A Proactive approach to Semantically Oriented Service Discovery

Dave Lynch, John Keeney,
Dave Lewis, Declan O'Sullivan
Knowledge and Data Engineering
Group
Trinity College Dublin
+353 1 6081765

ABSTRACT

This paper proposes a proactive approach to web service discovery which contrasts the passive approach exhibited by UDDI. The paper describes how a content based network implementation (Siena) has been extended to undertake matching based on ontological reasoning, resulting in a flexible knowledge-based delivery mechanism. In particular, it describes how this implementation has been used to support the proactive and potentially more efficient delivery of advertised web service profiles to users interested in services of those types.

Keywords

Content Based Networks, Ontologies, Semantic Web Services, Publish / Subscribe, Service Discovery.

1. INTRODUCTION

Web Service based computing has evolved immensely in recent years, supported forcefully by standards bodies, such as OASIS [18] and the W3C [29], and industrial forces, such as Microsoft, IBM and Hewlett-Packard, and a plethora of academic research. Web Services, by definition, are self-contained, self-describing applications that can be published, located and invoked remotely and in a dynamic fashion over the Internet [4]. This loosely-coupled remote-service invocation capability has proven to be a particularly attractive proposition for e-commerce and business integration models.

One of the noticeable omissions from the web services architecture described by the above bodies is a set of standards to support automatic discovery, automatic composition and invocation of web services. In their current form WSDL service descriptions and UDDI searches must be created or conducted with human intervention. A very desirable scenario is one whereby software agents can intelligently reason over required web-service functionality, automatically discover supporting modules and seamlessly integrate them on-the-fly into desired applications.

To realise such automation, the fundamental absence of semantic information to complement the syntactic provisions of WSDL must be remedied. Indeed, this is the vision of creators of the Semantic Web [3]. Their goals are to better define web semantics in machine-terms so that intelligent agents may feasibly reason over Internet and Web concepts thus enhancing user experiences of relevant information and interesting functionality.

In working to achieve this goal the Resource Description Framework or RDF [24] has paved the way for more complex application-specific semantic annotation standards such as the Ontology Web Language OWL [19], and an Upper Ontology for Web Services description OWL-S [10]. OWL-S semantically annotated descriptions are a major step towards enabling automatic invocation of services. The rise of OWL capable reasoners such as Pellet [17], OWLJessKB [15] and RACER [25] has further accelerated the progress.

While the UDDI approach outlines a method for describing *how web services function* there is a fundamental lack of support for describing *the web services' capabilities* in machine-understandable terms. While work has been done in adding semantic capability to UDDI [26][27], the focus has been on modifying the UDDI registry to accommodate OWL-S descriptions. Any modification of the UDDI in its current form still inherits poor support for large, loosely integrated wide-area registries of services.

The Publish/Subscribe (Pub/Sub) model for communication also lies firmly in the scope of loosely-coupled, large-scale distributed systems [11]. In the domain of large scale loosely coupled distributed systems, Pub/Sub has emerged as one of the more promising communications models. The model consists of three basic elements; *Subscribers*, who express interest in particular information by means of a *subscription language*, *publishers* of *information*, who publish information of interest and an intermediary *event notification service* connecting the two.

The selective *pushing* of service information towards interested subscribers using a Pub/Sub model may be viewed as an interesting alternative to explicit one-shot client-server based information retrieval such as that supported by the UDDI standard. It is argued here and henceforth that there is potential to integrate such a model into a web services discovery application. Focusing primarily on service discovery, this paper outlines an approach to web service discover which has the potential to be a more scalable, truly wide area alternative to UDDI registries using Content-Based Networking (CBN), an extension of Pub/Sub that supports the subscription matching of untyped messages based only on their contents.

It is the opinion of the authors that discovery for web services fits naturally into the publish/subscribe paradigm. This is particularly the case for pre-existing service compositions where the composition is based on choosing between predefined service types, such as those found in support of the value chain of an organisation. Here users have designed and tested service compositions that meet their particular needs but would like to be actively informed of changes to particular service definitions or

new candidate service profiles that may have advantages over a particular element of the designed composition. For example, there is increasing pressure to establish more transient ah-hoc relationships in organisational value chains whereby dynamic decisions can be made to, for instance, exchange one partner with a more competitive alternative. Currently, there are no standards or implementations that propose to proactively push web service descriptions towards interested parties, such as autonomous software agents. This paper introduces a distributed publish/subscribe based service discovery platform. This design and implementation is realised by focusing on enhancing the semantic capability of an existing CBN system, integrating a semantic web capability matching component and increasing the expressiveness that implementations subscription language to cater for semantically enhanced service requirements matching.

Furthermore, since semantic capability, in the form of OWL based ontological subscription support, has been added to an existing Pub/Sub based CBN system, it is our intention to exploit the annotated semantic information in service descriptions to assist in their content-based routing. We show that semantic information available from web service descriptions combined with semantics derived from a semantically-enabled subscription language, can help optimise the process of subscription matching and information routing within our proposed publish/subscribe platform for web service discovery. This can be achieved by modifying the structure of an existing subscription storage algorithm [6] preserving optimisations that arise upon reasoning over semantically enhanced subscriptions presented to our publish/subscribe system.

2. DESIGN

The design presented here is based upon the Siena CBN [6] due to source code availability and an abundance of associated technical reports and papers, and in addition, its focus on expressiveness in a wide-area distributed environment. Figure 1 illustrates conceptually the components of the design. These components are divided explicitly into three facets of concern; subscription, publication and matching. Within these there are requirements for a subscription set structure, a communications layer, ontology integration and ontology alignment.

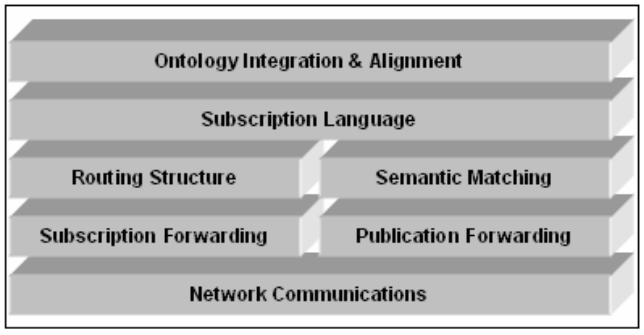

Figure 1: Conceptual Design Overview

The original implementation of Siena has no notification support beyond the simple types discussed in [6] (string, integer, bytes, boolean). Similarly, no support for OWL-S matching, or OWL programmatic support structure existed. In order to realise the goals of a semantically aware implementation the following enhancements were necessary.

- Modification of the Siena subscription matching algorithm.
- Extension of the Siena subscription language.
- Incorporation of OWL and OWL-S matching components.

2.1 Siena Subscriptions and Notifications

A Siena notification is a set of typed attributes. Each attribute is a triple consisting of a name, type and a value, where the type is limited to one of *string*, *integer, bytes, boolean, double*. A filter is constructed from a set of constraints which are each applied to the content of notifications. A constraint is a triple, consisting of the attribute name, a constraint operator, and a value. Where multiple constraints exist in a single filter they are evaluated as a conjunction. A filter **covers** a notification or event if that event satisfies each constraint applied to it by the content filter. An event or notification ***n*** is delivered to an interested party ***X*** if ***X*** has submitted a subscription filter that covers the notification. Also, a filter ***f*** **covers** another filter ***f'*** where together the set of constraints in ***f*** are more general than all of the individual constraints in ***f'***, and so all of the notifications that would be delivered or forwarded for ***f'*** would also be delivered or forwarded for ***f***, i.e. ***f*** is more general than ***f'***.

Each node in the hierarchical topology may have any number of incoming connections, other than clients, but only one outgoing connection to its parent node. Conceptually, the nodes have a client server relationship. Thus, a hierarchical node need only propagate information it receives to its parent node in the form of root subscriptions and publications. The main routing principle behind Siena is to push notifications as close as possible to parties that may be interested in that information. Known as downstream replication, this can be achieved both by subscription forwarding and advertisement forwarding. Subscription forwarding is the method used for routing in the Siena hierarchical implementation.

In the current implementation of Siena, notification routers are arranged in a hierarchy of nodes, where each node maintains a tree structure that keeps track of subscriptions and so informs the notification forwarding strategy for that node. In this tree structure general subscriptions are at the top and more specific covered subscriptions are arranged as subtrees. The tree of subscriptions is used to assist in pruning the number of subscriptions forwarded and therefore maintains scalability. Essentially, root subscriptions are the only ones sent. As such, subscriptions covered by previously forwarded subscriptions are pruned and network traffic is kept to a minimum. In order to ensure consistent notification across the network, Siena employs publication forwarding to master nodes, and leaves further notification beyond that of root subscriptions to the nodes on which the more specific subscriptions reside.

When the Siena node acting as the server to a notification producer ***X*** receives a subscription filter ***f*** from ***X***, the subscription tree is searched starting at each root subscription. If a subscription is found that covers the filter ***f*** and contains ***X*** in its subscriber set the search terminates. Otherwise, if the filter ***f*** already exists in the subscription tree, ***X*** is simply placed in the subscriber set of that particular filter. Finally, should neither of these apply a new subscription is inserted under the most specific covering filter, possibly a leaf node, with ***X*** added to its subscriber set. If no covering filter exists, the subscription is inserted as a root subscription. All root subscriptions are forwarded to master nodes right to the top of the Siena node hierarchy, with sub nodes acting exactly like subscribers.

Upon reception of notifications at a Siena router node (either from the notification producer or a super-node) the set of clients or other sub-nodes with subscription filters covering the notification are sent that notification, such that only a single message is propagated downstream towards subscribing clients and routers. If the master server was not the source of the notification then a copy of this notification is also sent to the master server. In fact, the relationship between a Siena node and its master is very similar to that of a subscriber client and the Siena node itself. The net effect of this is that no matter where a publication, or subscription, takes place on the network the correct subscriber subset is notified in a scalable manner.

2.2 Extending the Siena Subscription Language

One of the primary contributions of the design of this implementation is to enhance the Siena subscription language. The main change to the subscription language was the addition of three new ontological operators: *Subsumes*, *Subsumed by*, and *Equivalent*. The subsumption relationship describes how an ontological entity is more general than another ontological entity.

The ⊩ operator is used to express subsumption, the ⊫ operator is used to express the inverse subsumption, and the ≡ operator is used to express ontological equivalence. For example, as seen in an example class hierarchy in Figure 2, the ontological type **article** subsumes the type **research**, or **research** is subsumed by **article** since **article** is less specific than **research**. Equivalence refers to the relationship between two ontological types that refer to the same type of entity yet may be different ontological classes. On the left hand side of Figure 2, the subsumption operator is used to express the constraint that this particular subscription is interested in having the concept **publication** as an input. An example of the inverse subsumption operator is also shown where the subscriber has expressed an interest in the **article** concept or concepts that subsume this. There is an implicit and relation between each of the expressed constraints specified upon the same input. In the diagram this is illustrated by a broken line box within the subscription. As a result, this constraint is a complex one whereby the subscriber is interested in concepts that lie between publication and article in the class concept hierarchy defined in blue. Since the concept of a **book** is on the same level of the hierarchy as the **article** concept, an expression of interest in inputs of type **book** is implicit. A subscriber may not wish to register interest in concepts related by subsumption and simply request a concept equivalent to that in which interest is expressed. The

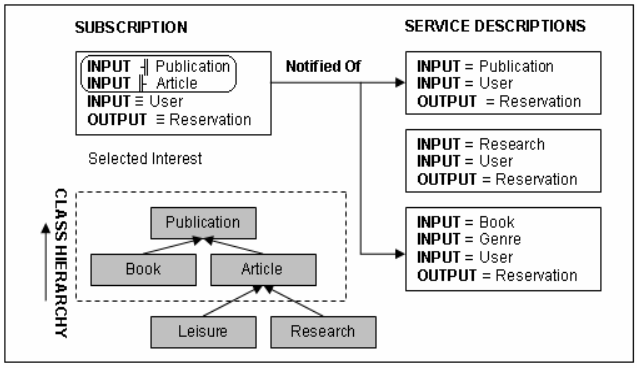

Figure 2: Publication Example

remainder inputs and outputs utilise the ≡ operator and an implicit or-relation exists between each of these constraints and the previous complex constraint. The enhanced constraint triple consists of and input or output, one of the operators described above, and the name of an OWL class.

While this example may seem to make the subscription specification more difficult for simple subscriptions, the advantages become apparent for more extensive ontologies. In addition, since the standard subscription language for Siena, and most content-based networking systems, allow filters to be defined using base data types, and only as a conjunction of filters (i.e. filter constraints are combined using the Boolean AND operator and so the failure of one constraint in a filter results a match failure for that filter), the specification of flexible subscriptions using ontological classes would entail the specification of multiple individual subscriptions to match for each class type specified as a string comparison, with no inbuilt consideration for class equivalences.

Depending on the makeup of particular ontologies; e.g. size, complexity, purpose; a more or less functional reasoner may be required to obtain a correct class hierarchy. For this reason it is necessary to carefully tune the specific level of reasoning required to each specific ontology on an application by application and a case by case basis. Further information on the comparative performance of a number of reasoners is available from [22]. Tests have been carried out with a number of reasoners including OWLJessKB [15], Pellet [6] and the reasoners bundled with Jena [5], with the reasoners deployed in a pluggable and exchangeable manner. A more detailed discussion of Siena primitives can be found in [6].

2.3 Maintaining the Siena Subscription Tree

While remaining at an abstract level it is necessary to discuss enhancements and modifications to the Siena subscription tree structure and subscription forwarding architecture at the design stage. The main consideration behind enabling ontology based subscriptions in such a manner is the preservation of the covering relation between filters. In particular, the partial ordering between subscriptions within the subscription tree structure must be maintained to allow subscriptions to cover each other as the number of subscriptions grows, thereby maintaining the inherent scalability of the Siena CBN. In order to accomplish this we must define a covering relation between our enhanced subscriptions.

Consider two filtering constraints **A** and **B**, such that **A** is given as (**x** op **a**), and **B** is given by (**x** op **b**), where op is one of ≡ (equivalent to), ⊫ (more specific than, or is subsumed by), or ⊩ (less specific than, or subsumes). The variable **x** is the variable for the service input or output in each notification to be compared to the ontology classes **a** or **b**, given in the filter specification. Table 1 describes when filter constraint **A** covers filter constraint **B**, i.e., when the set of possible notifications matching filter constraint **A** is a superset of the set of notifications matching filter constraint **B**. In this design it should be noted that the subsumption and reverse subsumption relationships between two service input or output classes do not hold if they are equivalent, i.e. if class **a** is equivalent to class **b**, then **a** is not more or less general than **b**.

A number of observations can be drawn from Table 1 that may not be immediately obvious. Lines 1, 5 and 9 show that a constraint does not cover itself or an equivalent constraint. This is to avoid the situation where **A** covers **B** and **B** covers **A**, which would lead to circular references and infinite looping in the

optimisation of a node's subscription tree. It should also be noted that ($x \dashv\Vdash y$) is equivalent to ($y \Vdash\!\dashv x$). For any filter **f** with multiple filtering constraints combined as a conjunction, **f** is covered by **f'** only if all of the filtering constraints in **f** are covered by filtering constraints in **f'**. The covering relationships for the other Siena operators are given in [6][9], and remain completely unchanged by the addition of the three new operators described here.

Table 1: Covering relationships between new Siena ontological operators

A Covers	*B*	*iff*	
$x \equiv a$	$x \equiv b$	*never*	1
$x \dashv\Vdash a$	$x \equiv b$	*if* ($a \Vdash\!\dashv b$)	2
$x \Vdash\!\dashv a$	$x \equiv b$	*if* ($a \dashv\Vdash b$)	3
$x \equiv a$	$x \dashv\Vdash b$	*never*	4
$x \dashv\Vdash a$	$x \dashv\Vdash b$	*if* ($a \Vdash\!\dashv b$)	5
$x \Vdash\!\dashv a$	$x \dashv\Vdash b$	*never*	6
$x \equiv a$	$x \Vdash\!\dashv b$	*never*	7
$x \dashv\Vdash a$	$x \Vdash\!\dashv b$	*never*	8
$x \Vdash\!\dashv a$	$x \Vdash\!\dashv b$	*if* ($a \dashv\Vdash b$)	9

2.4 OWL-S Matching component

As discussed in Section 2.1, we have extended Siena to enable ontological matching. This we believe provides a platform of general utility that can be used in a number of different domains, and ongoing research in parallel (see section 8) is reaffirming our belief. However, we also believe that for some kinds of applications, specialised ontologically based matchers will be more efficient. In order to explore this, it was decided to replace the general matcher described in section 2.2 with a matcher specifically designed for undertaking ontologically based service matching.

The OWL-S Matching component is based on the OWL-S Matcher Java implementation developed by Technischen Universitat of Berlin (OWLSM) [20]. The OWL-S Matcher uses JESS with the OWLJessKB knowledge based scripting engine for OWL concept reasoning [15]. The OWL-S Matcher was enhanced to use SAX for XML parsing for this project.

The process of matching in the OWL-S Matching component can be broken down into four distinct phases: input matching, output matching, service category matching and user constraint matching, each of which scores a numerical ranking, also based on the subsumption relation. The semantic matcher then aggregates a ranking in each of these categories and as a result can produce an accurate match with informative matching statistics.

The OWL-S matcher was used for two major purposes. The matcher component had a subsumption based reasoning sub-layer already implemented and therefore this reasoning sub-layer was used establish subsumption relationships between inputs and outputs of services and inputs and outputs of subscriptions. As well as this sub-layer a basic OWL-S service matching component was already implemented within the matcher. A support package representing subscriptions was implemented and the basic subsumption implementation was built upon. The logic for evaluating the subsumption relationship between simple constraints, complex constraints and subscriptions was implemented at each level of granularity. These relationships were necessary to preserve the partially-ordered nature of the Siena routing and matching mechanism while extending its capability to handle more complex data types than those already supported by the Siena routing mechanism.

3. PROACTIVE SERVICE DISCOVERY

At the most abstract level the exchange that takes place in the system to enable proactive service discovery takes the following form:

1. A Subscriber presents its request in terms of desired service profile to the Siena server.
2. The Siena server registers the service request and the location of the subscriber.
3. The Siena server is presented with an OWL-S Service by a publisher.
4. The Siena server parses this service extracting the service profiles.
5. These profiles are subsequently matched against the subscription/subscriber set and a notification list generated.
6. Each subscriber is notified of the publication of a new OWL-S service that matches the service requirements previously expressed.

Service Profile activity

The subscription mechanism starts with the examination of ontologies used in the set of service descriptions and subscriptions. Should an ontology be unfamiliar to the system the ontology is first integrated into the nodes' global knowledge base. The subscription is then examined, if it is a root subscription it is inserted into the Siena subscriptions data structure and the subscriptions forwarded to the server node. Otherwise the subscription is inserted into the tree, the subscriber is mapped and a record of its location is maintained.

Server Publication activity

Once the service profile has been presented and parsed the accompanying ontologies are integrated into the knowledge base. Should matching service requirement subscriptions be found in the subscription tree, those subscribers set are sent a copy of the published service profile.

Client Subscription activity

Firstly a client notification hander is initialised. Constraints are specified using the enhanced subscription language, where the set of ontologies and mappings used to specify concepts are provided, and the subscription is sent to a Siena server. Once the service profiles have been received back in response to the subscription, it is to the responsibility of the client to handle integration and invocation of the service.

4. Example Usage

An English expression of the subscription shown in Figure 2 follows:

> *This subscriber wishes to receive notification of the publication or modification of any service profile with at least two inputs and one output. One of these inputs must, conceptually, be a publication, article or book. This subscriber wishes to receive confirmation of*

reservation by receipt of reservation information or an equivalent concept and as a result requests that this be an output of the desired service.

The right hand side of figure 2 shows three sample service profiles that have been published to the Siena server. The first of these is covered by the subscription since we have satisfaction for each constraint placed on the content of the service profile. The second of these profiles fails on the first constraint, since the concept of research article is too specific in terms of the class hierarchy shown. The failure of one constraint results in the failure of the match as a whole. The third service profile illustrates an interesting application of the covering semantics used in the Siena content based routing system. Each constraint has been satisfied correctly and therefore the subscriber is notified of the existence of a matching subscription. An important observation is that the input concept **Genre** is also required of the published service profile. The omission of the **Genre** input parameter may be interpreted as an expression of not caring what other inputs exist on the service. It is assumed in this case that the registered subscriber agent is capable of reasoning over this input requirement in order to provide enough information to invoke the service successfully. However, the omission may also express disinterest in any other inputs or outputs of any time on behalf of the subscriber, perhaps since the subscribing agent is not capable of handling them. Since it is infeasible to express disinterest in every unsupported concept in a large scale system, e.g. by means of a Boolean NOT operator and since the inclusion of service inputs and outputs numbers decreases greatly the expressiveness of the notification in this scenario, from henceforth it is assumed that an automatic agent is capable of handling parameters it has not specifically expressed interest in. In this way the covering semantics of the content based routing scheme are preserved more effectively.

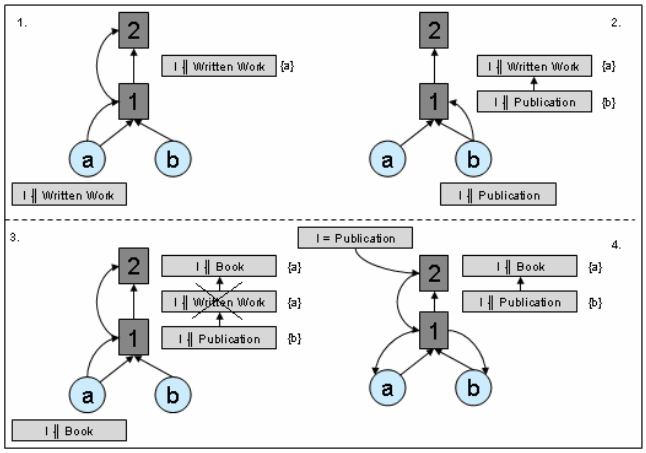

Figure 3: Example of subscription forwarding

Figure 3 illustrates subscription forwarding in a sample scenario. At stage 1 subscriber **a** registers interest in the concept of written work or less specific. Since this is a root subscription in the subscriber set for node **1** it is forwarded to node **2** where it handled in the same way as a subscription from a client. In the second illustration client **b** registers interest in the concept of a publication. This is covered by the previous subscription and arranged in the subscription tree structure accordingly. Note that this subscription is not forwarded to the master node. Illustration 3 sees **a** register a more general subscription and the subsequent covering sees the removal of the redundant subscription for **a**. When the publication takes place to master node **2** the root subscriptions of **1** are also present in **2** therefore the publication is forwarded to node **1** where matching takes place as usual and clients **a** and **b** are notified correctly. Only sending root subscriptions in such a manner keeps network cost low however the trade off between duplicated matching through nodes and distributed storage of publications is a subject for careful evaluation.

5. Evaluation

Figures 4 and 5 illustrate our two evaluation scenarios. Figure 4 shows a single node Siena set-up and figure 5 shows the distributed model. The effect of publishing to both the centralised and the hierarchical model should be exactly the same. Each node was hosted on a Dell D400 notebook with JDK 1.5, an Intel M 1.2Ghz processor and 256MB of RAM with minimum resident programs.

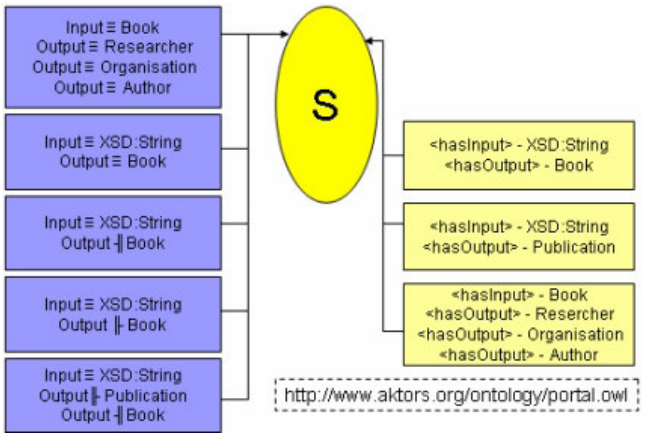

Figure 4: Single Siena Node Test Scenario

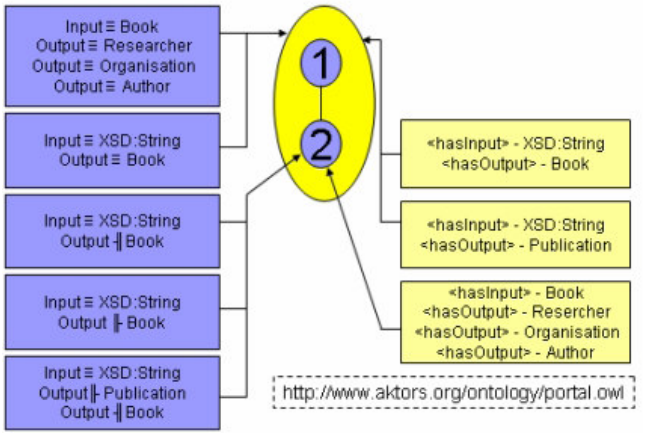

Figure 5: Two Siena Nodes Test Scenario

The average matching time for the most complex subscription over 10 separate runs was 23ms, with the lowest 10ms (least complex) and the longest 28ms. This match time includes the period of traversal through the subscription tree structure and, possibly multiple, calls to the reasoner for the capability match. Although a loose comparison, a covering match on a 5 subscription original Siena implementation (text-based) takes on average 8ms. This comparison can only be considered indicative of the more complex nature of a capability match and a full, larger scale study is needed to confirm if or how this matching time scales linearly. Initial indications, i.e. by inspection, indicate that

capability matching does not hugely increase the amount of processing required to match across the subscription tree structure. This can be considered an encouraging result.

The most interesting performance bottleneck occurs when we consider ontology integration time. Integration of OWL ontology, and parsing of an OWL-S service seems a very expensive process. In our sample implementation integration of a standard OWL ontology, (the *aktors* ontology) [2], takes on average 1.2 seconds and requires a 98kb download. On the surface of this measurement it seems that ontology integration is definitely a bottleneck when it comes to analysis of matching performance in this system. When the system is scaled to thousands of interested subscribers over thousands of services this inefficiency has the potential to overload servers.

Another interesting observation is the time it takes to parse and load an OWL-S ontology. On average, our simple book finder service, devoid of pre-conditions, effects, complex assertions and conditional executions, takes 12.5s to parse and load into the knowledge base from a server running on localhost. It is speculated that one of the reasons why this takes so long is the insistence of the OWL-S reasoner in loading, parsing and integrating all ontologies that are imported and referenced by the service itself. In this case, the *aktors* ontology was loaded and parsed despite already being asserted in the knowledge base. A similar scenario occurs when we consider the OWL-S specification and the XSD specification. This metric would indicate that the system as a whole would benefit from a finer level of OWL parsing and loading granularity.

We plan further work to test and benchmark a number of other ontology frameworks and ontology reasoners. We expect these experiments to lead to greatly reduced runtime costs and ontology loading and parsing times.

In conclusion, the tests confirm the correct function of the OWL-S enhanced wide area notification service in terms of OWL-S matching, enhanced Subscription Language and subscription and publication distribution.

However, performance evaluation conducted was not on a large enough scale to draw valid statistical conclusions regarding increases in network load and network cost as well as CPU cost, however initial tests indicate that parsing and integration of Web-Services and associated ontologies is a definite issue to be considered in future implementations. Further evaluation of how the incorporation of semantic concerns affects loosely coupled pub/sub knowledge distribution system is ongoing work for the authors. Preliminary evaluation of the performance of different ontology reasoners for use in message routing and subscription matching is given in [13][14][16]. Ongoing research is also investigating and simulating factors that affect the scalability and efficiency of the system when deployed in larger scale environment.

6. Related Work

Although heavily supported by languages such as OWL, OWL-S and RDF as well as SOAP and XML, research into semantic service discovery is still maturing and as a result a standard means of discovery is still a way off. As a result of this non-convergence research continues in several parallel avenues outlined below.

Although not a semantically enhanced standard, the recently agreed UDDI specification version 3 [27] presents some interesting additions. Most notably, in the context of this paper, a Subscription API has been added. Not explicitly a publish/subscribe application, at least in the distributed event notification sense, the movement of the standard towards a publish/subscribe concept is an interesting one. Since subscribers must express explicitly the requirements, in terms of the selected category, such as tModel, it is observed here that the UDDI subscription API more closely resembles a topic-based subscription. Aside from the lack of UDDI support for semantic concepts, more expressive textual content based subscriptions would offer more expressiveness of subscription language and hence more accurate notifications in terms of web service discovery. It could however, also be argued that since automatic discovery and invocation is not supported by UDDI web services that expressiveness and accuracy of category may not necessarily be as important as in a semantically enhanced version.

There is a very active body of research in semantically enhancing the UDDI registry standard. Since the UDDI standard is plentiful in features and a mature standard, it seems a logical progression to attempt to build on this maturity by adding semantic annotation. In [1] the authors endeavour to provide a structure whereby semantic information may be annotated onto current UDDI elements, such as tModel. Similarly [23] endeavours to "import" the semantic web into a UDDI standard implementation. Each of these works aims to introduce concept matching to the UDDI registry by incorporating reasoning and OWL-S support to current implementations. The active research in this area highlights one of UDDI's main weaknesses, lack of service capability support and emphasises a general consensus amongst the web service academic community that semantic support for capability matching of web services is primary the area forward.

In [23] an efficient way to apply the matching methodologies is also proposed from the design outlined in [11] to extend the UDDI Registry. This basic extension adds a *capability port* to the current UDDI implementation thus making it semantically aware. An interesting contribution of this paper is an evaluation of ranked matching and a resulting focus on accelerating performance by minimising the amount of matching and, therefore reasoning, that takes place. Any implementation of a semantic matching engine into the publish subscribe model must take this observation into consideration and must endeavour to minimise the frequency of concept matching that takes place.

The work described in [12] by Jaeger et. al. focuses on a finer grained approach to matching than presented in [21]. By consideration of the service category and finer-grained user constraints based on concept properties as well as input and output matching the work done by Jaeger et. al. proposes a more accurate approach to semantic matching. The semantic matcher uses an aggregation of these finer grained steps to produce a more accurate matching result. A Java prototype has been built and is hosted by the Technischen Universitat at Berlin [20]. As a freely available and mature implementation of progress in the semantic matching area, this matcher was the basis for the enhanced matcher used in section 5.

The METEOR-S project [28] is a large research effort focusing on the application of semantics to WSDL, in the form of WSDL-S, and semantic support to UDDI. Interestingly in the context of this research, [28] appears to offer significant contribution in the area of distributed, peer-to-peer infrastructures for semantic publication and discovery of services. METEOR targets the area of semantic web-service discovery as an important and apparently underdeveloped area in the context of current web-services

research. METEOR also makes the observation that for a UDDI registry in its current form, web-service discovery across multiple UDDI-registry nodes is inefficient. The research concludes that adding web service description semantics and annotating the UDDI nodes themselves may provide avenues for improving the efficiency of a distributed UDDI registry. This is interesting considering the content based routing avenue of this research. Work carried out for the METEOR-S project shows that the semantic concepts available in OWL can be used to for other purposes besides explicit matching of service descriptions against requirements.

7. CONCLUSIONS

The main of aim of this paper was to explore a potential for a UDDI alternative, to investigate the feasibility of a general publish/subscribe model for web service discovery, and to explore the potential usability of semantic content of OWL-S web service descriptions in enabling efficient content-based routing within this publish/subscribe model.

In addition we have described how we extended a Content Based Network (Siena) to support ontologically based matching in general, leading to a more flexible subscription model and knowledge based content routing. We argue that this knowledge-based routing approach is of great utility in several different domains, but there may be a need to introduce specific matchers to bring efficiency in particular application domains, such as semantic web service discovery. We demonstrated this by replacing the general ontology matcher implemented with a OWL-S ontology based matcher developed by the Technischen Universitat of Berlin.

In terms of a an alternative to UDDI we have shown that more tightly integrated, distributed models for service discovery are possible, feasible and are currently under development. We conclude here that active research into overcoming the shortfalls of UDDI continue both along the publish/subscribe track of research presented here, and along the METEOR-S track of research already undertaken and presented.

By developing a service-discovery platform that uses the publish/subscribe model we have proposed a proactive approach to service discovery that unites research conducted in he publish/subscribe and the service discovery domains. We have shown by proof of concept that the model holds strong in the presence of OWL-S based capability matching integrated at the routing-table level of the event notification system. Also, by introducing semantics to the text based notification and subscription storage structure of Siena, we have shown that OWL-S types and associated semantic information can be utilised in content-based routing system for semantic web services.

8. FUTURE WORK

This proof of concept has opened the door for a larger scale implementation and statistical evaluation. It is envisaged that this implementation would not only examine the presented implementation under wide-area scale, but may include a combined integration of peer-to-peer routing mechanisms, semantic clustering of peers, improved matching capabilities and enhanced subscription languages.

The low level networking performance characteristics [7] were not of prime importance to this case study use of a semantically enriched content delivery network. However, a full analysis of both the proof of concept and any peer-to-peer system in terms of network cost per subscription and per subscription terms would be useful in gauging the performance of the enhanced subscription matching algorithm for each subscription and publication. Such measures would more clearly demonstrate the general impact of the capability matcher.

In light of these proposed analyses, we feel that a study of two major modifications would be beneficial. Firstly, the incorporation of a peer-to-peer routing model to replace the current hierarchical model may have interesting implications for the subscription tree structure and the subsumption relations as defined previously. Secondly, investigation is warranted into how the fast-forwarding algorithms for Siena [8] may be used to further improve overall system performance.

Perhaps more removed from the immediate goals of the project, it is felt that further work on the reduction of the frequency of XML parsing, perhaps through assertion sharing, coupled with further research into knowledge base and ontology alignment would definitely be of benefit to the research area in the long term.

9. ACKNOWLEDGMENTS

This work was supported partially by Science Foundation Ireland and Enterprise Ireland through the Centre for Telecommunication Value Chain Research..

10. REFERENCES

[1] Akiragiu, R., Goodwin, R., Doshi, P., and Roeder, S., “A method for semantically enhancing the service discovery capabilities of uddi”. Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03) August 9 - 10, 2003, Acapulco, Mexico

[2] AKT. The aktors portal ontology, http://www.aktors.org/ontology/portal

[3] Berners-Lee, T., Hendler, J., Lassila, O., “The Semantic Web”, Scientific American, May 2001

[4] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., “Web services architecture”. Technical report, W3C, February 2004. Available online at http://www.w3.org/ws-arch/

[5] Carroll, J., Dickinson, I., Dollin, C., “Jena: Implementing the Semantic Web Recommendations”, in Proc. of World Wide Web Conference 2004, 17-22 May 2004, New York, NY, USA. http://jena.sourceforge.net/

[6] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L., “The Design and Evaluation of a Wide-Area Event Notification Service”, ACM Transactions on Computer Systems, Vol. 19, Issue 3, August 2001

[7] Carzaniga, A., Wolf, A.L., "A Benchmark Suite for Distributed Publish/Subscribe Systems". Technical Report CU-CS-927-02, Department of Computer Science, University of Colorado, April, 2002.

[8] Carzaniga, A., Wolf, A. L., “Forwarding in a Content-Based Network” in proc SIGCOMM’03, August 25-29 2003, Karlsruhe, Germany, ACM Press

[9] “Covering relationships in Siena”, http://serl.cs.colorado.edu/~carzanig/siena/forwarding/ssimp/namespacesiena.html#a1

[10] DAML Technical Committee. “OWL-S : Web Ontology Language for Web Services”. http://www.daml.org/services/owl-s/

[11] Eugster, P., Felber, P., Kenmarrec, A.M., and Guerrout, R., “The many faces of publish/subscribe”. ACM Computing Surveys (CSUR), Vol. 35, Issue 2, (June 2003), pp 114 – 131, 2003.

[12] Jaeger, M., Rojec-Goldmann, G., Liebetruth C., Muhl, G., and Geihs, K.,. “Ranked matching for service descriptions using owl-s”. Kummunikation in Verteilten Systemen (KiVS 2005), Kaiserslautern, Germany, February 2005

[13] Keeney, J., Lewis, D., O’Sullivan, D., Roelens, A., Boran, A., Richardson, R., “Runtime Semantic Interoperability for Gathering Ontology-based Network Context”, to appear in proc 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), April 3-7, 2006, Vancouver, Canada.

[14] Keeney, J., Lynch, D., Lewis, D., O’Sullivan, D., “On the Role of Ontological Semantics in Routing Contextual Knowledge in Highly Distributed Autonomic Systems” Technical Report (TCD-CS-2006-15), Department of Computer Science, Trinity College Dublin. 2006.

[15] Kopena, J., OWLJessKB: A Semantic Web Reasoning Tool. http://edge.cs.drexel.edu/assemblies/software/

[16] Lynch, D., “A Proactive approach to Semantically Oriented Service Discovery”, MSc dissertation, Deptartment of Computer Science, Trinity College Dublin. 2005.

[17] MINDSWAP. Pellet: An OWL Reasoner. http://www.mindswap.org/pellet

[18] The OASIS Organisation. http://www.oasis.org.

[19] OWL Technical Committeee. Web Ontology Language (OWL). http://www.w3.org/2004/OWL

[20] OWLSM: The TUB OWL-S Matcher. http://kbs.cs.tu-berlin.de/ivs/Projekte/owlsmatcher/

[21] Paolucci, M., Kawamura, T., Payne, R., and Sycara, K.. “Semantic matching of web services capabilities”. In proc. of International Semantic Web Conference (ISWC), Sardinia, Italy.

[22] “Pellet Performance. http://www.mindswap.org/2003/pellet/performance.shtml

[23] Srinivasan, N., Paolucci, M., and Sycara, K., “An Efficient Algorithm for OWL-S based Semantic Search in UDDI”. In proc. First International Semantic Web Services and Web Process Composition Workshop (SWSWPC 2004), San Diego, CA, USA, 2004.

[24] RDF Technical Committee. “The resource description framework (RDF)”. http://www.w3.org/RDF/

[25] RACER: An Inference Engine for the Semantic Web. http://www.franz.com/products/racer/.

[26] UDDI Technical Committee. “UDDI API specification v2.0.4. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf

[27] UDDI Technical Committee. “UDDI API specification v3.0.2”. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf

[28] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., and Miller, J., “METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services”. Inf. Tech. and Management, 6(1):17–39, 2005.

[29] The World Wide Web Consortium. http://www.w3c.org