: $I_3 = Hash(o)$. Identifiers I_1, I_2, I_3 are used to locate the nodes that will receive t and store it in their local database.

Apart from indexing an RDF description, Atlas also supports updates of RDF descriptions using atomic update operations expressed in the RDF Update Language (RUL) defined in [15].

4. Updating Semantic Grid resources

Supporting updates in DHTs has not received much attention so far, and even state of the art DHTs such as Bamboo currently offer very little update functionality. Our current implementation of RUL atomic updates relies directly on the atomic primitives `put/remove` provided by the Bamboo API. We hope that a new generation of DHTs will provide greater functionality regarding updates so that the full functionality of an update language such as RUL can be correctly implemented.

4.1 RUL statements

RUL [15] is an update language for RDF graphs which supports *insertions*, *deletions* and *replacements* of class and property instances both in an *atomic* and a *set-oriented* fashion. The term atomic is used for an update operation which affects a single class or property instance, while set-oriented is used for an operation where a set of instances is updated. The syntax of any RUL statement is the following:

```
UpdateStatement ClassOrProperty(UpdateExpression)
[FROM VariableBinding] [WHERE Filtering]
[USING NAMESPACE NamespaceDefs]
```

The non-terminal `UpdateStatement` can be an `INSERT`, `DELETE` or `REPLACE` keyword to introduce the corresponding statement for class or property instances. The `ClassOrProperty` non-terminal can be a schema variable or a class or property name, while `UpdateExpression` is an expression introducing the instance or instances affected by the update. The `FROM` clause determines the bindings of any variables appearing in the update clause as in RQL. The clause `WHERE` gives as usual the filtering conditions for the variables' bindings introduced in the clause `FROM`. The clause `USING NAMESPACE` gives a list of namespaces that disambiguate the use of names in the other clauses. The clauses `FROM`, `WHERE` and `USING NAMESPACE` are optional.

Atomic RUL updates do not use the `FROM` and `WHERE` clause. The following are two examples of atomic updates, using the RDF Schema presented in Figure 2, which describes a Web service.

EXAMPLE 1 *Insert the resource* `www.uoa.gr/MetadataService` *as a new Service:*

```
INSERT ns:Service(&www.uoa.gr/MetadataService)
USING NAMESPACE ns=&http://www.mygrid.co.uk/ontology#
```

EXAMPLE 2 *Delete resource* & `WSDL-service` *from the graph:*

```
DELETE ns:Service(&WSDL-service)
USING NAMESPACE ns=&http://www.mygrid.co.uk/ontology#
```

4.2 Implementation

To deal with updates in Atlas v0.8, we have extended the *RQL translator* component to support parsing of RUL statements and the *update processor* in order to implement them. In what follows, we present details of the implementation of each one of the supported update operations (i.e., `INSERT`, `DELETE` and `REPLACE`).

`INSERT` **Statements.** When a user/client wants to insert new triple(s), he contacts a node x and poses an `INSERT` statement to the Atlas network. Along with the `INSERT` statement a *secret* value is given. `INSERT` statements are implemented by indexing the triple three times using the `put` primitive of Bamboo as described in the previous section. Let us note here, that the user/client posing the request is also responsible for keeping track of the secret value corresponding to each inserted triple, which will make him capable of deleting a triple.

`DELETE` **Statements.** Whenever a user/client wants to delete a triple, he poses a `DELETE` statement to a node x. Since the `INSERT` operation stores each triple three times, it is straightforward that three `remove` operations need to be issued to Bamboo to implement the deletion. In order for this request to be effective, the secret that was provided in the three corresponding `put` operations has to be *revealed*. The user/client who initially stored a triple is responsible to "remember" the secret value provided and reveal it with the corresponding `DELETE` statement.

`REPLACE` **Statements.** There is no replace operation supported by Bamboo, thus whenever a user/client poses a `REPLACE` statement to a node x, this node poses a sequence of three `remove` operations followed by three `put` operations to Bamboo. These operations are similar to the ones described above.

5. Discovering Semantic Grid resources

If a user of Atlas wants to find out about a Semantic Grid resource, he has to pose an RQL query. In this section, we extend the algorithms of [9],

[14] for one-time query processing to enable distributed evaluation for queries that contain range predicates.

5.1 The QC algorithm

Our main addition to the algorithm QC as described in [9], [14] concerns range predicates and is based on indexing numerical values using a *locality preserving hash function*.

Assume a node x poses a query q that has the form

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \wedge c_1 \wedge c_2 \wedge \ldots \wedge c_m$$

which consists of the triple patterns $\phi_1, \phi_2, \ldots, \phi_n$ and the constraints $c_1, c_2, \ldots, c_m$. Each triple pattern ϕ_i and the constraints that refer to its variables, form a conjunction ψ_i. Each conjunction ψ_i of q will be evaluated by a possibly different node; these nodes form the *query chain* for q. The order we use to evaluate each conjunction is crucial and we will discuss the issues involved later on. Now, for simplicity we assume that we first evaluate the first conjunction, then the second and so on. Each conjunction ψ_i can be evaluated either by using the constant part of the triple pattern ϕ_i, or by using the range constraints.

Evaluating a conjunction using the constant part. In this case, node x chooses the node that will evaluate conjunction ψ_1 by using one of the constants in ϕ_1. In case that ϕ_1 has multiple constants, x will heuristically prefer to use first the subject, then the object and finally the predicate to determine the node that will evaluate ϕ_1. Intuitively, there will be more *distinct* subject or object values than *distinct* predicates values in an instance of a given schema. Thus, our decision help us to achieve a better distribution of the query processing load.

Evaluating a conjunction using the range predicates. When node x wants to evaluate a conjunction $\psi_i = \phi_i \wedge c_1 \wedge \ldots$ by using its range predicate, it sends a message to node n, who is responsible for the lower bound of the specified numeric range. Node n searches locally for relevant triples and then forwards the query to its immediate neighbor until the node that is responsible for the higher bound of the range is reached. Then, this node returns back answers to the node that posed the query.

Order of nodes in a query chain. The order in which each conjunction is evaluated is crucial, and affects network traffic, query processing load or any other resource that we try to optimize. If we want to minimize the size of the messages exchanged, we can choose to evaluate conjunctions with *low selectivity* in the beginning of the query evaluation. Selectivity information can

be made available to each node if statistics regarding the contents of the local triple tables are available. Then, when a node n determines the next conjunction ψ_{i+1} to be evaluated, n has enough statistical information to determine a good node to continue the query evaluation.

The same statistics could be used in order to help a node to choose whether it should evaluate a conjunction using the constant parts of the triple pattern or by using the range predicate. If the evaluation of a conjunction using the range predicates seems to require less effort (in terms of network messages and query processing load) than the evaluation using the constant parts of the triple, a node would prefer to use the former method to evaluate it.

5.2 The SBV algorithm

Let us now present the algorithm *spread-by-value* (SBV). SBV was presented originally in [14] and extends the ideas of the algorithm QC for achieving a better distribution of the query processing load. SBV does not create a single chain for a query, but exploits the values of matching triples found while processing the query incrementally and distributes the responsibility of evaluating a query to more nodes. We have extended SBV to deal with range predicates as well.

Indexing triples. Assume that we want to index a new triple $t = (s, p, o)$. In SBV, t will be stored at the responsible nodes of the identifiers $Hash(s)$, $Hash(p)$, $Hash(o)$, $Hash(s+p)$, $Hash(s+o)$, $Hash(p+o)$ and $Hash(s+p+o)$, where the operator $+$ denotes the *concatenation* of string values. We use the concatenation of constant parts whenever possible, since the number of possible identifiers that can be created by a combination of constant parts is definitely higher and will allow us to achieve a better distribution of the query processing load. If the object is numeric, we use a locality preserving hashing function $H(u)$, as described in Section 3, for computing the identifier.

Assume a node x poses a query q that has the form

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \wedge c_1 \wedge c_2 \wedge \ldots \wedge c_m$$

which consists of the triple patterns $\phi_1, \phi_2, \ldots, \phi_n$ and the constraints $c_1, c_2, \ldots, c_m$. As previously, each triple pattern ϕ_i and the constraints that refer to its variables form a conjunction ψ_i, and each conjunction ψ_i of q will be evaluated by a possibly different node. Again, we assume that we first evaluate the first conjunction, then the second and so on. Each conjunction ψ_i can be evaluated either by using the constant part of the triple pattern ϕ_i, or by using the range constraints.

In contrast with QC, the query plan produced by SBV is created *dynamically* by exploiting the values of the matching triples that nodes find at each step in order to achieve a better distribution of the query processing load. For example,

n_1 will use the values for variables of ψ_1, that it will find in local triples matching ψ_1, to bind the variables of $\psi_2 \wedge \cdots \wedge \psi_n$ that are common with ψ_1 and produce a new set of queries that will jointly determine the answer to the original query q. Since we expect to have multiple matching values for the variables of q_1, we also expect to have *multiple next nodes* where the new queries will continue their evaluation. The nodes at the leafs of these chains will deliver answers back to the node that submitted the query q.

Evaluating a conjunction using the constant part. Node x chooses the node that will evaluate conjunction ψ_1 by using one of the constants in ϕ_1. In case that ϕ_1 has multiple constants, x uses the *combination* of all constant parts. For example, if $\phi_j = (?s_j, p_j, o_j)$, then $I_j = Hash(p_j + o_j)$.

Evaluating a conjunction using the range predicates. In this case node x chooses the node that will evaluate conjunction ψ_1 based on the numeric range defined by the range predicate. Query evaluation proceeds in the same manner as in QC, by starting at node n that is responsible for the lower bound of the specified numeric range and ending when the node responsible for the higher bound is reached.

6. Related Work

In this section, we will present related work for RDF(S) query processing on top of DHTs.

In [6], Min Cai et al. studied the problem of evaluating RDF queries in a scalable distributed RDF repository, named RDFPeers. RDFPeers is implemented on top of MAAN [5], which implements multi-attribute and range queries with the use of a *locality preserving* hash function. [6] was the first work to consider RDF queries on top of a DHT and proposed algorithms for evaluating triple pattern queries, range queries and conjunctive multi-predicate queries for the one-time query processing scenario. Our algorithm QC is essentially an extension of the algorithm of RDFPeers to the class of TPQL queries.

Another interesting work is the GridVine system [3]. GridVine is built on top of P-Grid [1] and can deal with the same kind of queries as RDFPeers. In addition, it has an original approach to global semantic interoperability by utilizing gossiping techniques [2].

Finally, [4] is a recent paper which considers RDF(S) reasoning on top of DHTs.

7. Conclusion

We have presented Atlas v0.8 and discussed the operations of publishing and updating RDF information and answering one-time queries. We refer the

reader to [14] for a detailed evaluation of our algorithms using simulation. Currently, we are evaluating our algorithms in PlanetLab using the complete implementation of Atlas.

References

[1] Karl Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *Proceedings of 9th International Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, 2001.

[2] Karl Aberer, Philippe Cudre'-Mauroux, and Manfred Hauswirth. Start making sense: The Chatty Web approach for global semantic agreements. *Journal of Web Semantics*, 1(1), 2003.

[3] Karl Aberer, Philippe Cudre-Mauroux, Manfred Hauswirth, and Tim Van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW2004)*, New York, May 2004.

[4] Dominic Battre, Andre Hoing, Felix Heine, and Odej Kao. On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT based RDF stores. In *Fourth International Workshop on Databases, Information Systems and Peer-to-Peer Computing in scope of 32st International Conference on Very Large Data Bases (VLDB 2006)*, Seoul, Korea, September 2006.

[5] M. Cai, M. Frank, and P. Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003)*, 2003.

[6] M. Cai, M. R. Frank, B. Yan, and R. M. MacGregor. 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, December 2004.

[7] Óscar Corcho, Pinar Alper, Ioannis Kotsiopoulos, Paolo Missier, Sean Bechhofer, and Carole A. Goble. An overview of S-OGSA: A reference Semantic Grid architecture. *Journal Of Web Semantics*, 2006.

[8] Zoi Kaoudi, Kostis Kyzirakos, Matoula Magiridou, Iris Miliaraki, Antonis Papadakis-Pesaresi, Erietta Liarou, Stratos Idreos, George Anadiotis, Manolis Koubarakis, Spiros Skiadopoulos, and Evi Pitoura. Deployment of Ontology Services and Semantic Grid Services on top of Self-organized P2P Networks. Deliverable D4.2v2, Ontogrid project, April 2007.

[9] Zoi Kaoudi, Iris Miliaraki, Matoula Magiridou, Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Semantic Grid Resource Discovery in Atlas. *Knowledge and Data Management in Grids*, 2006. Talia Domenico and Bilas Angelos and Dikaiakos Marios D. (editors), Springer.

[10] Greg Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th International World Wide Web Conference*, May 2002.

[11] Manolis Koubarakis, Iris Miliaraki, Zoi Kaoudi, Matoula Magiridou, and Antonis Papadakis-Pesaresi. Semantic Grid Resource Discovery using DHTs in Atlas. In *3rd GGF Semantic Grid Workshop*, Athens, Greece, 13-16 February 2006.

[12] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Continuous RDF Query Processing over Distributed Hash Tables. In *Submitted to conference*.

[13] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Publish-Subscribe with RDF Data over Large Structured Overlay Networks. In *Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2005), VLDB 2005 31st International Conference on Very Large Data Bases*, Trondheim, Norway, 28-29 August.

[14] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In *Proceedings of 5th the International Semantic Web Conference (ISWC 2006)*, Athens, GA, USA, November 2006.

[15] Matoula Magiridou, Stavros Sahtouris, Vassilis Christophides, and Manolis Koubarakis. RUL: A Declarative Update Language for RDF. In *Proceedings of the Fourth International Semantic Web Conferece (ISWC2005)*, 2005.

[16] National Institute of Standards and Technology. Secure hash standard, 1995. Publication 180-1.

[17] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Replication, Load Balancing and Efficient Range Query Processing in DHTs. In *10th International Conference on Extending Database Technology (EDBT06)*, 2006.

[18] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *USENIX Annual Technical Conference*, 2004.

[19] Peter Triantafillou, Nikos Ntarmos, and Theoni Pitoura. The RangeGuard: Range Query Optimization in Peer-to-Peer Data Networks. In *3rd Hellenic Data Management Symposium, HDMS04*, June 28-29 2004.