Towards semantically-enhanced distributed service discovery

Raphael Romeikat, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Romeikat, Raphael, and Bernhard Bauer. 2007. "Towards semantically-enhanced distributed service discovery." In *Second International Conference on Internet and Web Applications and Services (ICIW'07), 13-19 May 2007, Morne, Mauritius*, edited by Deapesh Misra, Stefania Galizia, Christian Emig, Axel Martens, Dumitru Roman, Andreas Wombacher, Danny Hughes, et al., 26. Los Alamitos, Calif.: IEEE. https://doi.org/10.1109/iciw.2007.65.

Towards Semantically-Enhanced Distributed Service Discovery

Raphael Romeikat, Bernhard Bauer
Institute of Computer Science
University of Augsburg
Augsburg, Germany
{romeikat|bauer}@informatik.uni-augsburg.de

Abstract—**In this paper, we present a new approach for service discovery combining semantic web and peer-to-peer techniques. A reference ontology is used to describe and discover services in our approach. We do not need a central point of control at any time. All information required for service description and discovery is completely distributed across the nodes of a peer-to-peer overlay network. We describe the design of a Semantically-enhanced Distributed Discovery System (SDDS) allowing dynamic and efficient registration and discovery of services. We also present performance analysis and discuss open issues of our system.**

Keywords-service; discovery; peer-to-peer; ontology

I. INTRODUCTION

Services computing has become a strategic research area of information technology. Its scope covers the whole lifecycle of services including service creation, service deployment, service discovery and service composition, to mention just a few aspects. Amongst those, service discovery is an important field as before a service can be invoked, it has to be located first. In the meantime, services have also found their way into Grid technologies. The Open Grid Services Architecture (OGSA) [1] represents an evolution towards a Grid architecture based on web service concepts and technologies.

Common service description languages such as the Web Service Description Language (WSDL) offer a way to describe abstract and technical functionalities of a service. They do not include semantic information, so two services can have totally different intentions although having similar or even the same descriptions. Regarding automated service discovery and composition, more than syntactical and technical information is required. This is where semantic techniques come into play. The formal meaning of syntactical data is usually specified by adding meta-data. In that area, OWL-S [2] and SAWSDL [3] are recent techniques, which help annotating services with semantic information.

The present scheme of service discovery is based on directories designed in a centralized or hierarchical way, just as Universal Discovery, Description and Integration (UDDI) [4] or the Globus Monitoring and Discovery System (MDS) [5]. Due to their design, such systems have two shortcomings. Firstly, there is a performance issue as all communication flows through a single component, which may become a bottleneck when a lot of inquiries have to be processed at the same time. Secondly, any centralized system represents a single point of failure. This might be evaded using redundant data on multiple servers, but the fundamental issue still persists. With a large number of services and a large number of participants in a system, distributed service discovery is the way to be preferred.

In this paper, we present a new approach for service discovery called Semantically-enhanced Distributed Discovery System (SDDS). Our approach avoids any bottleneck or single point of failure and makes use of semantic information for service discovery at the same time. The design of SDDS is based on a structured peer-to-peer (P2P) overlay network, where knowledge about the services is completely distributed. All information necessary is self-organized in a structured Chord ring. Every node within that ring represents a service provider or a service requestor or both at the same time. As a service may comprise several operations, we actually refer to service operations rather than to services. We annotate input and output parameters of service operations on the basis of a reference ontology. Service operations are described and discovered on the basis of these annotations. We do realize that semantic service descriptions actually include more details such as preconditions and effects. For the time being, we focus on input and output parameters as a first step; further aspects are to be added later.

There have been some approaches for service discovery which try to avoid a centralized architecture and make use of semantic information when describing and discovering services. An approach which uses semantic descriptions of services combined with a P2P network topology is described in [6]. A drawback of this method is a lot of communication overhead due to the unstructured underlying architecture. Another approach is undertaken by METEOR-S providing a solution based on a P2P network of UDDI registries including semantic annotations, as described in [7] and [8]. However, the architecture envisioned in METEOR-S suffers from a single point of failure as there is only one single entry point to enter the network of registries. The approach described in [9] is similar to our one, but uses an unstructured P2P network. By contrast, we specially target a structured network architecture in order to enable efficient service discovery.

The rest of this paper is organized as follows. In chapter II, we give an introduction to ontologies and P2P systems. In chapter III, we describe the design of SDDS. Finally, open issues are discussed in chapter IV of this paper, and a summary is given in chapter V.

II. TECHNICAL BACKGROUND

A. Ontologies

An ontology is a data model representing concepts of a certain domain and relations between them, whereas concepts are called classes and relations are called properties. By the use of ontologies, knowledge of a certain domain can be shared and reused. There has been a lot of research about ontologies due to the Semantic Web initiative started by Tim Berners-Lee [10]. Ontologies are a powerful technology enabling interoperability and mechanical reasoning over web content. In the meantime, specification of ontologies has been standardized by the W3C consortium by introducing the Web Ontology Language (OWL) [11].

We now regard a simplified and slightly modified version of W3C's ontology example about wines. There is a class called *Wine* with two object properties describing the wine's color and maker, represented by the two classes *WineColor* and *Winery*. The latter one is equivalent to a class *Vineyard*, which itself is derived from a more general class *Producer*. Fig. 1 shows a graphical representation of this example.

Trying to find out the producer of a wine, one might search for a service operation offering *Wine* as input and *Producer* as output parameter. Probably, no service offers an operation with exactly the signature desired. This will be the case if there are two services, *ServiceA* offering *OperationA* with *Wine* as input and *Winery* as output parameter, *ServiceB* offering *OperationB* again with *Wine* as input but *WineColor* as output parameter, for example. Surely, *ServiceB* is not applicable for finding out a vine's producer. However, it is a different matter with *ServiceA*. Although *OperationA* returns *Winery* instead of *Producer*, the result does satisfy the initial request. Regarding the semantic information represented by the ontology, one can see that *Winery* is equivalent to *Vineyard* and *Vineyard* is a subclass of *Producer*, so *Vinery* is a specialization of *Producer*. It is the task of our Semantically-enhanced Distributed Discovery System to perform semantic conclusions concerning equivalence and subclass properties within an ontology as described in the wine example above.

B. P2P Systems

We now summarize peer-to-peer systems, as the architecture of our discovery system is based on recent work in that area. According to Foster and Iajnitchi, peer-to-peer systems are decentralized, self-organizing, distributed systems, in which all or most communication is symmetric [12]. Peer-to-peer systems usually involve a large number of participants, also called nodes or peers. Such networks are useful for a lot of purposes. Sharing content files containing audio, video or other digital data has become very common, but applications using P2P technology for passing real-time data such as telephony traffic have also gained attraction.

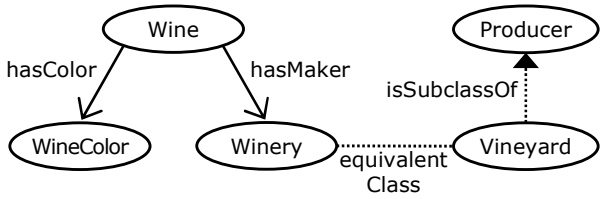

Figure 1. Simple ontology about wines

Starting in the nineties, several peer-to-peer based file sharing systems have been developed. One of the first systems was Napster, designed especially for sharing MP3 files amongst participants [13]. Although Napster became very popular, it implied a big drawback as it used a central index server for locating users' files. The approach had limited scalability and contained a single point of failure, which made it quite easy for jurists to shutdown Napster after several legal proceedings. Gnutella is another early file sharing system [14]. In contrast to Napster, there is no central authority to organize the Gnutella network. Instead, each node is connected to a couple of other nodes resulting in an unstructured overlay network. The structure of the network is highly dynamic as nodes constantly join or leave; most nodes remain in the network for less than 24 hours. Gnutella uses a flooding-based approach to route queries. In order to avoid flooding the whole network, a time-to-live field and the number of hops along the routing path are used to limit further routing at a certain point. Gnutella's highly distributed design eliminates a single point of failure. However, the approach makes search results indeterministic and does not guarantee that a file desired can be reached at all, even if it exists at some distant node of the network.

In order to overcome the shortcomings of early P2P systems concerning resource discovery, distributed hash table (DHT) approaches were introduced such as Pastry [15] or Chord [16]. They construct structured overlay networks with all nodes having equal roles and responsibilities. Different routing algorithms are utilized in order to forward messages purposively instead of flooding the whole network. Thus, benefits of distribution are preserved while efficient retrieval of objects is guaranteed, resulting in good scalability, fault tolerance and low maintenance cost. Search results are correct and complete. Correctness implies that only relevant objects are found; completeness implies that all relevant objects are discovered. A key technique used to achieve these goals is the fact that each node only coordinates with a few other neighbor nodes, typically $O(\log n)$ in an n-node network. When nodes join or leave the network, only a limited amount of work is necessary to keep the overlay structure. Such systems can finish a search operation in $O(\log n)$ hops using $O(\log n)$ messages, depending on the exact routing algorithm used.

III. THE DESIGN OF SDDS

In this chapter, we describe the design of our Semantically-enhanced Distributed Discovery System. As mentioned previously, we use a reference ontology to annotate parameters of service operations. Each input and output parameter is annotated with a class of the reference ontology. All semantic annotations are stored decentralized in a Chord ring. One may search for services at any peer by specifying one or more classes as input or output parameters or both at the same time. In our approach, the reference ontology is known to each service provider and requestor. Like that, consistent usage of terms is achieved. At the moment, the reference ontology is used in a static way, so its structure and classes need to be known at deployment time. We do not focus on a certain language to specify service descriptions and semantic annotations as our approach is a conceptual one.

A. Chord

The design of SDDS is based on the structured Chord overlay network, which we describe now. In Chord, objects are designated by a key, which may be a filename or any other distinctive qualifier. Each node and each key is assigned a unique m-bit identifier out of a circular identifier space containing 2^m identifiers. A node's identifier is obtained by hashing its IP address and port number; a key's identifier is obtained by hashing the key's value itself. All nodes self-organize into a ring topology in ascending order based on their node identifiers in the circular space. Keys are assigned to nodes using consistent hashing; i.e. key K is assigned to the first node whose identifier is equal to or greater than the identifier of K. This node is responsible for the resource with key K and is called its successor node, signified by successor(K). Please note that, due to the ring topology, comparisons and calculations concerning identifiers are performed modulo 2^m. Chord's consistent hashing mechanism offers two benefits. Firstly, hash values created are evenly spread with high probability. Secondly, only an $O(1/n)$ fraction of all keys are reassigned in case the responsible node leaves the n-node network [17]. However, it is important to set m to a value large enough in order to ensure that hash values of all identifiers are disjoint. To simplify matters, we use the terms node and key for their identifiers respectively.

Each Chord node basically maintains a set of successors, called finger nodes. The finger nodes represent a node's routing information and are spaced exponentially around the identifier space. The i-th finger of node N is the first node that succeeds N by at least 2^{i-1} on the identifier circle, where $1 \leq i \leq m$. The first finger node ($i = 1$) is the immediate successor of N. Thus, each node maintains at most m finger nodes in a so-called finger table with size $O(\log n)$. Chord improves fault tolerance and efficiency by additionally maintaining the predecessor and a constant number of successors for each node. Fig. 2 shows a Chord ring consisting of eight nodes storing three keys; the finger nodes and finger table of node 38 are shown exemplarily.

In order to lookup an object with key K, a node N will route a lookup request to successor(K) as the latter one is the node responsible for K. Based on its finger table, N forwards the request to the finger node whose identifier most immediately precedes K. By repeating this process, the request gets closer and closer to successor(K). Finally, successor(K) receives the lookup request for K and returns the respective object back to node N. As finger nodes are spaced exponentially around the identifier space, each hop clockwise from one node to the next one covers at least half of the distance between N and successor(K). This is why a lookup requires $O(\log n)$ routing hops in an n-node network.

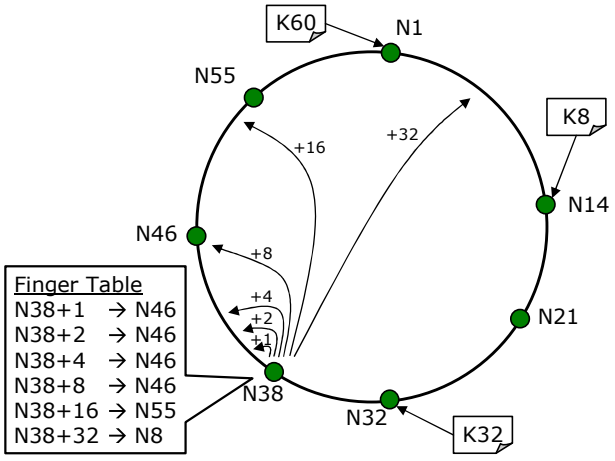

Figure 2. A Chord ring consisting of 8 nodes storing 3 keys

As nodes may join and leave the network, some effort is necessary to maintain the ring topology. For this purpose, each Chord node periodically runs a stabilization protocol in the background ensuring each node's successor pointer is up to date. Basically, each node N asks for the predecessor of its immediate successor N' at regular intervals. N itself will usually be the response of such a query, of course. In case a new node N'' joins the network between N and N', N will realize the change at the next run of its stabilization protocol. Now, successor and predecessor pointers are updated and all keys K with $N < K \leq N''$ are reassigned from N' to N''. There are two scenarios for a node leaving the network. If a node leaves the network on purpose, it will first reassign all keys and objects it is responsible for to its successor and inform its successor and predecessor about being neighbors from now on. The breakdown of a node is handled by the stabilization protocol which again updates predecessor and successor pointers. However, Chord does not provide fault tolerance for the objects stored on a broken node; this data may be lost when a node fails.

B. Ontology Managers

The underlying DHT mechanism of the Chord system offers distributed lookup based on exact matches for a given key. However, Chord does not support several types of queries that are desirable in the field of services computing such as multi-attribute queries or queries allowing for richer data structures. Therefore, we extend the Chord system by putting an additional management layer on top. This layer consists of one Ontology Manager (OM) per peer and undertakes tasks the basic Chord system cannot cope with. The Ontology Managers collaboratively perform service discovery and deal with registration of new service operations as well as deregistration of existing ones at runtime. Appropriate interfaces are provided by each OM. Fig. 3 shows the basic system model of SDDS.

The discovery mechanism of SDDS is not limited to exact-match queries for input and output parameters as equivalence and subclass properties within the reference ontology are taken into account as well, as mentioned in chapter II.A. In order to properly handle them, we need to be aware which classes can be substituted by other ones when searching for service operations. A class can obviously be substituted by another one

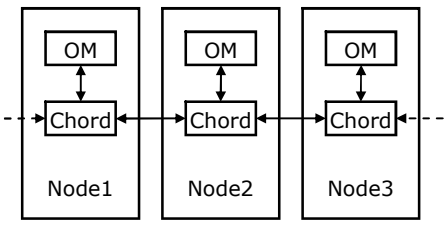

Figure 3. Basic system model of SDDS

expressly specified as equivalent class, no matter if it is about input or output. However, classes can be substituted by subclasses or superclasses as well, depending if an input or output parameter is desired. In the following, we use a set *input* containing desired input parameters of the search and a set *output* for output parameters respectively. Assume a search with $input = \{C\}$, which means we are looking for service operations accepting C as input parameter. A service operation offering a superclass C' of C as input also satisfies the query, because C' is more general than C and thus can be passed an object of type C just as well. In object orientated programming, this principle of substitution is known as contravariance. With output parameters, it is just the other way round. Given a search with $output = \{C\}$, another service operation offering a subclass C'' of C as output also satisfies the query, because a more special object is returned, which is even more information than requested. This principle is also known as covariance. For these purposes, we introduce the concept of substitutional sets. $subst_I(C)$ contains classes which can substitute an input parameter C; $subst_O(C)$ contains classes which can substitute an output parameter C. Formal definitions of $subst_I$ and $subst_O$ are given in (1) to (6).

$$C \in subst_I(C) \quad (1)$$
$$C' \text{ equivalent to } C \wedge C' \in subst_I(C) \Rightarrow C' \in subst_I(C) \quad (2)$$
$$C'' \text{ superclass of } C' \wedge C' \in subst_I(C) \Rightarrow C'' \in subst_I(C) \quad (3)$$

$$C \in subst_O(C) \quad (4)$$
$$C'' \text{ equivalent to } C' \wedge C' \in subst_O(C) \Rightarrow C' \in subst_O(C) \quad (5)$$
$$C'' \text{ subclass of } C' \wedge C' \in subst_O(C) \Rightarrow C'' \in subst_O(C) \quad (6)$$

Let us once more regard the example given in chapter II.A, where a search for $input = \{Wine\}$ and $output = \{Producer\}$ is performed. Instead of only searching for *Wine* and *Producer* respectively, all classes in $subst_I(Wine)$ as possible input and all classes in $subst_O(Producer)$ as possible output parameters are considered. *OperationA* of *ServiceA* is a satisfying answer to the query as its input parameter *Wine* is included in $subst_I(Wine) = \{Wine\}$ and its return parameter *Winery* is included in $subst_O(Producer) = \{Producer, Vineyard, Winery\}$.

We now have a closer look onto how the Ontology Managers perform service discovery. There are several steps performed collaboratively by the peers. First of all, a query for more than one parameter is split up into subqueries and each parameter is processed separately. In our example, there would be a subquery for *Wine* as input and another one for *Producer* as output. For each input and output parameter desired, the respective substitutional set is determined next, resulting in $subst_I(Wine)$ and $subst_O(Producer)$. For each substitutional set, service operations amongst all services are determined that have a parameter matching with a class in the substitutional set considered. Finally, the results of the subqueries are collected and analyzed in order to filter those service operations that require at most the input parameters provided and return at least the output parameters requested.

Two major tasks are to be solved in order to realize this approach in a distributed and efficient way. For both tasks, the Ontology Managers take advantage of the underlying Chord system. Firstly, the substitutional set is determined for each parameter requested. As the reference ontology remains static in our approach, we only calculate the substitutional sets for each class of the reference ontology once at deployment time and store them for later usage. Storage is performed in a distributed way using Chord's consistent hashing mechanism. The keys to be hashed are the names of the classes; the objects to be stored are the substitutional sets. Like that, each class C of the reference ontology is assigned to a certain node that is responsible for C and stores $subst_I(C)$ and $subst_O(C)$. This structure is also called inverted vertically partitioned index [18]. Such an index minimizes the cost of searches by ensuring that no more than m nodes are responsible for answering a query containing m classes. Like that, it takes $O(\log n)$ hops to determine a substitutional set of one class in an n-node network. Fig. 4 shows how $subst_I(Producer)$ and $subst_O(Producer)$ are stored in a Chord ring, assuming consistent hashing assigns the class Producer to node 1.

Secondly, starting from a substitutional set for a certain parameter requested, matching service operations are determined. Depending on the reference ontology's size, the substutional sets can become rather large, so processing each class within a substutional set separately would not be a good idea. Therefore, we store mappings of whole substitutional sets onto service operations. Substitutional sets are not disjoint and parameters usually occur in several substitutional sets, so this approach requires more space for storage. However, this is a good tradeoff as a lot of efficiency is gained for discovery. The mappings described are again stored in a distributed way using Chord's consistent hashing mechanism. The keys to be hashed are the substitutional sets; the objects to be stored are references to appropriate service operations. Like that, each substitutional set *subst* is assigned to a certain node that is responsible for *subst* and able to return references to appropriate service operations. Please note that it is not necessary to distinguish between $subst_I$ and $subst_O$ at this point; now the only task is to match the substitutional sets onto input and output parameters of service operations. We also refer to such matches as input and output matches, $match_I$ and $match_O$. Equations (7) and (8) give formal definitions of $match_I$ and $match_O$, where $params_I(Op)$ are the input and $params_O(Op)$ are the output parameters of operation *Op*.

$$match_I(S) = \{Op \mid \exists\, P_I \in params_I(Op)\,.\, P_I \in S\} \quad (7)$$
$$match_O(S) = \{Op \mid \exists\, P_O \in params_O(Op)\,.\, P_O \in S\} \quad (8)$$

Given a substitutional set S, it takes $O(\log n)$ hops to determine input matches $match_I(S)$ and output matches $match_O(S)$, n being the number of nodes in the network. Fig. 5 shows how input and output matches of the set $\{Producer, Vineyard, Winery\}$ are stored at a certain node of a Chord ring, assuming consistent hashing assigns S to node 32. We refer to an operation *Op* of a service S as *S.Op*.

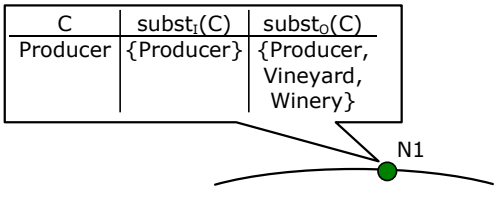

Figure 4. Storage of substitutional sets in a Chord ring

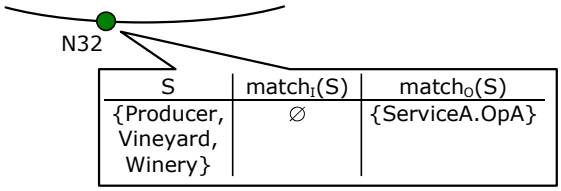

Figure 5. Mapping of substitutional sets onto service operations in a Chord ring

After determining input and output matches for each parameter, the resulting service operations are filtered in such a way that only those operations remain which require at most the input classes provided and which return at least the output classes desired. In other words, an operation is allowed to have more output parameters than requested but must not have more input parameters than specified. The result of the overall discovery process is formally defined in (9) to (11). A discovery for service operations on the basis of m classes in an n-node network takes $O(m \log n)$ hops altogether, which lets the system scale well, even if the number of nodes becomes very high. Fig. 6 shows the steps that are performed in the example about wines when searching operations with *Wine* as input and *Producer* as output parameter, as described in chapter II.A.

$$result_I = \{Op \mid \forall P_I \in params_I(Op)\ \exists C_I \in input\ .\ P_I \in subst_I(C_I)\} \quad (9)$$

$$result_O = \{Op \mid \forall C_O \in output\ \exists P_O \in params_O(Op)\ .\ P_O \in subst_O(C_O)\} \quad (10)$$

$$result = result_I \cap result_O \quad (11)$$

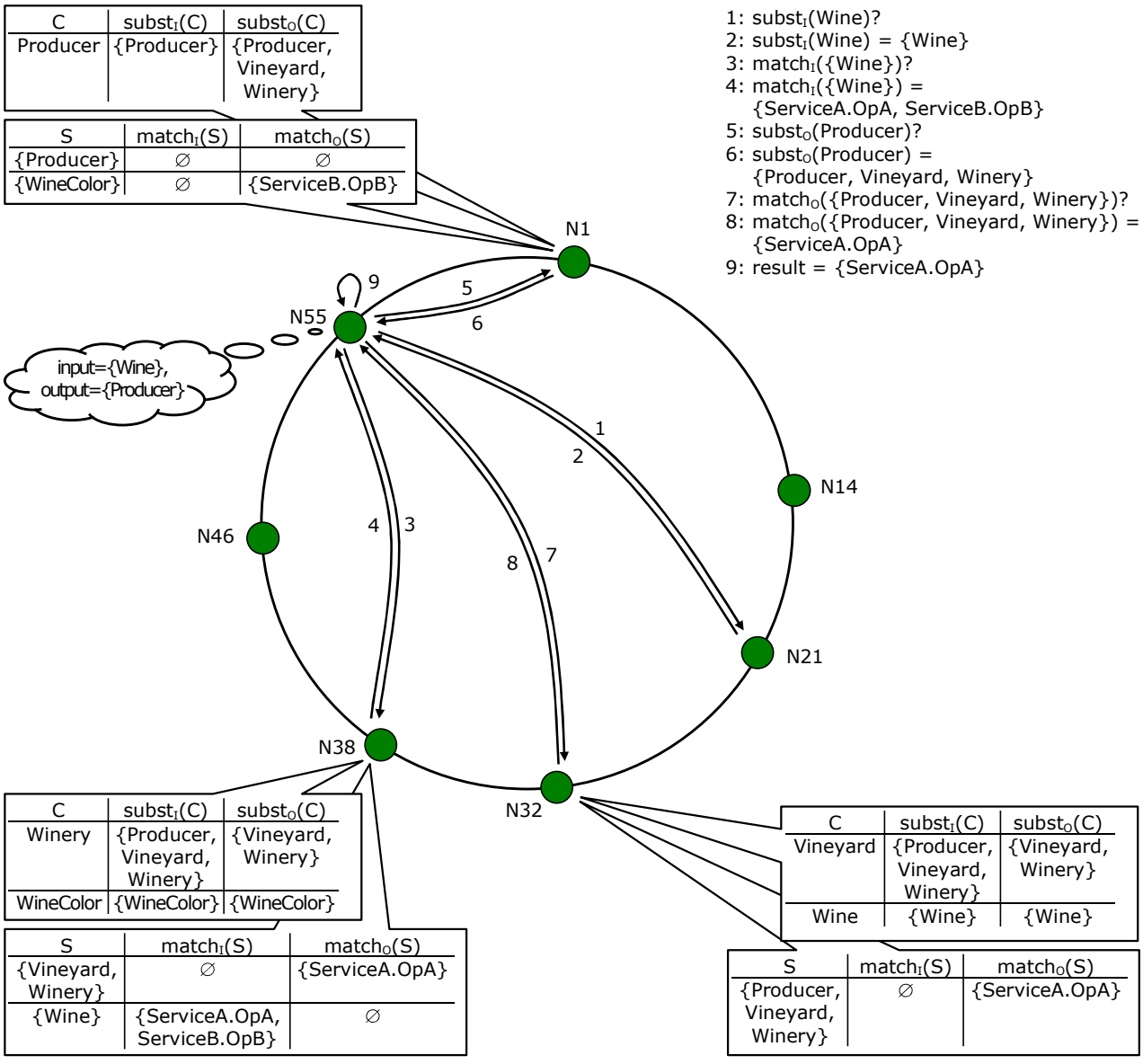

Figure 6. Discovery mechanism on the basis of the wine example

New service operations may be registered at runtime by any peer, and existent ones may be unregistered as well. In both cases, the substitutional sets are not affected as the reference ontology remains the same. However, input and output matches need to be updated. If a new service operation is registered, an input match will be added wherever a substitutional set contains a class corresponding to an input parameter of the operation; output matches will be added for output parameters respectively. With the design presented so far, all nodes would have to be checked, so this would take O($m\,n$) hops for an operation containing m parameters in an n-node network. In order to improve efficiency, additional knowledge is stored telling which substitutional sets contain a certain class. Like that, the number of hops required can be reduced to O($m \log n$). Deregistration of existent service operations is performed analogously.

IV. Summary and Open Issues

In this chapter, we summarize a couple of important characteristics of SDDS before discussing open issues. First of all, our system works completely decentralized avoiding any central component that could become a bottleneck or single point of failure. All nodes have equal roles and tasks. At the same time, good scalability is achieved as the number of nodes participating in the network may change without bounds. Even with a very large number of nodes, discovery is performed efficiently due to logarithmic number of hops necessary compared to the number of nodes in the network. Furthermore, our discovery mechanism guarantees correctness and completeness of search results. Node failures do not impact the structure of the system as they are resolved by Chord's stabilization protocol. Thus, SDDS offers attractive properties of self-organization.

There are some open issues to be discussed. Firstly, our approach assumes that service operations are fully specified. However, it is worth to think about partly specifications and smooth queries based on incomplete information. As mentioned before, the structure of the system is resistant to node failures. However, information stored at a failed node may be lost. An appropriate replication mechanism would be a desirable feature. Concerning the ontology involved, only classes and hierarchical relations between them have been regarded so far. More details need to be considered for fully-fledged service discovery, such as properties, preconditions and effects. Furthermore, our system is designed on the basis of a static reference ontology. It would be desirable to allow each peer having its own, possibly incomplete ontology, which is completed by knowledge distributed over the network, as proposed in [19], for example. Besides that, security issues have not yet been addressed. In some environments, information about service location could be considered sensitive, so communication would require authentication and authorization. Next, we will create a prototype implementation of our system and perform measurements about performance and scalability on the basis of a significant test series.

V. Conclusions

In this paper, we presented a new approach for service discovery in a distributed and semantically-enhanced way. Our system uses a reference ontology to annotate input and output parameters of service operations. All information is completely distributed over a structured peer-to-peer overlay network. We illustrated the core algorithm and explained how it enables our system to discover the service operations desired, even if there is no service operation whose signature exactly matches the parameters desired. This has been achieved by mapping the structure of the reference ontology and the signatures of the service operations onto a Chord ring. Furthermore, we presented performance analysis of our system and showed that the number of hops required to answer a query is logarithmical compared to the number of nodes in the network, offering good scalability for large networks. Based on our results, we think the ideas presented in this paper will be useful for service discovery in future services computing environments.

References

[1] I. Foster, H. Kishimoto, A Savva et al, "The Open Grid Services Architecture, Version 1.5," GFD.80, Open Grid Forum, July 2006

[2] D. Martin et al, "OWL-S: Semantic markup for web services," W3C Member Submission, November 2004

[3] J. Farrell, H. Lausen, "Semantic annotations for WSDL," W3C Working Draft, September 2006

[4] L. Clement, A. Hately, C. v. Riegen, T. Rogers, "UDDI Version 3.0.2," OASIS Technical Committee Draft, October 2004

[5] The Globus Alliance, "GT Information Services: Monitoring & Discovery System (MDS)," http://www.globus.org/toolkit/mds

[6] M. Paolucci, K. P. Sycara, T. Nishimura and N. Srinivasan, "Using DAML-S for P2P discovery," Proceedings of the 1st International Conference of Web Services, Las Vegas, June 2003, pp. 203-207

[7] A. A. Patil, S. A. Oundhakar, A. P. Sheth and K. Verma, "METEROR-S Web Service Annotation Framework," Proceedings of the 13th International World Wide Web Conference, New York, May 2004, pp. 553-562

[8] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller, "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services," Journal of Information Technology and Management, January 2005

[9] F. Banaei-Kashani, C.-C. Chen and C. Shahabi, "WSPDS: Web Services Peer-to-Peer Discovery Service," Proceedings of the 5th International Conference on Internet Computing, Las Vegas, June 2004, pp. 733-743

[10] T. Berners-Lee, J. Hendler and O. Lassila, "The semantic web," Scientific American, May 2001, pp. 34-43.

[11] M. K. Smith, C. Welty, D. L. McGuinness, "OWL Web Ontology Language Guide," W3C Recommencation, February 2004

[12] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of Peer-to-Peer and Grid Computing," 2nd International Workshop on Peer-to-Peer Systems, Berkely, February 2003

[13] Napster, LLC, "Napster," http://www.napster.com

[14] OSMB, LLC, "Gnutella.com," http://www.gnutella.com

[15] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems," Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, November 2001, pp. 329-350

[16] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," Proceedings of the ACM SIGCOMM Conference, San Diego, August 2001, pp. 149-160

[17] K. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," Proceedings of the 29th Annual ACM Symposium on Theory of Computing, May 1997, pp. 654-663

[18] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," Proceedings of the 4th International Middleware Conference, Rio de Janero, June 2003, pp. 21-40

[19] F. Heine, M. Hovestadt and Odej Kao, "Towards ontology-driven p2p grid resource discovery," Proceedings of the 5th International Workshop on Grid Computing, Pittsburgh, November 2004, pp. 76-83