The ICENI Semantic Service Adaptation Framework

Jeffrey Hau, William Lee, Steven Newhouse
London e-Science Centre,
Imperial College London, Science, Technology and Medicine,
180 Queen's Gate, London, SW7 2AZ, United Kingdom
lesc@imperial.ac.uk

August 14, 2003

Abstract

With the advent of web services standards and a service-oriented Grid architecture, it is foreseeable that competing as well as complimenting computational services will proliferate. Current efforts in standardising service interface focuses on how one can execute these services in terms of their syntactic descriptions. Their capabilities and relations with other service types are only articulated through natural language in the form of documentation. In this paper, we present the ICENI semantic service adaptation framework. We seek to capture the capability of services by annotating their programmatic interface using the Web Ontology Language (OWL)[2] in relation to some domain concepts thereby allowing services to be semantically matched based on their ontological annotation. By inferences on this metadata, syntactically different but semantically equivalent service implementations may be autonomously adapted and substituted. Combining it with familiar high-level programming language, we demonstrate a practical service-oriented programming model.

Keywords: Ontology, Middleware, Adaptation.

1 Introduction

The World Wide Web has evolved from being a collection of hyperlinked documents to a platform delivering networked services. The e-Commerce community needs to integrate heterogeneous distributed systems within their supply-chain while the Grid Computing community must access federated and widely-distributed computing resources in a scalable, secure and coordinated manner. By representing these system units as well-defined services, we facilitate the reuse and substitution of compatible components into the workflow of a larger system.

Compatible services are often identified through subclass relationship or interface inheritance. Substitutable implementation are reused through polymorphism of superclass or abstract interface. Far from being an autonomous operation, the burden of using binding specific tools or APIs to generate stubs or perform remote invocation has added complexity in consuming remote services. This approach to identifying reusable services is insufficient in a service-oriented architecture. Multiple service providers might not agree on a single service interface hierarchy for competitive reason. These services are deemed incompatible although they potentially provide the same functionality. The intended capability of contemporary services are mostly expressed in natural languages, which hinders automated matching because of potential ambiguity and incompleteness. Research in formal methods has explored the precise description and intention of interfaces. Even though the task of developing formal specification is overwhelming and the need for program verification deemed unnecessary for many applications.

In this paper we present a framework for identifying compatible services based on their expressed capabilities in relation to some ontological concepts. By associating service operations to concepts and parameters to properties of the concepts, we demonstrate these semantic metadata, which links the syntactic elements of the service to the domain ontology, can be inferred to identify compatibility. We argue that the intelligence provided by the inference can identify the points of compatibility between the client requirement and the ser-

vice. In addition, the middleware needs to possess a set of transformation rules for adapting and proxying the service consumable by the client.

1.1 Background

The Web Services Standards[9][10][8] collectively model the publish, find and bind operations of the service oriented architecture. The Web Services Description Language (WSDL) captures the programmatic interface, network binding and message formats of the service irrespective of their implementing technologies. Its extensible nature is well-suited to act as a common language for describing service interface encoded in CORBA IDL, Java Interface or other proprietary interface description language. Nonetheless, it only expresses the syntactic specification of the service in terms of the name of the methods, as well as the expected types. The Open Grid Services Interface[11] (OGSI) builds on the relative maturity of WSDL with the added notion of transient service. It mandates a set of core interfaces and service data to reflect the need for a set of fundamental services in a Grid environment. It implants limited semantics to all Grid services in terms of the ability to query state and control life-cycle. However, custom services have to resort to the traditional methods to express their capability.

Research in Autonomic Computing[16] aims to transform computing tasks that require constant human interference and awareness, to be self-managing, self-healing and self-optimising. Efforts in the Semantic Web[4] has contributed standards and tools to autonomically extract structured information from the World Wide Web. As the Internet is evolving into a platform for service delivery, we would not only like to search and identify related service but to consume the service according to their service interface.

2 The Metadata Space

Expressing resource semantics on the World Wide Web has been the driving force behind the recent development of the Semantic Web. The Grid community is exploiting recent development in grid computing technologies and service oriented architecture to build the Semantic Grid[5], an grid infrastructure with resource and services semantics. In this paper we propose the concept of *metadata space* as realisation of the Sematic Grid. The separation of the metadata space from the physical grid in figure 1 clarifies the the difference between a service metadata and its implementation. In the metadata space, the usual manual extraction of service semantics becomes an autonomic process. Each grid resource is represented by its semantic annotation and thus resource interaction is focused purely on the semantic descriptions.

More specifically, the metadata space is an environment with a standard metadata publication and discovery protocol to facilitate the processing of metadata and semantic interaction between grid resources. The advantage of having the metadata space separated from the physical grid is to decouple grid resources from their implementation and hosting environment. Every participant in the metadata space is characterised as a metadata publisher. The metadata published by a publisher falls into one of the three categories - requirement, implementation or domain. We distinguish the publisher by their metadata category. This differs from the traditional service oriented architecture where participants can be characterised as either a client, a service, or a registry. The metadata published into the metadata space can then be processed by the Meta-Services to provide the autonomic semantic matching and interface adaptation. The following subsections discuss the different roles of publishers in more details.

2.1 Implementation Publisher

This role can be played by any service providers in the traditional service oriented architectures. The transformation form service to implementation publisher depends on the publication of its semantic annotation into the metadata space. An example of implementation publisher's semantic annotation is shown in figure 2[1]. The implementation provider behaves as a typical service provider within the its hosting environment. It projects its identity/exsistence into the metadata space by the publication of its semantic annotation.

[1]Following is a list of XML namespace prefix used in this paper rdf= http://www.w3.org/1999/02/22-rdf-syntac-ns#, owl= http://www.w3.org/2002/07/owl#, rdfs= http://www.w3.org/2000/01/rdf-schema#, art=http://lesc.ic.ac.uk/example/arithmetics

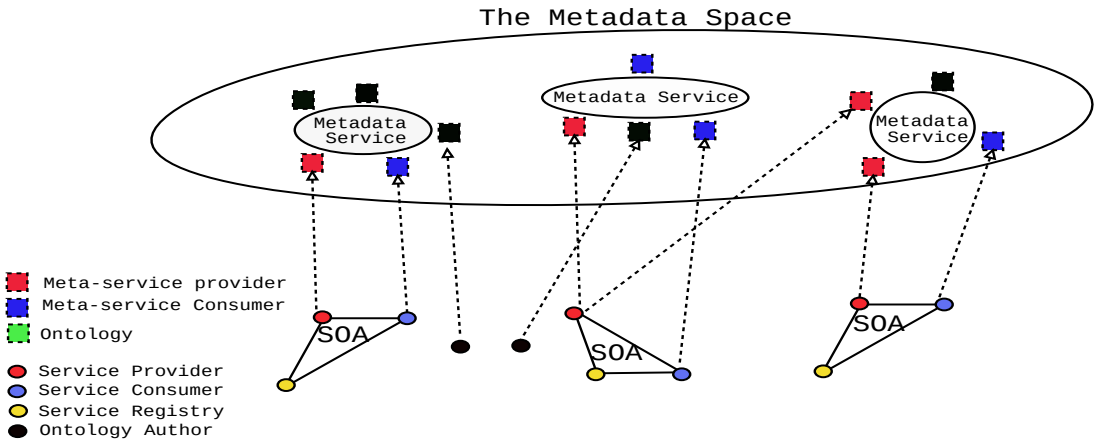

Figure 1: The metadata space in relation to different Grid SOAs

```
//Semantic Annotation
<Description about="MathsService/add">
  <type resource="&art;Add"/>
  <art:hasOperand
     resource=".../add/param/a"/>
  <art:hasOperand
     resource=".../add/param/b"/>
  <art:hasResult
     resource=".../add/return"/>
</Description>

//Java service implementation
public int sum(int[] ele) {
      //method body omitted
}
```

Figure 2: A Java service provider's semantic annotation in RDF

2.2 Requirement Publisher

A requirement publisher is any grid service consumer with the capability of publishing semantic annotation into the metadata space. The requirement publisher expresses its syntactic requirement in a programming language interface. Semantic requirement is then inlined into the source code through annotation. The syntactic service interface description has no importance in the metadata space. Requirement and implementation are matched by using the published semantic metadata. The interoperation between different service oriented architectures is achieved autonomically with the adaptation process. The requirement publisher (service consumer) does not need to intervene programmatically. The publishers can program with their interface without worrying about the plumbing code that obscures the application design.

2.3 Ontology Publisher

Anyone who writes and publishes ontology into the metadata space is characterised as an ontology publisher. This role is usually taken up by domain expert or standard bodies e.g the Dublin Core Metadata Initiative[17]. Ontology are used by meta-services as a knowledge store to support their inference engines. They can only then provide the essential semantic inference capability for the metadata space. The semantic matching and service adaptation process both rests crucially on the quality and quantity of the published ontologies in the metadata space. Trust and consistency of ontologies are important issues and are examined in section 5.

2.4 Meta-service

The meta-services provide the metadata space with semantic matching and service adaptation capabilities. Semantic matching services find compatible implementation and requirement based on their published service semantics only. Adaptation services generate service adaptation by using the syntactic service detail such as type declaration, service binding etc. Adaptation services are characterised by services that reside in multiple grid architectures and/or different syntactic description. This enable them to act as the bridge across different services, thus providing a necessary adaptation between the requirement and the implementation publisher. Meta-services form the hotspots (see figure 1) of the metadata space, all three types of published information are gathered to enable the matching and adaptation process.

3 Semantic Matching and Service Adaptation

Semantic matching and interface adaptation are the two operations that enable the autonomic adaptation of grid services. We envision a range of different metadata-service satisfying these purposes to emerge in the semantic grid. This section focuses on the high level requirement/specification rather than the implementation detail of these services.

3.1 Semantic Annotation

The meaning of services is implicitly expressed by the implementation expressed in the form of the programming language source code. The purpose of the semantic annotation is to express this intrinsic meaning explicitly and in a machine processable way. The Resource Description Framework (RDF)[1] from W3C was designed to serve this purpose and the Web Ontology Language (OWL) builds on RDF to provide a way of adding domain specific vocabulary for resource description by using concepts taxonomy.

Semantic annotation of a service is developed in 2 stages. First, the user annotates a service with intended meaning. This express the service in terms of some domain concept and property in OWL. Next, different aspects of a service method[2] (for example, name, parameter, return type) need to be described independently as distinct resources. This stage concentrates on expressing the syntactic meaning of the service by annotating the semantics of the definition of the service method. This metadata is important for the adaptation. Without this annotation, the adaptation service will not be able to *understatnd* the meaning of the different service aspects. In figure **??** we introduce a toy object oriented programme ontology for annotating programme structure. This type of annotation can be generated automatically by a suitable implemented source code parser, thus moving some of the complexity of the annotation process away from the human writer.

3.2 Semantic Matching

The first step in autonomic service adaptation is to find services that are conceptually equivalent to the client's requirements. These requirements are expressed through the semantic annotation of the interface by using OWL. This ties each of the interface method to a domain concept. Semantic annotations from implementation publishers are defined to be conceptually equivalent to the requirement when the requirement concept is inferrable from the implementations's annotations. In practice, whether one concept is inferrable from another depends on the search and inference capability of the semantic matching service and the presence of suitably defined ontology in the metadata space.

A semantic matching service will need to perform two main inference operations - class and property inferencing. Each interface annotation ties the concept of a method to a ontology class. The result of class inferencing is a set of services that are conceptually equivalent to the client interface method. Class property are represented by method signatures. Property inferencing narrows down the conceptually matched services inspecting properties and their relations. This process refines the original list by taking into account ontology class properties. This is a crucial step in enabling service adaptation. It filters out conceptually incompatible services from the requirement. For example, a summation service with a parameter that refers to a precision property of a summation concept is incompatible with a requirement interface that is defined with two integers, both relate to the operand property of the summation concept. The final list of services are the ones that matched with the client annotation's class concept and properties. These are defined as conceptually compatible.

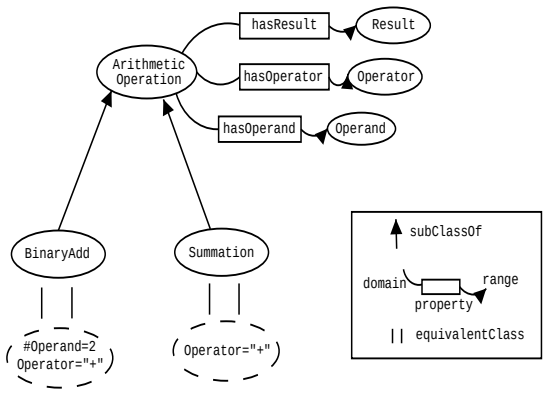

Figure 3: A toy ontology about addition

[2]It is important to note that semantic annotations can be written at different levels in the source code (statement, block, class,..) to give different levels of expressiveness. Due to the scope of the paper, we concentrate our efforts on the method level. Please see discussion for future works.

3.3 Code Adaptation

Adaptation service generates the adapted implementation code for the requirement publisher. Receiving a list of conceptually equivalent implementations from the matching service, this service dynamically generates the adaptor proxy on demand. In the adaptation stage, the focus shifts from the semantic metadata about the method to the method structure metadata. The aim of the adaptation service is to find a transformation function with the required method signature as domain and the list of conceptually compatible service signatures as the range.

Invocation Architecture Selection First step in the generation of the adaptor is to identify the hosting architecture of the target service. If the target service's hosting architecture cannot be handled by any of the available invocation handler then the adaptation process for the particular service cannot continue. This identification process is achieved by inspecting the target service's syntactic description. We use WSDL as the syntactic service description langauge and the binding element in the WSDL document determines the hosting architecture.

Signature Transformation The structural information about method signature is described in the method structure metadata. This metadata alone is not enough for the transformation. Every adaptation service needs a set of basic transformation rules - axioms, to start a backward chaining transformation process. For example, a simple axiom that expresses the notion of an identity, is necessary in order to eliminate parameters that are conceptually equivalent (a+b = a+b+0). Type information can also be extracted from the requirement and implementation's program structure metadata. Inference from one to the other requires the presence of a suitable type ontology in the metadata space. In the example Java type ontology in figure 4, it is possible to infer that a pair of integers a and b are conceptually equivalent to an array of two integers. More specifically, the ontology allows an OWL inference engine to infer that an integer is an instance of element and therefore it can be treated as one of the element in an integer array. This is a very simple example of type inference, more sophisticated inferencing ability (such as cross programming language type inferencing) will come with more refined type ontology.

```
<rdf:RDF>
   <owl:class rdf:ID="Array"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty
              rdf:resource="#eleType"/>
            <owl:Cadinality
              rdf:datatype="&xsd;Integer">
               1
            </owl:Cardinality>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:class>
   <owl:class rdf:ID="IntArray">
      <rdfs:subClassOf rdf:ID="#Array"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty
              rdf:resource="#eleType"/>
            <owl:allValuesFrom
              rdf:resource="#Integer"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:class>
   <owl:ObjectProperty rdf:ID="eleType">
      <rdfs:domain rdf:resource="#Array"/>
      <rdfs:range rdf:resource="#Element"/>
   </owl:ObjectProperty>
   <owl:class rdf:ID="Integer">
     <rdfs:subClassOf rdf:ID="#Element"/>
   </owl:class>
</rdf:RDF>
```

Figure 4: A Simple Ontology of Java Types

4 The ICENI Semantic Layer

The Imperial College e-Science Networked Infrastructure[18] - ICENI - is a service oriented/integrated Grid middleware that provides an augmented component programming model to aid the application developer in constructing Grid applications, and an execution infrastructure that exposes compute, storage and software resources as services with defined conditions of when and by whom these resources may be used. It utilises open and extensible XML schemas to encapsulate meta-data relating to resource capability, service availability and application behaviour.

We are currently developing a prototype implementation of the semantic matching and service adaptation framework presented in this paper. The goal is to integrate this framework with the metadata facility in ICENI to bring about the semantic reasoning and adaptation ability. The prototype implementation is scheduled to be completed in three stages, the metadata space implementation lays the foundation as the knowledge store. Ontology processing is built on top of the metadata space and finally the service adaptation framework. In

the following sections we present the current implementation status of each of the layers.

4.1 Metadata Space

The metadata space represents a layer on top of grid architecture implementations. To reflect this important distinction, ICENI metadata space API is constructed from a core collection of java interfaces to enable the separation of function representations and implementations. The metadata space API (MS-API) gives maximum flexibility for the publication and discovery of RDF annotated semantic service metadata by allowing custom architecture handlers.

The main functionality that the MS-API provides is for the different grid architectures to be binded to the metadata space through Architecture Handlers. Once a handler is registered with the metadata space, users can then carry out publication and discovery operations through a uniform interface. This consistent interface shifts the underlying complexity of programming with different service oriented architectures away from the end user to the developers. Architecture Handlers are currently categorised into two types - Meta-service and Metadata Handlers.

Meta-service handlers are the bindings between the concrete service implementations and their Meta-service representations in the metadata space. In our current prototype implementation, we have developed a Meta-service handler targeting the Jini[28] architecture. This handler enables the searching and invocation of Meta-services with concrete Jini implementation. We have also implemented a basic semantic matching service in Jini. The core of the matching service is its ontology inference engine - Euler[27]. Euler is a pure Java, backward chaining rules based proof engine. It uses the resolution inference mechanism and only follows Euler paths to avoid endless deductions.

Metadata handler binds the different data store implementations with the metadata space abstraction. The current handler under development is based on the JXTA architecture. JXTA allows the service semantic metadata to be published and discovered in a peer to peer manner. This is to be evaluated against the standard web service (client-server) model of publishing and discovering service metadata.

4.2 Metadata Processing

The metadata processing layer is built using the JENA2[7] semantic web toolkit. JENA2 provides comprehensive support for RDF, DAML and OWL. This includes query, data storage and built-in ontology inference engines. The one of the main reason for choosing JENA as the processing toolkit is its pluggable framework. JENA2 allows different inference engine[13][14] and ontology language to be used in our framework. It is unrealistic to assume that all participants in the metadata space will be using OWL as the ontology language and a standard inference engine for all their needs. JENA2 is currently still in beta and the inference support is still preliminary. We have currently adapted Euler as the main inference engine for our framework.

The metadata processing layer is primarily concerned with the parsing and generation of semantic metadata. Service interface metadata can be automatically generated from a Java source code parser. The service semantic data will have to be hand written by the user. The aim of the meta processing layer is to provide a JAVA2RDF parser for the generation of the service interface metadata and a service semantic metadata validator. There is no plan in writing a high-level semantic data editor since there are currently many widely available tools for writing ontology based metadata such as OilEd[24] and Protege[25].

4.3 Service Adaptation

The prototype service adaptation engine transforms interface signature by using a set of graph transformation[26] rules. In RDF every sentence has the form $< object >< predicate >< subject >$. It is therefore natural to view RDF statements as labelled directed graphs with URI as graph label and every graph edge has direction going from object to the subject. When semantic metadata is expressed in terms of RDF, although written in XML, it can be easily transformed into labelled directed graphs. In our adaptation engine, we take this 'graphical' approach to tackle the problem of transformation. The engine try and match the client requirement with the list of semantically compatible services using the available graph transformation production rules. Once a match is found the adaptation engine then generates the dynamic proxy by using the Java reflection's Proxy and InvocationHandler API to provide the client-required interface.

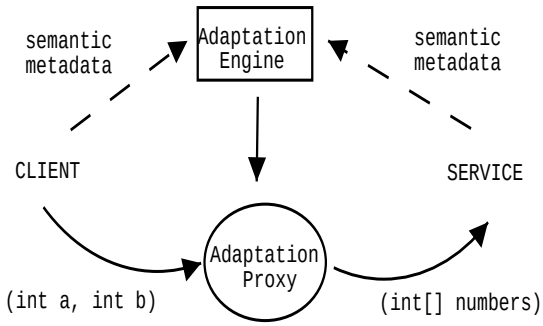

Figure 5: The adaptation proxy

We plan to use the Apache Web Service Invocation Framework[15](WSIF) as the basis of our work on dynamic service architecture selection. WSIF enables users to interact with abstract representation of web/grid services through their WSDL description. Our future work involve extending WSIF into a Grid Service invocation framework by adding the capability of processing WSDL extension elements defined in OGSI.

5 Discussion

5.1 Workflow and Scheduling

Componentised software encapsulates reusable functionality to be deployed on-demand. Current research on Workflow languages[19][20] and Grid Scheduling focuses on the description of service interaction, as well as exploiting knowledge on performance characteristics of service implementation to make deployment decision. By raising the abstraction of service capability using ontology, schedulers can widen their scope of service selection. Ontological equivalent service implementation can be exploited if it can be adapted transparently to the abstract workflow. Effort in establishing ontological vocabulary for describing workflow and web services[21] creates a foundation for further research in optimising workflow based on the conceptual intention of the composition.

5.2 Trust in Ontology and Adaptation

The metadata space is analogous to a public bulletin board. Concepts are being introduced by independent parties in a distributed and uncoordinated manner. This gives rise to the problem of ensuring logical consistency of ontology replicated at multiple autonomous peers. Moreover, the metadata space needs to guard against malicious attempts in introducing false concepts. Possible solutions might be to sign the OWL fragment using XML Digital Signature[23], and implement a distributed voting[22] in achieving common ontology consensus among peers. The quality and correctness of the adaptation would be difficult to verify without formal analysis of the adaptation code. By optionally exposing the adaptation service selection to the user-level, flexible selection schemes can be devised based on the level of trust.

5.3 Stateful Services

Service is considered as a collection of functionalities encapsulated in a set of methods. Our current prototype focuses on annotating and adapting at the method level. Multiple stateless services can be grouped and interchanged, so that they can be adapted to serve a single client interface requirement. However, invoking a method on a stateful service would impose state change, that causes side-effect when other methods are being called. As a result, stateful services can only be grouped to form one instance if their states are explicitly made shared and kept consistent.

6 Conclusion

In this paper we have presented ICENI's semantic service adaptation framework based on ontological annotations. The three aspects of this framework were presented: the metadata space, semantic matching and service adaptation. The metadata space was proposed as an independent knowledge store for the semantic grid. Semantic matching allows us to search for services using their semantic rather than syntactic descriptionn. Finally, the service adaptation process exploits the intelligence gathered during the matching phase, and uses a set of transformation rules to generate an architecture independent binding on demand.

7 Acknowledgements

We gratefully acknowledge the support from the UK e-Science Core Programme sponsored by the Department of Trade and Industry; the Overseas Research Student Award administered by Universities UK on behalf of the Department of Education and Skills; and finally the European DataGrid Project funded by the European Union.

References

[1] Lassila O., Swick R.R., Resource Description Framework (RDF) Model and Syntax Specification W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[2] Patel-Schneider P.F., Hayes P., Horrocks I., OWL Web Ontology Language Semantics and Abstract Syntax W3C Working Draft 31 March 2003, http://www.w3.org/TR/owl-semantics/

[3] Heflin J, Web Ontology Language (OWL) Use Cases and Requirements W3C Working Draft 31 March 2003, http://www.w3.org/TR/webont-req/

[4] W3C Semantic Web Activity, http://www.w3.org/2001/sw/

[5] The Semantic Grid Community Portal, http://www.semanticgrid.org/

[6] JXTA, http://www.jxta.org/

[7] Jena Semantic Web toolkit, http://www.hpl.hp.com/semweb/jena.htm

[8] Web Service Description Language (WSDL), Christensen E., Curbera F., Meredith G., Weerawarana S., http://www.w3.org/TR/wsdl

[9] Box D., Ehnebuske D, Kakivaya G., Layman A., Mendelsohn N., Nielsen H. F., Thatte S., Winer D., Simple Object Access Protocol (SOAP)1.1 W3C Note 08 May 2000, http://www.w3.org/TR/SOAP/

[10] Universal Description, Discovery and Integration, http://www.uddi.org

[11] Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Snelling D., Vanderbilt P., Open Grid Service Infrastructure (OGSI) draft, Feburary 2003

[12] Sintek M., Decker S., Kesselman C., Nick J. M., Tuecke S., The Physiology of the Grid (Draft)

[13] Sintek M., Decker S., TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web, June 2002

[14] RACER, http://www.fh-wedel.de/ mo/racer/index.html

[15] Web Service Invocation Framework, http://ws.apache.org/wsif/

[16] Autonomic Computing, http://www.research.ibm.com/autonomic/

[17] Dublin Core Metadata Initiative, http://dublincore.org/

[18] Furmento N., Lee W., Mayer A., Newhouse S., Darlington J., ICENI: An Opern Grid Service Architecture Implemented with JINI, SuperComputing 2002, November 2002

[19] Andrews T., Curbera F., et al, Business Process Execution Language for Web Service Specification Version 1.1, May 2003

[20] Arkin A., et al., Web Service Choreography Interface (WSCI) 1.0 W3C Note 8 August 2002

[21] Ankolekar A., et al., DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference (ISWC), June 2002

[22] Williams A., Krygowski T., Thomas G., Using Agents to Reach an Ontology Consensus, AAMAS 02, Bologna, Italy, July 2002

[23] XML Signature, http://www.w3.org/Signature/

[24] OilEd, http://oiled.man.ac.uk/

[25] Protege-2000, http://protege.stanford.edu/

[26] Corradini A., Montanari U., Rossi F. et al., Algebraic Approaches to Graph Transformation Part 1: Basic Concepts and Double Pushout Approach, 1997

[27] Jos De Roo, http://www.agfa.com/w3c/euler/

[28] http://www.jini.org/