Towards Ontology-Driven P2P Grid Resource Discovery

Felix Heine, Matthias Hovestadt and Odej Kao
Paderborn Center for Parallel Computing - PC²
Universität Paderborn, Germany
{fh,maho,okao}@upb.de *

Abstract

In this paper we present a new approach to semantic resource discovery in the Grid. A peer-to-peer network is used to distribute and query the resource catalogue. Each peer can provide resource descriptions and background knowledge, and each peer can query the network for existing resources. We do not require a central ontology for resource description and matching. Each peer may have its own, possibly incomplete, ontology, which will be completed by the knowledge distributed over the network. This allows to find matching resources even if the concepts used to describe the resources are unknown to the provider, as the network supplies the missing parts of the ontology.

1 Introduction

Resource matching in large heterogenous Grid environments becomes increasingly difficult the more detailed a request is specified. Nowadays, the term resource is used in a very broad manner, including hardware resources, software packages, licences, Grid services etc. Thus, the specific attributes needed to describe these resources are getting more and more complex. In the same manner the requests are getting more differentiated.

As Grids are complex distributed systems there is a need for interoperability and mechanical reasoning. Therefore, the Semantic Grid initiative [4] aims at incorporating semantic technologies like ontologies and description logics into Grid middleware. Several approaches have been made to use ontologies in the matchmaking problem [3, 12, 5], but they use ontologies only locally and assume that the provider and requestor use the same background knowledge. This means that the resource broker must have access to a complete ontology of the domain in which it is acting.

As domain knowledge increases dynamically and decentralized, it will become impossible to ensure the up-to-dateness and consistency of the ontologies.

Therefore, an architecture is required where domain-specific knowledge can be acquired by a resource broker on the fly. This paper introduces an ontology-based peer-to-peer search network for Grid resources which addresses this issue. It is used to answer resource requests in a distributed Grid environment with additional semantics. The novel aspect of our approach is that the P2P network is used both to distribute the assertional and the conceptual knowledge. This means, the inherent incomplete ontology of any peer will be complemented by the knowledge of other peers. This allows the network to deduce answers to queries which could not have been answered by querying individual peers.

The following example illustrates the basic idea: A resource request states that a cluster system with at least 4 nodes is needed which runs the Linux operating system with kernel 2.4. A resource provider (peer A) in the network has a Linux system based on RedHat 9 with 4 nodes but does not understand the condition kernel 2.4. Another peer B knows the rule kernel 2.4 is implemented by RedHat 9 and has published it to the network. By combining these pieces of information from A and B, the network can deduce that peer A can provide matching resources for the query.

Within this paper we first give an overview about related work, followed by a brief introduction of description logics (DL) and distributed hash tables (DHT) in section 3. These two technologies are the foundation of our algorithm. In section 4 we describe the core algorithm for retrieving peers with matching resources. In the remainder of the paper we present simulation-based results and conclude after giving an outlook to our upcoming next steps.

2 Related Work

Research on ontologies has recently gained much attention due to the Semantic Web initiative started by Tim Berners-Lee [2]. Its focus is to exploit the power of semantic technologies to enable mechanical reasoning over
the web’s content. Therefore, the storage of ontologies had to be standardized in order to enable world-wide interoperability. This led to the creation of the Web Ontology Language (OWL) [13] which has just become a W3C recommendation.

Several researchers have realized that the central features of ontologies, namely interoperability and mechanical reasoning, are also applicable to other areas. The Grid community has started the Semantic Grid initiative to exploit the usage of semantics at different levels of the Grid [4]. This initiative has already led to some advances, especially in the field of resource description and resource matching. Brooke et al. [3] use ontologies in the GRIP project to describe and match resources in a heterogenous architecture comprising different Grid middleware packages like UNICORE and Globus. The emphasis of this work is on interoperability. Tangmunarunkit et al. [12] use ontologies in resource matching to gain the ability of mechanical reasoning, thus being able to match requests much more flexible than in former key-value based resource matching systems.

The issue of mechanical reasoning has been immersed by González-Castillo, Trastour, and Bartolini. In [5] they present their experiences on a prototypic implementation of a matchmaker using Semantic Web technology. However, they do not address distributed storage of information.

In contrast Löser is following a much broader approach in [9], where sources are classified in taxonomies and saved in a fully distributed system. It extends traditional distributed hash tables with hierarchical structures, so that the system is able to process hierarchical or conjunctive queries. He uses a Super-Peer network for load balancing and to prevent flooding queries over the whole system. However, Löser does not address semantic implications.

In [8] Li and Horrocks analyze the usability of DAML+OIL and DAML-S for service description. They prove the ability using a prototypic matchmaker which bases on a DL reasoner. By simulating an e-commerce scenario they evidenced that the matchmaking process can be executed efficiently, so that this technology seems to be able to be deployed in large e-commerce environments.

Traditional approaches to resource matching like Condor Class-Ads [10] lack the ability to do mechanical reasoning. Therefore, they are not directly comparable to our system.

3 Used Methods

As outlined in the first section, our goal is to realize a distributed deductive system which allows semantic resource matching. Research on topics of deductive systems and distributed peer-to-peer computing led to very promising results that we use as a foundation of our work. In the following, we introduce these topics, namely description logics and distributed hash tables.

3.1 Description Logics

There has been much research about description logics over the past decades [1]. The Semantic Web initiative has identified DL systems to be a good basis for their efforts to allow mechanical reasoning over the Web. This is made explicit by the second level of OWL, called OWL DL, which is designed to be used with description logics [13].

DL systems have evolved as knowledge representation systems, with a solid mathematical grounding by being founded on first order logic. However, in contrast to first order logic theorem provers, modern DL systems are efficient and useful in practical systems without losing the clear semantic foundation inherited from first order logic.

In common DL systems the knowledge is separated in two parts: The taxonomical box (T-Box) and the assertional box (A-Box). The T-Box stores conceptual knowledge about the world and can be compared to the schema of a database system. It is created by defining concepts, which are sets of individuals in the domain of interest, and roles which represent the relationships between individuals. For example, a concept could be PARENT, a role HAS–CHILD.

The A-Box represents the concrete knowledge about individuals within the domain. It consists of concept assertions and role assertions. For example, peter could be asserted to be a member of the role PARENT, and the pair (peter, mary) could be asserted to fill the role HAS–CHILD.

In the past, a lot of different kinds of DL systems have been evolved. Each of them supports some other expressiveness of concept- and role-forming expressions. For our system, we use RACER [6] which implements the SHIQ logic [7]. RACER has an interface to OWL. We use this interface to be able to process OWL documents. In the following, we will abstract from any concrete syntax to store ontologies and assume that the underlying system is able to handle the standard syntax. Also, we will not cope with uniqueness of identifiers, as we assume that each name is unique by the usage of namespaces. Thus, if two peers use the concept PROCESSOR, we assume they talk about the same concept.

A central feature of the T-Box part of every DL system is the calculation of the subsumption relation, which means to prove that some concept \( X \) is subsumed by some other concept \( Y \), i.e., every individual which is a member of \( X \) must also be a member of \( Y \), no matter which A-Box is taken.

From this subsumption relation the so-called classification graph is calculated, which is built by taking the concepts as vertices and the subsumption relation as directed edges. This graph is acyclic, as a cycle would mean that all concepts on the cycle are identical. Therefore, we refer to it as the classification DAG.
3.2 Distributed Hash Tables

Second generation P2P networks are based on distributed hash table (DHT) algorithms. These algorithms allow to locate a peer in a P2P network which holds some information for a specific key. They provide a single method: lookup(key), which returns the node responsible for the given key. There are different DHT algorithms, we use currently Chord [11], which allows to find the target node in $O(\log(n))$, where $n$ is the size of the network.

4 Algorithm for Distributed Resource Matching

This section describes the matching process in detail, focussing on semantic aspects of the process. A DHT based P2P network is used to distribute both the assertional and the conceptual knowledge. The RACER system is used as the deductive component of the system. Arbitrarily, any DL system supporting classification of concepts and instance checking can be used.

In this section, we explain the model we use to describe and match resources. Based on this model, we explain the problem of semantic query evaluation without network flooding in a P2P network. We show how information is distributed over the network, and how peers join and leave the network. In the last three subsections, we describe the evaluation of simple and complex queries and discuss the soundness and completeness of the algorithm.

4.1 Model

Each peer\footnote{We use the terms peer and node equivalently} runs a local description logic system and uses it to store its own resource catalogue. It describes the existence of certain resources by making A-Box assertions. Every resource is described as an individual, which is asserted to be member of one or more concepts. These concepts describe all aspects of the resource, more generic ones like resource type, processor type, and operating system, and also very specific aspects like system library versions, minor kernel differences, etc.

Therefore, it is impossible to force every peer to use the same T-Box. We allow each peer to have its individual T-Box. It can permanently add new concepts to describe new types of attributes of the resources. Nevertheless, we assume some generic concepts used in the different T-Boxes to overlap.

A cluster with Pentium processors running the Linux operating system can be described using the following RACER statements:

\begin{align*}
\text{(instance clstr1 CLUSTER)} \\
\text{(instance clstr1 PENTIUM)} \\
\text{(instance clstr1 LINUX)}
\end{align*}

The resource is named clstr1 and is described to be a CLUSTER having a PENTIUM processor and running the LINUX operating system. The background knowledge in the T-Box is used to classify the concepts CLUSTER, PENTIUM, and LINUX. For example, LINUX may be classified to be a sub-concept of UNIX, and SUSE9 may be described to be a sub-concept of both KERNEL2.4 and LINUX, saying that each resource which is an instance of SUSE9 is also an instance of KERNEL2.4 and LINUX.

We introduce an artificial top concept named RESOURCE which is defined to be a super-concept of every other concept in the classification used for the resource description. Therefore, each node which has at least one resource, no matter what kind of resource, will have an instance for the RESOURCE concept.

This means, that querying the resource descriptions of a peer is equivalent to the operation of instance checking for a certain concept. A resource request is described in terms of a concept definition. The concept definition includes all details the user has specified in his request. In the current version of our system, the concept definition for the request is restricted to concept forming expressions which do not use roles and only include conjunction, disjunction, or negation.

In contrast, the concept forming expressions for the resource descriptions are allowed to use arbitrary expressions. Thus, the expressiveness of the resource specifications depends directly on the expressiveness of the underlying description logic.

4.2 Problem Formulation

Using the model described in section 4.1 in a distributed environment brings up two issues:

- It is not known which peers to ask for matching resources. Without further information, the only way to get all matches is to flood the network by asking each peer to do instance checking for the concept definition which describes the resource query.
- Peers may not even know that they have matching resources, as described above, because they lack important parts from the domain knowledge.

In the following subsections, we describe a P2P network which addresses these issues. Both conceptual and assertional knowledge from all peers is distributed over the network. Queries can be evaluated using this knowledge which will find matching resources without asking each peer individually.
As an example, consider a network with three nodes. Each of these nodes has a different local ontology. The classification DAGs of these ontologies are shown in figures 1 through 3. The first node with IP address 1.1.2.1 classifies processors according to their vendors, the next according to their word-length. The third node just has some AMD specific knowledge. The first node has a resource with ID 7 of type \textsc{pentium}. This is indicated by 1.1.2.1 [7] in the \textsc{pentium} box in figure 1, saying that the resource with ID 7 from node 1.1.2.1 is an instance for this concept.

Our algorithm distributes these local classification DAGs over the P2P network. This results in a virtual classification DAG which is shown in figure 4. The network also propagates assertional knowledge about resources. This is shown by the 1.1.2.1 [7] in the \textsc{pentium}, \textsc{intel}, \textsc{32bit}, and \textsc{proc} boxes. They indicate that the resource with ID 7 from node 1.1.2.1 must be an instance for these concepts.

Now consider a query asking for a 32-bit processor from Intel. Although the first node has a matching resource, it will not be able to resolve the request on its own because it does not know about 32-bit processors. By querying the network, which resolves queries using the virtual classification DAG, resource 7 from node 1.1.2.1 is identified to match the resource request.

It is also noticeable that the integration of the DAG from the third node causes the removal of the redundant edges from \textsc{athlon32} and \textsc{athlon64} to \textsc{amd}.

Our goal is to provide a system which allows distributed instance checking in a peer-to-peer network where both the A-Box and the T-Box are distributed over the network. However, in contrast to DL reasoning procedures, we only aim to achieve soundness, but not completeness. As peers may constantly join and leave the network, no completeness can be guaranteed. Nevertheless, the algorithm should be able to find as many solutions as possible.

The concepts are distributed over the network by using the DHT algorithm. The name of the concept is handed to the DHT algorithm as a key to determine the node which has to store the information for this concept. For each concept, both A-Box and T-Box knowledge is stored. The stored T-Box part is a list of super-concepts. These lists, distributed over the network, form the virtual classification DAG. During the join of a node, there may be some redundant edges, in case a super-super-concept is stored for a concept. The peers will exchange messages to remove these edges.

The A-Box part is a list of resources from each peer which are instances for the concept, called the instance list. For each resource, the IP address of the node holding it and the ID of the resource itself on the node are stored. In the following, we call this list instance list and denote it by "IP\_1 [resources of IP\_1], IP\_2 [resources of IP\_2], ..., IP\_n [resources of IP\_n]."

This list is the larger, the more general a concept is. For the concept \textsc{resource} the list includes all resources from all peers. Therefore, an efficient storage and handling is elementary. During the processing of the instance lists, typical operations are union, intersection, and subtraction.

We store the list as a hash map. It is indexed by the IP address and stores a list of resource IDs as a bit-vector. Therefore, we require each node to generate a linear numbering of its resources. With this data structure, the instance lists are stored efficiently both with regard to memory consumption and required operations.
4.4 Peers joining the network

If a new node joins the network, it has to publish its local classification DAG to other nodes which are responsible for storing the appropriate information. The local classification DAG is generated by the underlying DL system. The node iterates through the DAG and identifies the nodes which need to be informed about new super-concepts or new instances. The information which has to be sent to each node is collected within a single message.

When a node receives such a message, it has to incorporate the new concepts, super-concepts, and instances into its own knowledge base. During this process new messages to other nodes may be generated in two cases:

- If the node gets to know a new instance of some concept, it must send messages to all nodes holding information about the super-concepts of this concept.
- If it gets to know a new super-concept of a given concept, it must send all instances it has stored for this concept to the node which stores the new super-concept.

A node always answers to a message by sending a message back which contains the list of super-concepts it knows for its own concepts. This answer is used by the original node to delete super-super-concepts, as it is only necessary to store direct super-concepts. As an example, consider a node which has stored AMD to be a super-concept of ATHLON32. Some other node publishes ATHLON to be a super-concept of ATHLON32 and AMD to be a super-concept of ATHLON. The result can be seen in figure 5. Peer 1 is responsible for storing the ATHLON32 concept. It has stored both AMD and ATHLON as super-concepts of ATHLON32, while peer 3 has stored that AMD is a super-concept of ATHLON.

When peer 1 receives the message that AMD is a super-concept of ATHLON during step 1, it can remove AMD from the list of super-concepts for ATHLON32. The flow of messages is illustrated in figures 5 and 6, which show the first two iterations. Both the propagation of nodes having instances for the concepts and the removal of unnecessary super-concepts can be seen. Information that a node has newly received are typed in boldface. These parts need to be sent out. The instance lists are shown as bit-vectors.

In step 1, peer 1 sends out information about the concepts AMD and ATHLON. They contain the instance list for the ATHLON32 concept. The concept has instances at the nodes 1.1.3.7 (resource IDs 0, 1, 2, 4, 6, 7) and 1.1.3.9 (resource IDs 0, 4, 7). These messages are sent to peer 2 and peer 3, which are holding the information about AMD and ATHLON, respectively.

Concurrently, peer 3 sends a message to peer 2 about instances for the concept AMD at node 1.1.3.9 (resource IDs 1, 2, and 3). All messages are answered with lists of super-concepts for the given concept. Peer 1 thereby learns that AMD is a super-concept for ATHLON, therefore it can remove AMD from the list of super-concepts for ATHLON32. The resulting status and the next messages are shown in figure 6. Peer 2 has stored the instances for the AMD concept.

In step 2, peers 2 and 3 have new information and need to send out messages. Only the message from peer 3 to peer 2 is shown. The one from peer 2 reaches another node storing the concept PROC. It is not shown here.

In the last step (not shown here), the network has reached a stable status. As peer 2 only received information which had already been stored no further information is to be distributed. Thus, no more communication is needed.
4.5 Peers leaving the network

The ontology distributed over the network will be growing as more and more peers join and publish their local knowledge. Therefore, if a peer leaves the network, its related knowledge has to be removed from the network.

In this paper we assume that peers do not leave the network accidentally. They have to inform the network that they are about to leave and send out some final messages.

Currently, only the A-Box knowledge is removed. This is done in the same way in which it is published. The leaving peer sends out messages to every node which is responsible for the concepts it has instances for. This information is propagated to each super-concept in order to remove the instance information stored at the super-concepts.

The T-Box knowledge is not removed, because every ontology which has been published to the network is considered to be correct and thus to be beneficial to the behavior of the whole system. It is subject to further refinements to implement data provenance in order to be able to remove certain parts of the ontology.

4.6 Querying for Simple Concepts

The crucial question is: how to select the nodes that might have instances for the concept the requestor is looking for? First, we look at the sub-problem of determining a node set for a simple concept. In the next step, we use this basic functionality to provide querying of concepts which are built by conjunction, disjunction, and negation of simple concepts.

By distributing knowledge across the network as described in section 4.3 we need to determine the node which is responsible for the concept being queried by the hash-value of the concept name. We directly request this node for a list of nodes having instances for this concept. As the instances are propagated to the super-concepts, this procedure will identify nodes even if the target node does not know the concept which has been queried.

In the second step, direct negotiation with each of the target nodes is performed in order to question the availability of the resources.

4.7 Querying for Complex Concepts

A complex query is formulated in terms of a concept definition, based on conjunction, disjunction, and negation of simple concepts. The instance list is built up successively by asking the network for an instance list for every simple concept which occurs in the concept forming expression of the complex concept. The concept forming expression is parsed and the instance list is build bottom-up by applying union or intersection, where concepts are formed by disjunction or conjunction, respectively.

Negation is handled in two different ways. If negation appears in combination with conjunction, for example \(A \land \neg B\), the instance lists are subtracted. If negation appears alone or in combination with disjunction, like in \(A \lor \neg B\), the instance list \(B\) is subtracted from the instance list for the concept \(RESOURCE\), which is the set of all instances from all nodes. This means that such negations are inefficient, as the instance list of \(RESOURCE\) will be the set of all instances, no matter what kind of resource. However, a query like: “give me all resources which do not run the Linux operating system” is not likely to appear. It does not restrict the type of the resource at all, leading to a result set which will not only include cluster resources, as probably intended, but also every type of resource which does not run any operating system, e.g., license resources.

4.8 Soundness and Completeness

The soundness of the system is obvious by construction. As we propagate instances along super-concept relations, each generated instance is guaranteed to be an instance of the concept which is being queried. This holds true also during the join of a new node, as we only add correct information and the network is never in an intermediate state with incorrect values.

The system is not complete in two cases. It is obviously not complete during the process of distribution of the information. However, as shown in section 5, it does not take a large number of iterations until the final state is reached.

The other problem concerns missing information for sub-concepts. As we only consider super-concept relations in the network, we will not be able to find matching instances of sub-concepts which are not known to the peers which provide these resources.

For example consider a concept \(X\) defined as a sub-concept of \(Y\) containing all instances of \(Y\) which have a filler for the role \(R\). A node which knows only of concept \(Y\) will not be able to deduce that some instance it holds for concept \(Y\) might also be an instance for concept \(X\). Therefore, information about instances has to be propagated to sub-concepts also. However, this destroys the soundness, as instance lists with too many instances are generated. In this case, the final answer can only be produced by direct negotiation with the target node for each instance.

The crucial point about this discussion is that the node-lists need to be both as small as possible in order to minimize the direct contact with nodes, and as large as needed to contain every node which has in fact some instances for the query.
5 Results

To get an insight view into the performance of our algorithm, we simulated several network scenarios. We were interested in the following questions:

- How many iterations does it take until new information is fully distributed over the network? That means, how long does it take until the answer to a query will be correct with respect to the new information which has been published?

- How many messages are sent across the network during the publishing process?

The simulation generates a random classification DAG, which is, again randomly, distributed over the nodes. Thus each node holds a subgraph of the whole DAG. Each node is asserted to have some instances for the concepts of the random DAG. Starting with this setting, the distribution of the information over the network is simulated, thereby counting the iterations and messages needed to finish the process.

The interesting parameters to modify are the number of nodes, the size of the DAG (i.e., the number of concepts in the ontology), and the number of instances. We ran simulations with different DAGs, distributing each DAG over more and more nodes. Thereby we increased the overall number of instances, while we kept the number of instances per node fixed. The results are shown in figure 7 and 8. The number of nodes is shown on the X axis, which corresponds to the number of instances (we have taken 10 instances per node). The number of iterations and messages are shown on the Y axis of figure 7 and 8, respectively. Note that the X axis is not linear.

We have run each simulation 10 times, in order to avoid special effects due to the random generation of the DAGs. The values in the diagrams are average values from the different runs. Thus, the number of iterations may not be integer.

Figure 7 shows that the number of iterations does not grow significantly as the number of nodes grows. This can be explained due to the fact that the number of iterations only depends on the size of the longest path in the classification DAG along which the instances have to be propagated. The small variations in the number of iterations is explained by the random distribution of the instances, which has to be re-calculated each time the node count is increased.

As the number of concepts grows, the number of iterations only grows slightly. With 552 concepts, we have an average of 9.3 iterations, with 49923 concepts the average is 15.3 iterations. This is a factor of 1.6 in the number of iterations compared to a factor of 90.4 in the number of concepts. As the concepts are organized as a DAG, and only the longest path is relevant for the number of iterations, this becomes clear.

Now look at figure 8. The number of messages grows moderately compared to number of nodes. The average number of messages per node decreases as the number of nodes increases, e.g., from 29.8 to 0.3 messages per node in case of 552 concepts, and from 426.2 to 31.4 in case of 49923 nodes.

We expect the number of messages to have an upper limit for a fixed DAG which is reached in a situation with a huge number of nodes where each node only stores one concept. If there are fewer nodes and thus more concepts per node, some messages get bundled because they are sent to the same node, and thus the number of messages decreases. The figures confirm this assumption.

As an example, consider a small network with 1000 nodes and 552 concepts. Here, the number of messages per node is 2.9, compared to 31.4 at a network with 10000 nodes and 49923 concepts, which is a factor of \(\sim 10\) in nodes and \(\sim 100\) in concepts, compared to a factor of \(\sim 10\) in the number of messages per node.
6 Future Work

This paper describes first steps towards a system for distributed resource matching based on peer-to-peer technology and semantics. Currently, the system is restricted in several ways. We are working on the elimination of these limitations. In the following, we summarize these issues:

- **Completeness**: The improvement of the completeness by incorporating information regarding sub-concepts is important to generate more resource matches as currently possible.

- **Expressiveness of queries**: We are interested in improving the expressiveness of the query language aiming on providing the full functionality of the underlying DL system, including concrete domain attributes and sophisticated role expressions [1].

- **Fault tolerance**: When a peer leaves the network, we assume that the peer will leave cooperatively by informing the other peers that he is about to leave. In a real world scenario, peers will break down due to hardware failures, network failures, etc. Redundant storage of the knowledge is needed to cope with this issue.

- **Garbage collection**: In this paper, we assume that all conceptual knowledge which is distributed over the network, will reside there forever. We need some kind of garbage collection to cope with this problem.

- **Ranking results**: We are interested in a mechanism to rank the matches a priori without questioning each peer individually. For example, it would be useful to share some knowledge about the current availability of resources in order to avoid asking for a resource which is currently booked out or otherwise unavailable.

- **Routing optimization**: Currently, the routing of messages relies completely on the underlying DHT network. However, a node always communicates with the same nodes as long as the set of super-concepts stays the same. Thus, caching the addresses for these nodes is a good idea to avoid repetitive lookups of these nodes via the DHT algorithm.

7 Conclusion

In this paper we presented a novel approach for distributed resource matching in the Grid. Our system uses ontologies based on description logics to describe the resources. It distributes the information over a peer-to-peer network based on distributed hash tables. We outlined the core algorithm and depicted how it enables the system to detect resource matches, even if a provider within the Grid does not know all terms used in a resource query. This has been achieved by combining the knowledge of all peers within a distributed classification DAG, so that queries can be resolved against this DAG.

Simulations were used to gain insight into the behavior of the system while distributing new information over the network. The simulation results reveal that the system scales well even for a large number of concepts, nodes, and resources.

Based on these promising results, we believe that the system presented here will be beneficial for resource matching in future large heterogeneous Grid environments.

References


