GridVine: Building Internet-Scale Semantic Overlay Networks*

Karl Aberer[1], Philippe Cudré-Mauroux[1], Manfred Hauswirth[1], and Tim Van Pelt[2]

[1] School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL), Switzerland
{karl.aberer, manfred.hauswirth, philippe.cudre-mauroux}@epfl.ch
[2] Department of Computer and Information Science
Linköpings Universitet, Sweden
tim@vanpelt.com

Abstract. This paper addresses the problem of building scalable semantic overlay networks. Our approach follows the principle of data independence by separating a logical layer, the semantic overlay for managing and mapping data and metadata schemas, from a physical layer consisting of a structured peer-to-peer overlay network for efficient routing of messages. The physical layer is used to implement various functions at the logical layer, including attribute-based search, schema management and schema mapping management. The separation of a physical from a logical layer allows us to process logical operations in the semantic overlay using different physical execution strategies. In particular we identify iterative and recursive strategies for the traversal of semantic overlay networks as two important alternatives. At the logical layer we support semantic interoperability through schema inheritance and Semantic Gossiping. Thus our system provides a complete solution to the implementation of semantic overlay networks supporting both scalability and interoperability.

1 Introduction

Research on semantic overlay networks has recently received a lot of attention in the emerging area of peer-to-peer data management [10,6,15,17]. All of these approaches are based on a generalization of the concept of federated databases, where peers store data or (content) metadata according to their local schemas and freely define mappings (translations, views) from their schemas to those of

* The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and in part and by the Swiss National Funding Agency OFES as part of the European project KnowlegeWeb No 507482.

other peers. Thus a network is constructed where peers are logically interconnected through schema mappings and where queries can be propagated to other peers using different schemas, even over multiple hops. The main concern of these works is on the problem of consistent query answering and reconciliation of different mappings occurring in the semantic overlay network [8,22].

In parallel, structured overlay networks, e.g., Chord [21] or P-Grid [2], have been developed as new infrastructures for routing requests for resources that are distributed over large populations of peers using application-specific keys in an efficient manner. Naturally, such networks can be employed in order to efficiently respond to simple keyword-based queries. Structured overlay networks clearly also have the potential to support the efficient operation of a semantic overlay network. However, research on how to take advantage of this potential is in its infancy.

In this paper, we introduce an architecture and implementation leveraging on the potential for scalability offered by structured overlay networks in the realization of interoperable, large-scale semantic overlay networks. A key aspect of the approach we take is to apply the principle of data independence [13] by separating a logical from a physical layer. This principle is well-known from the database area and has largely contributed to the success of modern database systems. At the logical layer, we support various operations to maintain the semantic overlay network and to support semantic interoperability, including attribute-based search, schema management, schema inheritance and schema mapping. We also provide support for a specific schema reconciliation technique, i.e., Semantic Gossiping, that we have introduced earlier [3]. We provide these mechanisms within the standard syntactic framework of RDF/OWL. At the physical layer, we provide efficient realizations of the operations exploiting a structured overlay network, namely P-Grid. This requires mappings of operations and data to the physical layer. Important aspects of this mapping are:

– The mapping of data and metadata to routable keys.
– The introduction of a specific namespace for resources present in the peer space, such that the infrastructure can resolve resource requests.
– The implementation of traversals of the semantic network taking advantage of intermediate schema mappings. An interesting aspect is the possibility to use different strategies to implement such traversals at the structured overlay network layer, in ways that are substantially different from naive solutions. We analyze two types of processing strategies, iterative and recursive. As in standard database query processing, the data independence principle thus opens up the possibility of optimization using different query processing strategies.

We have developed a first implementation following the architectural principles outlined above building on the existing implementation of the P-Grid structured overlay network. We report on initial experiments showing the effect of using different query processing strategies for semantic network traversal.

The rest of this paper is structured as follows: We start with an overview of our approach in Section 2. Our architecture and implementation use P-Grid as a physical layer, which is briefly described in Section 3. Section 4 presents the mechanisms used to index metadata and schemas and to resolve queries. Section 5 describes semantic interoperability while Section 6 is dedicated to GridVine, the implementation of our approach. Finally, we discuss related work in Section 7 and conclude.

2 Overview of Our Approach

2.1 Data Independence

Following the principle of data independence enounced above, our approach revolves around a two-layer model: a physical layer based on the P-Grid access structure underpinning GridVine, a logical semantic overlay layer (see Fig. 1). P-Grid (Section 3) is an efficient, self-organizing and fully decentralized access structure based on a distributed hash table (DHT). GridVine uses two of P-Grid's basic functionalities: the *Insert(key, value)* primitive for storing data items based on a key identifier and the *Retrieve(key)* primitive for retrieving data items given their key.

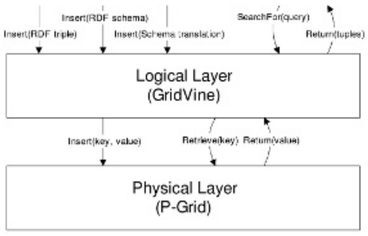


Fig. 1. The two-layer model

Taking advantage of these two rather limited primitives, we build a full-fledged semantic overlay network on top of P-Grid. The system exposes a new set of primitives (depicted on top of Fig. 1) allowing end-users to insert metadata, schemas and schema translations as well as retrieve semantic information using expressive query languages. Capitalizing on recent developments, we chose the RDF / RDFS pair as languages to encode metadata and vocabulary definitions in GridVine. These two languages represent the fundamental building blocks of the emerging Semantic Web [23] and are predestined to become *de facto* standards for encoding metadata as well as their corresponding schematic information.

The exact mechanisms we choose for inserting metadata into the P-Grid are naturally of utmost importance, since they directly influence the query capabilities of the overall system, and are extensively discussed in the following (Section 4). In order to support the processing of schema-specific information, we introduce a meta-schema specifying common characteristics for all custom schemas derived by the users. Also, we introduce new addressing spaces, i.e., URI schemes, to identify resources both in the physical (P-Grid data items) and logical (semantic information) layers.

2.2 Decentralized Semantics

Classification of resources and definition of vocabularies are essential for leveraging metadata creation and fostering semantic interoperability through reuse of conceptualizations. Legacy information sharing systems typically support static sets of centrally imposed, predefined schemas. We consider such a monolithic approach as far too rigid for adequately capturing information sources in a network of autonomous and heterogeneous parties. Not only is this not desirable from an ideological perspective, it also misses out on the power of P2P. Seeing users as experts of the information they share, they themselves are most fit to come up with a proper schema to describe their data. However desirable it may be to let users come up with their own schemas in a bottom-up manner, it also severely endangers global semantic interoperability and search capabilities: How could one ensure optimal precision and recall when searching for data items that might be referred to by a large variety of terms? Our answer to this question if twofold, including both schema inheritance and Semantic Gossiping mechanisms.

Schema inheritance provides GridVine with basic schema reusability and interoperability capabilities. As for other social networks [7], we expect the popularity of schemas in GridVine to follow scale-free preferential attachment laws, such that a small subset of schemas gain unparalleled popularity while the others remain mainly confidential. By allowing users to derive new schemas from well-known base schemas, we implicitly foster interoperability by reusing sets of conceptualizations belonging to the base schemas.

Semantic Gossiping [3,4] is a semantic reconciliation method that can be applied to foster semantic interoperability in decentralized settings. The method aims at establishing global forms of agreement starting from a graph of purely local mappings among schemas. Following this approach, we allow peers in GridVine to create, and possibly index, translation links mapping one schema onto another. These links can then be used to propagate queries in such a way that relevant data items annotated according to different schemas can also be retrieved. Query forwarding can be implemented using several approaches. In the following, we identify two radically different strategies for forwarding queries: iterative forwarding, where peers process series of translation links repeatedly, and recursive forwarding, where peers delegate the forwarding to other peers. Schema inheritance and Semantic Gossiping are further described in Section 5.

3 The P-Grid P2P System

GridVine uses our P-Grid [2] P2P system as its physical layer. P-Grid is based on the principles of distributed hash tables (DHT) [18]. As any DHT approach, P-Grid associates peers with data keys from a key space, i.e., partitions of the underlying distributed data structure. Each peer is responsible for some part of the overall key space and maintains additional (routing) information to forward queries and requests. Without constraining general applicability, we use binary keys in the following. P-Grid peers refer to a common underlying tree structure in order to organize their routing tables. In the following, we assume that the tree is binary. This is not a fundamental limitation as a generalization of P-Grid to k-ary structures has been introduced in [5], but will simplify the presentation.

Each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level, which enables the implementation of prefix routing for efficient search. The cost for storing the references (in routing tables) and the associated maintenance cost are scalable as they are proportional to the depth of the underlying binary tree.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $key(d)$ is calculated using an order-preserving hash function. $key(d)$ has $\pi(p)$ as prefix but we do not exclude that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. In addition, peers also maintain references $\sigma(p)$ to peers having the same path, i.e., their replicas.

P-Grid supports two basic operations: *Retrieve(key)* for searching a certain key and retrieving the associated data item and *Insert(key, value)* for storing new data items. Since P-Grid uses a binary tree, *Retrieve(key)* intuitively is efficient, i.e., $\mathcal{O}(log(|\Pi|))$, measured in terms of messages required for resolving a search request, in a balanced tree. For skewed data distributions we show in [1] that due to the probabilistic nature of the P-Grid approach, the expected search cost measured by the number of messages required to perform the search remains logarithmic, independently how the P-Grid is structured. This is important as it allows us to apply simple order-preserving hashing functions to metadata annotations, which may lead to non-uniformly distributed key distributions. As P-Grid uses an order-preserving hash function to compute keys and define their association with peers, it processes prefix and range queries of arbitrary granularity efficiently, i.e., $O(log(|\Pi|) + |\{p|\kappa \ is \ prefix \ of \ \pi(p)\}|$. Prefix queries will be an important constituent in the generic implementation of metadata queries. *Insert(key, value)* is based on P-Grid's more general update functionality [11] which provides probabilistic guarantees for consistency and is efficient even in highly unreliable, replicated environments, i.e., $\mathcal{O}(log(|\Pi|) + replication \ factor)$.

4 Semantic Support

In the following, we elaborate on how GridVine handles the creation and indexing of RDF triples (Section 4.1) and schemas (Section 4.2). Section 4.3 discusses then query resolution mechanisms.

4.1 Metadata Storage

In GridVine, statements are stored as RDF triples and refer to data items shared in the P-Grid infrastructure. A structured overlay network allows to implement an application specific addressing space. In our case, we introduce a specific URI schemes *pgrid* : // for resources, and *pgrids* : //, for schema-elements. This does not exclude the use of other URI schemes in conjunction with P-Grid's specific ones, as long as the infrastructure ensures that all the identifiers can be resolved.

In the case were all resources are identified by P-Grid URIs, a typical situation would be a statement where the subject is identified by a P-Grid key, i.e., a binary string such as *11110101*, whereas the predicate and object refer to P-Grid's specific RDF schemas (or literals). This allows us to constrain the applicability of the schema constructs. An example of such a statement would be *the P-Grid resource 11110101* (*subject*) *is entitled* (*predicate*) *Rain, Steam and Speed* (*object*), which, translated into the XML syntax of RDF, would result in a file like the one transcribed in Fig. 2.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="pgrid://11110101">
    <Title xmlns="pgrids://01001101:bmp#">Rain, Steam and Speed</Title>
  </rdf:Description>
</rdf:RDF>
```

Fig. 2. An RDF statement encoded in XML

Since most RDF query languages [19] are based on constraint searches on the triples' *subject*, *predicate* or *object*, we reference each individual triple three times, generating separate keys based on their *subject*, *predicate* and *object* values. Thus, the insertion operation of a triple $t \in T$ is performed as follows:

$$Insert(t) \equiv \; Insert(t_{subject}, t), Insert(Hash(t_{predicate}), t), Insert(Hash(t_{object}), t).$$

Prefix searches, e.g., on the beginning of a string representing an object value, are inherently supported by P-Grid's routing mechanisms. Supporting substring searches imposes to index all the suffixes of a generated key as well. Thus, if we introduce l as the average length of the strings representing *subjects*, *objects* or *predicates*, $3l$ *Insert()* operations incur on average when indexing an RDF triple in GridVine.

4.2 Schema Definition and Storage

We encode category descriptions using RDF Schema (RDFS). RDFS is an extension of RDF providing mechanisms for describing groups of resources and their relationships. Among other capabilities, it allows to define classes of resources (*classes*) and predicates (*properties*) and to specify constraints on the subject (*domain*) or the object (*range*) of a given class of predicates.

GridVine schemas allow to declare new *categories* to describe application specific resources. The categories are all derived by subclassing from a generic RDF class called $P - GridDataItem$ representing any P-Grid addressable resource. Properties referring to this class as domain allow to declare application-specific vocabularies (i.e., metadata attributes) with arbitrary values as ranges.

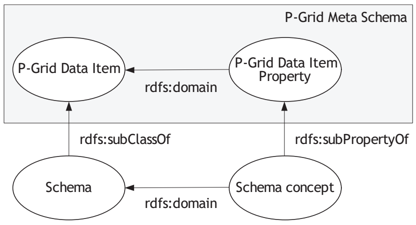


Fig. 3. Relations of a P-Grid custom schema to the P-Grid meta schema

All properties derive from a generic $P - GridDataItemProperty$ property. P-Grid meta-schema and its relation to user-defined RDF schemas are summarized in Fig. 3.

We create distinct RDFS files for every category, grouping the definition of a subclass as well as all its affiliated properties. We create a unique identifer for the category by concatenating the path $\pi(p)$ of the peer creating a category to the category itself. We then insert it into P-Grid as any other file:

$Insert(rdf\_schema) \equiv Insert(Hash(\pi(p) : class\_name), rdf\_schema)$.

Note that due to the possibility of performing substring searches, schemas can also be searched by their category name only.

4.3 Resolving Queries in GridVine

The simplest query one may pose against the system consists in a triple pattern with only one bound variable, i.e., a query retrieving (parts of) a set of triples given the value of their *subject*, *predicate* or *object*. For example, the following RDQL [20] query: *SELECT ?y WHERE (<pgrid://01101000>, ?y,?z)* returns all the predicates used to annotate data item *01101000*. In GridVine, we call such a query a *native query*. Native queries are resolved routing one message through the P-Grid infrastructure. In our case, a message containing the above query and the address of the peer p from which the query originates is routed as explained in Section 3 to the peer(s) q responsible for storing data item d with $key(d)$ = 01101000. Upon reception of the query, q checks its local database and sends back to p the set of data items $\delta(q)$ matching the query, which in turn parses the answer and displays the result. The whole process still generates $\mathcal{O}(log(|\Pi|))$ messages: $\mathcal{O}(log(|\Pi|))$ messages to resolve the P-Grid entry plus one message for the answer. Native queries on literals (e.g., searches on the value of a property) are resolved exactly in the same way, but start beforehand by hashing the literal in order to get its corresponding key.

Variables can be introduced as subject, predicate or object of a triple pattern. GridVine resolves triple patterns differently depending on the number of unbound variables they contain:

Three unbound variables triple patterns would retrieve all the triples stored in the system, implying $\mathcal{O}(|\Pi| log(|\Pi|))$ messages. They are not allowed in GridVine.

Two unbound variables triple patterns are standard native queries, and may as such be resolved using the method described above.

One unbound variable triple patterns can be resolved by issuing a native query; The predicate of the query may either be the first or the second bound expression of the triple patterns. The query issued must include both predicates in order for the query destination to filter out the triples correctly.

Zero unbound variable triple patterns are constant and require no further resolution.

The cost of resolving triple patterns grows logarithmically with the number of peers, making GridVine scale gracefully in the large. Triple patterns are powerful primitives that can be used to support more expressive query languages as well. GridVine supports RDQL query resolution through simple triple pattern combinations, following strategies similar to the ones presented in [10].

5 Semantic Interoperability

As previously mentioned, we believe that semantic heterogeneity is a critical threat for large-scale semantic networks. We detail below the mechanisms we take advantage of in order to foster semantic interoperability in GridVine.

5.1 Schema Inheritance

We let users derive new categories from the existing ones. However, we impose that the new class representing the subcategory subclasses the base category class. RDFS enforces monotonic inheritance of properties through class hierarchies; In our case, the subcategory automatically inherits the properties defined in the base category through the *domain* definitions. Additionally, subcategories may introduce sets of new properties specific to the subclass. Fig. 4 below provides an example where a category for annotating JPEG files is derived from a more generic category of image files.

The process can of course be applied recursively in the sense that a subcategory may in turn serve as a super-category for a new derivation, creating complex hierarchies of categories and classes from the most popular base schemas. Since subcategories subsume their base categories, subcategories of a given category may be used indifferently as instances of the base category. In particular, searches on a property belonging to a base category automatically affect subcategory instances as well (least-derived property indexation). Thus, we create

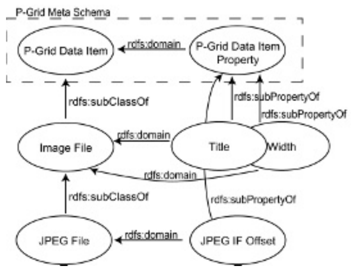


Fig. 4. A simple example of category inheritance

sets of semantically interoperable schemas through properties inherited by all descendants of a (potentially very popular) base schema.

5.2 Semantic Gossiping

In [3,4], we introduced Semantic Gossiping as a new semantic reconciliation method. Semantic gossiping aims at establishing global forms of agreement starting from a graph of purely local mappings among schemas. Peers that have annotated their data according to the same schema are said to belong to the same *semantic neighbourhood*. Each peer has the possibility to create (either manually or automatically) a mapping between two schemas, in effect creating a link between two semantic neighbourhoods. The network as such can be seen as a directed graph of translations.

This translation graph exhibits two interesting properties: First, using local translations and with the possibility to learn about other existing translations in the system, transitivity allows for the forwarding of queries to semantic domains for which there is no direct translation link (transitive closure). A second observation is that the graph has cycles. One of the fundamental assumptions that underlies the approach is that the translations between different semantic domains may be partially or totally *incorrect*. Analysis of composite cycles and returned results makes it possible to check the quality of translations and to determine the degree of semantic agreement in a community, as described in [3].

Following this approach, we allow peers in GridVine to create translation links mapping one schema onto another. Translations are used to propagate queries from one semantic domain to another (see Section 6.1). Since RDFS does not support schema mapping, we encode translations using OWL [23]. Translation links consist of series of *owl:equivalentProperty* statements which characterize the correspondences between the two categories at the property level. Fig. 5 below is an example of translation. Individual property equivalence statements are reified in order to account for partially-overlapping properties: Peers can thus refer to the various equivalence statements and qualify the individual mappings with a semantic similarity value as introduced in our Semantic Gossiping papers [3,4].

```
<?xml version="1.0"?> <?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:RDF xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Image_Description xmlns="pgrids://10000101:exif#">
    <owl:equivalentProperty rdf:ID="m1" rdf:resource="pgrids://01001101:bmp#Title"/>
  </Image_Description>
  <Exif_Image_Width xmlns="pgrids://10000101:exif#">
    <owl:equivalentProperty rdf:ID="m2" rdf:resource="pgrids://01001101:bmp#Width"/>
  </Exif_Image_Width>
</rdf:RDF>
```

Fig. 5. An example of translation

To forward queries properly, we need to retrieve all the translation links relating a given schema to other schemas. Thus, we index the translations based on their source schemas (e.g., *pgrids://10000101:exif* in Fig. 5):

$$Insert(owl\_file) \equiv Insert(Hash(\pi(p_{source}) : source\_class\_name), owl\_file).$$

Forwarding a query is then logically handled using gossiping as described in [3, 4]: Starting from a given semantic domain, the query gets transformed and iteratively traverses other semantic domains following translations links until it is considered as being too different (either from a syntactic or a semantic point of view) from the original query. A new query is issued after each translation step. In the following section, we show that different physical implementations of Semantic Gossiping can be realized using the P-Grid overlay network.

6 Implementation

6.1 Architectural Overview

GridVine was implemented by extending our existing Java-based P-Grid library which is available upon request. Fig. 6 shows the architecture of the implementation as a UML class diagram. The left side shows the P-Grid library with the Gridella GUI and the right side shows GridVine's semantic extensions. The arrows in the figure denote "uses" relationships. The *Gridella* component provides a GUI and uses the *P-Grid* component to issue queries; the *Semantics* component uses the *P-Grid* component to issue and receive queries and uses the *RDF*, *RDFS*, and *OWL* components to handle incoming requests. The *RDF* component is responsible for creating and managing RDF based metadata and provides the gossiping functionality. The *Extractors* subcomponent facilitates automatic metadata extraction to leverage the burden of manual annotation (e.g., automatic extraction of EXIF information from images). Functionalities related to schemas are provided by the *RDFS* component, while the *OWL* component handles all issues regarding translations. The *P-Grid*, *Semantics*, *RDF*, *RDFS*, and *OWL* components are implemented as *Singletons*, i.e., only a single instance of each of these classes exists at runtime and handles all requests.

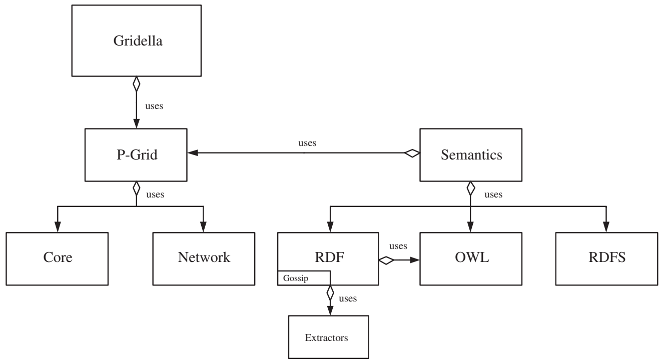
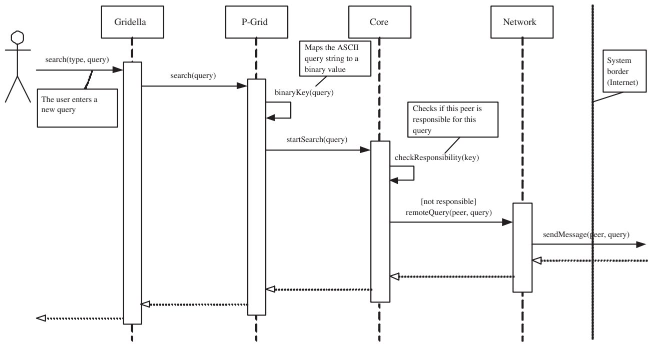
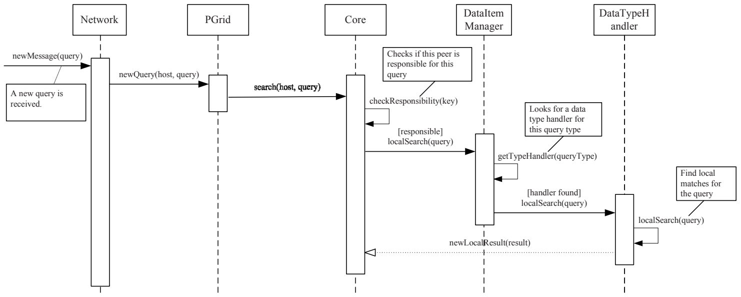


Fig. 6. The GridVine component model


Gridella
P-Grid
Core
Network
search(type, query)
The user enters a new query
search(query)
Maps the ASCII query string to a binary value
binaryKey(query)
startSearch(query)
Checks if this peer is responsible for this query
checkResponsibility(key)
[not responsible] remoteQuery(peer, query)
sendMessage(peer, query)
System border (Internet)

(a) Initiating a query


Network
PGrid
Core
DataItem Manager
DataTypeHandler
newMessage(query)
A new query is received.
newQuery(host, query)
search(host, query)
Checks if this peer is responsible for this query
checkResponsibility(key)
[responsible] localSearch(query)
Looks for a data type handler for this query type
getTypeHandler(queryType)
[handler found] localSearch(query)
Find local matches for the query
localSearch(query)
newLocalResult(result)

(b) Handling an incoming query

Fig. 7. GridVine queries

Querying. Fig. 7(a) shows the initiator's side of a query in P-Grid which GridVine uses to provide its semantic search capabilities.

The user initiates a query via the Gridella GUI which hands it over to the P-Grid component to perform the actual search. The parameter type defines the type of data to search for (GUID, File, RDF, RDFS, OWL) and is implicitly assigned by the system (and encoded in the query). The query is then routed to other peers which are "closer" to the required result as described in Section 3.

If a peer receives a query, it checks whether it can answer the query, i.e., wether it is responsible for the partition of the key space the query belongs to, otherwise the query will be forwarded as shown in Fig. 7(a). If the peer can answer the query, as shown in Fig. 7(b), it checks the type of the query (GUID, File, RDF, RDFS, OWL) and hands it over to the corresponding datatype handler which processes the query according to its type. Datatype handlers are defined and registered with the P-Grid library by the users of the library, i.e., the *Gridella* and *Semantics* components.

Semantic Gossiping. The introduction of RDF type queries enables Semantic Gossiping which is explicitly activated by the issuer of a query through a special flag in the query. Fig. 8 sketches how gossiping is implemented. In the figure,

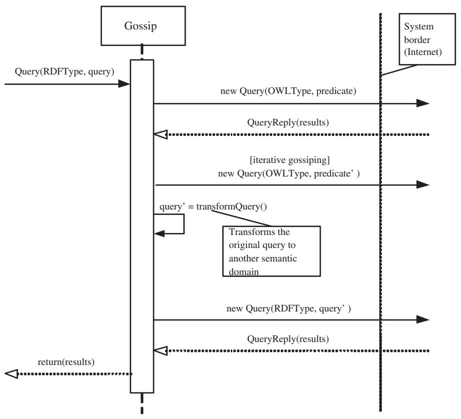

Gossip
System border (Internet)
Query(RDFType, query)
new Query(OWLType, predicate)
QueryReply(results)
[iterative gossiping]
new Query(OWLType, predicate')
query' = transformQuery()
Transforms the original query to another semantic domain
new Query(RDFType, query')
QueryReply(results)
return(results)

Fig. 8. Gossiping mechanism

we just show the relevant part and omit the preceding flow of control (incoming message – P-Grid – Datatype handler (Semantics)) for simplicity.

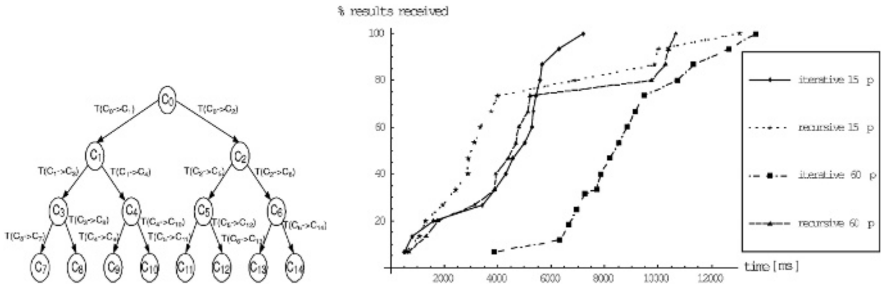
In resolving translation links we support two approaches: iterative and recursive. In iterative resolution, the peer issuing the RDF query tries to find and process all the translations links by itself; It first issues a query to retrieve the translation links capable of transforming the category used in the original query; Upon finding a translation, it translates the original query into a transformed query (Query') and issues a search for the transformed query. Furthermore, the gossiping peer issues a query for a translation of the translation (Predicate'). This continues until no more translation is available or the transformed query is considered as being too different from the original query following syntactic and semantic similarity measures [4].

In recursive resolution, the issuing peer tries to resolve the translation by delegating it rather than doing it itself: First, it looks for translations of the predicates used in the query and translates the query upon finding a translation. The transformed query is issued and results for the query are returned to the issuer of the query. The receiver of the transformed query follows the same procedure recursively.

6.2 Experimental Evaluation

We briefly discuss below an initial performance evaluation of the two Semantic Gossiping techniques GridVine implements. The tests were performed using the current implementation on a Fast Ethernet network of 60 SUN Ultra10 stations (Solaris 8). We first created 15 different semantic domains (i.e., 15 different categories C_0 to C_{15}) related to each other through 15 translation links as depicted in Fig. 9(a). We chose to organize the translations in a very regular way (i.e., a tree) in order to get a better grasp on the results obtained; Note however that our approach and implementation work equally well on more complex translation graphs (see also [3,4]).

We launched 15 peers, each on a separate computer and each locally storing a triple related to a different category. By issuing a query from the peer using C_0, we could retrieve all the 15 triples from the 15 semantic domains by forwarding the query through the translation link hierarchy. A second setting was created by replicating this first setting four times, running 60 peers using the same category setting (i.e., we had 4 peers per category each storing one triple locally).



(a) Evaluation setup (b) Semantic Gossiping, 15/60 peers

Fig. 9. Semantic gossiping evaluation, 15/60 peers, 15 translation links

The results, time elapsed versus quantity of results (up to 15/60 results) received by the peer issuing the query, for both settings and for iterative and recursive forwarding are displayed on Fig. 9(b). As expected, iterative forwarding works in a fairly linear manner. Also, note the initial delay incurred by letting one peer process and send all the queries for iterative forwarding with 60 peers. Our recursive approach proceeds more in stages, as it delegates the whole process of query forwarding to intermediary peers. This second approach proves to be particularly scalable with the number of peers: Results are rather independent of the number of peers or results returned, since the number of peers processing and forwarding the query increases with the network size.

7 Related Work

Hyperion [9] is an on-going project which proposes an architecture and outlines a set of challenges for decentralized data management in P2P systems. SWAP [12] is an approach combining P2P and Semantic Web techniques. It relies on an RDF(S) model and on structure extraction for handling queries in a P2P setting. Edutella [15] employs a super-peer topology and facilitates the clustering of data based on ontology, rule, or query. In PeerDB [17], each peer holds a set of locally available metadata (*Local Dictionary*) and a set of metadata that can be accessed by other nodes in the network (*Export Dictionary*). Metadata can be added through an SQL query facility. The system if based on BestPeer [16] which employs mobile agents to satisfy queries. No global schema is imposed but the process is not fully automated, as the user has to decide which mappings are actually meaningful. The Piazza peer data management project [22] takes an approach to semantic heterogeneity that is similar to Semantic Gossiping. Unlike our approach, Piazza does not provide any measures to judge the (in)correctness of mappings. The indexing is centralized and thus the scalability of the system is limited.

All the above approaches address semantic interoperability but offer limited scalability. Other approaches address scalability but do not deal with semantic interoperability. For example, Peer-to-Peer Information Exchange Retrieval (PIER) [14] is a database-style query engine built on top of a DHT. Its main focus is to provide database query processing facilities to widely distributed environments. One of PIER's restrictions is that it imposes global, standard schemas following the rational that some schemas will become de facto standards. RDFPeers [10] builds on the Multi-Attribute Addressable Network (MAAN), which extends Chord, to efficiently answer multi-attribute and range queries on RDF triples. RDFPeers is a scalable RDF store, but does not provide any schematic support (e.g., to handle user-defined schemas or to address semantic interoperability issues).

8 Conclusions

To the best of our knowledge, GridVine is the first semantic overlay network based on an scalable, efficient and totally decentralized access structure supporting the creation of local schemas while fostering global semantic interoperability. Following the principle of data independence, our approach separates the logical and physical aspects such that it can be generalized to any physical infrastructure that provides functionalities similar to our P-Grid P2P system.

References

1. K. Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Swiss Federal Institute of Technology, Lausanne (EPFL), 2002. http://www.p-grid.org/Papers/TR-IC-2002-79.pdf.
2. K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: A self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3), 2003.
3. K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. Start making sense: The Chatty Web approach for global semantic agreements. *Journal of Web Semantics*, 1(1), 2003.
4. K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. The Chatty Web: Emergent Semantics Through Gossiping. In *International World Wide Web Conference (WWW)*, 2003.
5. K. Aberer and M. Punceva. Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, 2003.

6. K. Aberer (ed.). Special issue on peer to peer data management. *ACM SIGMOD Record*, 32(3), 2003.
7. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(47), 2002.
8. A. Anaby-Tavor, A. Ga, and A. Trombetta. Evaluating matching Algorithms: the Monotonicity Principle. In *IJCAI Workshop on Information Integration on the Web*, 2003.
9. M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. *SIGMOD Record*, 32(3), 2003.
10. M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *International World Wide Web Conference (WWW)*, 2004.
11. A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.
12. M. Ehrig, P. Haase, R. Siebes, S. Staab, H. Stuckenschmidt, R. Studer, and C. Tempich. The SWAP Data and Metadata Model for Semantics-Based Peer-to-Peer Systems. In *Multiagent System Technologies (MATES)*, 2003.
13. J. M. Hellerstein. Toward network data indepence. *ACM SIGMOD Record*, 32(3), 2003.
14. R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Conference On Very Large Data Bases (VLDB)*, 2003.
15. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *International World Wide Web Conference (WWW)*, 2002.
16. W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A self-configurable peer-to-peer system. In *International Conference on Data Engineering (ICDE)*, 2003.
17. B. C. Ooi, Y. Shu, and K.-L. Tan. Relational Data Sharing in Peer-based Data Management Systems . *ACM SIGMOD Record*, 32(3), 2003.
18. C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 1997.
19. E. Prud'hommeaux and B. Grosof. Rdf query and rules: A framework and survey. http://www.w3.org/2001/11/13-RDF-Query-Rules/.
20. A. Seaborne. Rdql - a query language for rdf. http://www.w3.org/Submission/RDQL/.
21. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM Conference*, 2001.
22. I. Tatarinov, Z. Ives, J. Madhavan amd A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyaska, G. Miklau, and P. Mork. The Piazza Peer Data Management Project. *ACM SIGMOD Record*, 32(3), 2003.
23. World Wide Web Consortium. Semantic web activity at the w3c. http://www.w3.org/2001/sw/.