XML Metadata Services

Mehmet S. Aktas, Sangyoon Oh, Geoffrey C. Fox, and Marlon E. Pierce

***Abstract*—** As the Service Oriented Architecture (SOA) principles have gained importance, an emerging need has appeared for methodologies to locate desired services that provide access to their capability descriptions. These services must typically be assembled into short-term service collections that, together with code execution services, are combined into a meta-application to perform a particular task. To address metadata requirements of these problems, we introduce XML Metadata Services to manage both stateless and stateful (transient) metadata. We leverage the two widely used web service standards: Universal Description, Discovery, and Integration (UDDI) and Web Services Context (WS-Context) in our design. We describe our approach and experiences when designing "semantics" for XML Metadata Services. We report results from a prototype of the system that is applied to mobile environment for optimizing Web Service communications.

***Index Terms*—**XML metadata services, Grid/Web services, Service Oriented Architectures, WS-Context

I. Introduction

As the Service Oriented Architecture (SOA) principles have gained importance, an emerging need has appeared for methodologies to locate desired services that provide access to their capability descriptions. As these services interact with each other within a workflow session to produce a common functionality, another emerging need has also appeared for storing, querying, and sharing the resulting metadata needed to describe session state information.

Zhuge identifies the two mainstream research focuses in next-generation Web in [1]. The first research theme investigates how to overcome the existing Web's limitations such as difficulties in supporting intelligent services. Some example areas of investigation of this approach are Semantic Web and Web Services. The second research theme focuses on the Grid as an alternative application platform. The Grid offers a model for solving computational science problems by utilizing the idle resources of large numbers of distributed computers. Zhuge also mentions the Semantic Grid research, as an extension to the Grid, evolved as result of integration of the two aforementioned mainstream research themes.

Mehmet S. Aktas is with Department of Computer Science and the Community Grids Laboratory, Indiana University Bloomington, IN 47404. (Phone: 812-856-0755, email: maktas@cs.indiana.edu).

Sangyoon Oh is with Department of Computer Science and the Community Grids Laboratory, Indiana University Bloomington, IN 47404. (email: ohsangy@cs.indiana.edu).

Geoffrey C. Fox is with Indiana University Departments of Computer Science and Physics and the Community Grids Laboratory, Bloomington, IN 47404. (email: gcf@indiana.edu.)

Marlon E. Pierce is with the Community Grids Laboratory, Indiana University Bloomington, IN 47404 (email: mpierce@cs.indiana.edu).

As the SOA-oriented architectures gained popularity in both the traditional and Semantic Grid, metadata management problems of Grid applications form an important area of investigation. For an example, Geographical Information Systems (GIS) provide very useful problems in supporting "virtual organizations" and their associated information systems. These systems are comprised of various archival data services (Web Feature Services), data sources (Web-enabled sensors), and map generating services. All of these services are metadata-rich, as each of them must describe their capabilities (What sorts of features do they provide? What geographic bounding boxes do they support?). Organizations like the Open Geospatial Consortium define these metadata standards.

These services must typically be assembled into short-term, stateful service collections that, together with code execution services and filter services (for data transformations), are combined into a composite application (e.g. a workflow).

To address metadata requirements of these problems, we introduce XML Metadata Services to manage both stateless and stateful (transient) metadata. We use and extend the two Web Service standards: Universal Description, Discovery, and Integration (UDDI) [2] and Web Services Context (WS-Context) [3] in our design. We utilize existing UDDI Specifications and design an extension to UDDI Data Structure and UDDI XML API to be able to associate both prescriptive and descriptive metadata with service entries. We extend WS-Context specifications to provide search/access/storage interface to session metadata.

In this paper, we describe the "semantics" of the proposed XML Metadata Services and give an overview of implementation details. In addition, we also discuss a motivating application scenario and the way that the hybrid XML Metadata Service is being used. We report results from a prototype that has been applied to mobile environment for optimizing Web Service communications.

II. Background

There have been some approaches introduced to provide better retrieval mechanism by extending existing UDDI Specifications. UDDI-M [4] and UDDIe [5] projects introduce the idea of associating metadata and lifetime with UDDI Registry service descriptions where retrieval relies on the matches of attribute name-value pairs between service description and service requests. In our design, we too extend UDDI's Information Model, by providing an extension where we associate metadata with service descriptions

similar to existing solutions where we use name-value pairs to describe characteristics of services. Apart from the existing methodologies, we provide both general and domain-specific query capabilities. An example for domain-specific query capability could be XPATH and RDQL queries on the auxiliary and domain-specific metadata files stored in the UDDI Registry.

The primary use of our approach is to support information in dynamically assembled workflow-style Grid applications where services are tied together in a dynamic workflow to solve a particular problem. There are varying specifications, such as WSRF [6], WS-Context, WS-Transfer [7], and WS-Metadata Exchange [8], that have been introduced to define stateful interactions among services. Among them, we have chosen the WS-Context specifications to create a metadata catalog system for storing transitory metadata needed to describe distributed session state information. Unlike the other specifications defining service communications, WS-Context models a session metadata repository as an external entity where more than two services can easily access/store highly dynamic, shared metadata.

III. XML Metadata Services: Semantics, Authentication, and Authorization Mechanisms

We have designed and built a novel architecture [9-10] for an hybrid WS-Context complaint metadata catalog service supporting handling and discovery of not only quasi-static, stateless metadata, but also session related metadata. We based the information model and programming interface of our system on two widely used specifications: WS-Context and Universal Description, Discovery and Integration (UDDI) as depicted in Figure 1.

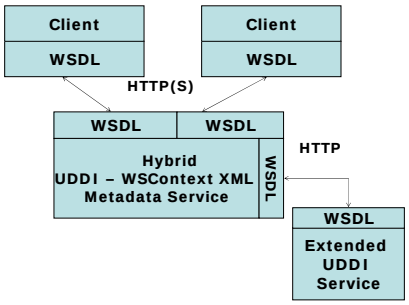

Figure 1. This figure illustrates the two clients interacting with the Hybrid UDDI – WSContext XML Metadata Service, while the hybrid service is interacting with an external UDDI Service for handling and discovering of static metadata.

We have identified following base elements of the semantics of proposed system: a) data semantics, b) semantics for publication and inquiry XML API, and c) semantics for security and access control XML API. These semantics have been designed under two constraints. First, both UDDI and WS-Context Specifications should be extended in such a way that client applications to these specifications can easily be integrated with the proposed system. Second, the semantics of the proposed system should be modular enough sot that it can easily be operated with future releases of these specifications..

A. Extensions to UDDI Data Model

The extended version of UDDI information model consists of various additional entities to existing UDDI Specifications (Detailed design documents can be found at http://www.opengrids.org/-extendeduddi). These entities are represented in XML. We describe extensions to UDDI information model as following: serviceAttributeEntity: A service attribute data structure describes metadata associated with service entities. Each “serviceAttribute” corresponds to a piece of metadata and it is simply expressed with (name, value) pairs. A “serviceAttribute” can be categorized based on custom classification schemes. A simple classification could be whether the “serviceAttribute” is prescriptive or descriptive. A service attribute may also correspond to a domain-specific metadata and could be directly related with functionality of the service. leaseEntity: A lease entity describes the lifetime associated with services or context. This entity indicates that the service or context will be considered alive and can be discovered by client applications until the lease expires.

B. WS-Context Data Model

Although WS-Context Specification presents XML API to standardize behavior and communication of the service, it does not define an information model. We introduce an information model comprised of various entities. Here, entities are represented in XML and stored by the WS-Context Service. The proposed information model composed of instances of the entities as following. *sessionEntity:* A session entity describes a period of time devoted to a specific activity, associated contexts, and services involved in the activity. A session can be considered as an information holder for the dynamically generated information. Each session is associated with its participant web services. Also, each session contains contexts which might be associated with either services or session or both. *contextEntity:* A context entity describes dynamically generated metadata that is associated either to a session or a service or both. *leaseEntity:* A lease entity describes a period of time during which a service or a context can be discoverable. A lease entity is associated to both session and context entities.

C. Extended UDDI and WS-Context Inquiry and Publication API Sets

We present extensions/modifications to existing WS-Context and UDDI APIs to standardize the additional capabilities of our implementation. We then integrate both extended UDDI and WS-Context API sets within a uniform programming interface: Hybrid WS-Context XML Metadata Web Service. The API sets of the hybrid service can be grouped as following: ExtendedUDDI Inquiry, ExtendedUDDI Publication, WS-Context Inquiry, WS-Context Publication, WS-Context Security and Publisher XML APIs.

1) Extended UDDI Inquiry API: We introduced various

APIs representing inquiries that can be used to retrieve data from extended UDDI Service as following: *find_service:* Used to extend the out-of-box UDDI find service functionality. The find_service API call locates specific services within the UDDI Service. It takes additional input parameters such as serviceAttributeBag, contextBag and Lease to facilitate additional capabilities of the proposed system. *find_serviceAttribute:* Used to find aforementioned serviceAttribute elements. The find_serviceAttribute API call returns a list of serviceAttribute structure matching the conditions specified in the arguments. *get_serviceAttributeDetail:* Used to retrieve semi-static metadata associated to a unique identifier. The get_serviceAttributeDetail API call returns the serviceAttribute structure corresponding to each attributeKey values specified in the arguments.

2) Extended UDDI Publication API: We introduce various extensions to UDDI Publication API set to publish and update semi-static metadata associated with service. *save_service:* Used to extend the out-of-box UDDI save service functionality. The save_service API call adds/updates one or more web services into the UDDI service. Each service entity may contain one to many serviceAttribute and/or one to many contextEntity elements and may have a life time (lease). *save_serviceAttribute:* Used to register or update one or more semi-static metadata associated to a web service. *delete_serviceAttribute:* Used to delete existing serviceAttribute element from the UDDI Service.

3) WS-Context Inquiry API: We introduce extensions to WS-Context Specification for both inquiry and publication functionalities. The extensions to WS-Context Inquiry API set are outlined as following: *find_session:* Used to find sessionEntity elements. The find_session API call returns a session list matching the conditions specified in the arguments.

get_sessionDetail: Used to retrieve sessionEntity data structure corresponding to each of the session key values specified in the arguments. *find_context:* Used to find contextEntity elements. The find_context API call returns a context list matching the criteria specified in the arguments. *get_contextDetail:* Used to retrieve the context structure corresponding to the context key values specified.

4) WS-Context Publication API: We outline the extensions to WS-Context Specification Publication API set to publish and update dynamic metadata as following: *save_session:* Used to add/update one or more session entities into the service. Each session may contain one to many serviceAttribute, have a life time (lease) and be associated with service entries. *delete_session:* Used to delete one or more sessionEntity structures. *save_context:* Used to add/update on or more context (dynamic metadata) entities into the service. *delete_context:* Used to delete one or more contextEntity structures.

D. Authentication Mechanism

In order to avoid unauthorized access to the system, we adopted semantics from existing UDDI Security XML API and implemented a simple authentication mechanism. In this scenario, each publication/inquiry request is required to include authentication information (authInfo XML element). Although this information may enable variety of authentication mechanisms such as X.509 certificates, for simplicity, we implemented a username/password based authentication scheme. A client can only access to the system if he/she is an authorized user by the system and his/her credentials match. If the client is authorized, he/she is granted with an authentication token. An authentication token needs to be passed in the argument lists of publication and inquiry functions, so that these operations can take place.

1) WS-Context Security API: We adopt the semantics from out-of-box UDDI Security API set in our design. The Security API includes following function calls. *get_authToken:* Used to request an authentication token as an "authInfo" (authentication information) element from the service. The autInfo element allows the system implement access control. To this end, both publication and inquiry API set includes authentication information in their input arguments. *discard_authToken:* Used to inform hybrid WSContext service that an authentication token is no longer required and should be considered as invalid.

E. Authorization Mechanism

When a context is published to the system, by default an owner-relationship is established between the publisher and the context. The owner of the context specify various permissions such as what access rights a) the owner, b) the members of the owner's group, and c) the rest of the users will have to the context. For each of these categories there exist read, write and read/write access rights. This basic security mechanism is also used in UNIX operating system. Upon receiving a request, the system checks access permission rights specified in a context, before granting inquiry/publication request to the context.

1) WSContext Publisher API: We introduce various APIs to provide find/add/modify/delete on the publisher list, i.e., authorized users of the system. These APIs include the following function calls. *find_publisher:* Used to find publishers registered with the system matching the conditions specified in the arguments. *save_publisher:* Used to add or update information about a publisher. *delete_publisher:* Used to delete information about a publisher with a given publisherID from the metadata service. *get_publisherDetail:* Used to retrieve detailed information regarding one or more publishers with given publisherID(s).

Given these capabilities, one can simply populate the hybrid service with metadata as in the following scenario. Say, a user publishes a new service into the system. In this case, the user constructs both "metadataBag" filled with "serviceAttributes" and "contextBag" filled with "contexts" where each context describes the sessions that this service will be participating. As both the "metadataBag" and "contextBag" is constructed, they can be attached to a new "service" element which can then be published with extended "save_service" functionality of the hybrid WS-Context XML Metadata Service. On

receiving publishing service metadata request, the system applies following steps to process service metadata. First, the system separates the dynamic and static portions of the metadata. Then, the system delegates the task of handling discovery of static portion ("metadataBag") to extended UDDI service. Next, the system itself provides handling and discovery using dynamic portions of the metadata in the metadata replica hosting environment. Further design documentation on both hybrid WS-Context and extended UDDI XML Metadata Services is available at http://www.opengrids.org.

IV. An Application Usage Scenario

In order to present the applicability of our system, we briefly outline a metadata storage component (the Context-store) of an application use domain (mobile environment) in which the proposed hybrid XML Metadata Service is used.

Description: A Context-store component is responsible for storing redundant/unchanging parts of messages used in service communication.

Requirements: Let's consider a user has a cell phone, which is running a videoconferencing application packaged as a "lightweight" Web Service. Such service could be a conferencing, streaming, or instant messaging service. To optimize service communication, the redundant/unchanging parts of the messages, exchanged between two services, must be stored on a third-party repository, i.e., Context-store.

Usage Scenario: The redundant/unchanging parts of a SOAP message are XML elements which are encoded in every SOAP message exchanged between two services. These XML elements can be considered as "context", i.e. metadata associated to a conversation. Here, hybrid WS-Context XML Metadata Service is being used as the Context-store [11]. Each context is referred with a system defined URI where the uniqueness of the URI is ensured by the system. The corresponding URI replaces the redundant XML elements in the SOAP messages which in turn reduce the size of the message for faster message transfer. Upon receiving the SOAP message, the corresponding parties in service conversation interact with WS-Context Service to retrieve the context associated with the URIs listed in the SOAP message.

V. An Overview Of The Prototype Implementation Of The Hybrid XML Metadata Service

We assume a range of applications which may be interested in integrated results from two different metadata spaces; UDDI and WS-Context. When combining the functionalities of these two technologies in one hybrid service, we may enable uniform query capabilities on context (service metadata) catalog. To this end, we have implemented a uniform programming interface, i.e. a hybrid information service combining both extended UDDI and WS-Context. (see Session 3 for detailed discussion on Information Model and XML API Sets of the hybrid service). Here, we give a brief overview of the system components, their functionalities and discuss how these components interact with each other.

Our implementation consists of various modules such as Query and Publishing, Expeditor, Access, Storage and Sequencer Modules. The Query and Publishing Module is responsible for performing operations issued by end-users. The Expeditor Module forms a generalized caching mechanism. One consults the expediter to find how to get (or set) information about a dataset in an optimal fashion. The Access and Storage modules are responsible for actual communication between the distributed XML Metadata Services in order to form a distributed replica hosting environment. In particular, the Access module deals with client request distributions, while the Storage module deals with replication. Finally, the Sequencer Module is used to label each metadata which will be stored in the system.

When receiving a query, the Query and Publishing Module first processes the query and extracts the dynamic metadata portion of the query. Then, it forwards the query to Expediter, where the Expeditor Module checks whether the requested data is in the cache.

The Expeditor Module implements a generalized caching mechanism and forms a built-in memory. It utilizes the TupleSpaces paradigm [12] which is a space based programming providing mutual exclusive access that in turn enables data sharing between processes. For the purposes of this research, a tuple is termed as context and the tuplespaces as ContextSpaces. The Expeditor Module implementation is built on MicroSpaces libraries [13]. MicroSpaces is a free, open-source, and a light-weight implementation of TupleSpaces paradigm. The MicroSpaces codebase is expanded in the following ways in order to incorporate with our implementation. First, a context management scheme is implemented to manage storage and dynamic replication decisions for the contexts stored in the ContextSpace. This is succeeded by implementing a Java Thread which is responsible for a) checking the ContextSpace for updates with frequent time intervals, b) storing updated contexts into MySQL database and c) deciding on dynamic replica placements. Second, an Expeditor Handler library is implemented in order to query/publish data in local database. An Expeditor handler allows processes to do operations on the ContextSpace as the primary storage. We employ a factory design pattern in implementing Handlers to communicate with the ContextSpace. In this design, a data store java class (ContextSpaceDataStore) is impelemented to enable processes to interact with the ContextSpace. In order to communicate with the data store, an instance of the data store has to be initiated from a factory java class called WSContextSpaceFactory. The Expeditor Module also contains an interface java class for a registry which defines all the primary functionalities that can be done on the data store. The registry interface is implemented by ExpeditorRegistryEngine java class which in turn enables processes to create objects to perform operations (without specifying the exact class name of the object that will be created). Once an instance of a given function is created by the ExpeditorRegistryEngine, the newly created object obtains

an instance of ContextSpaceDataStore from the data store factory in order to perform requested functionality.
If the Expeditor Module can not find the result, then the query is performed on the local MySQL database using JDBC Handlers. If the query asks for external metadata, then the Query and Publishing Module will forward the query to Access Module, where the Access Module multicast a probe message to available services through a messaging infrastructure which is based on publish/subscribe paradigm. We use Naradabrokering (NB) [14] software which is an open-source and distributed messaging infrastructure implementing publish/subscribe paradigm. This way the service communicates with the original data sources to satisfy the query under consideration. The query is responded by those services that host the matching context. At last, on receiving the results, the Query and Publishing Module returns the results to the querying client.

VI. PERFORMANCE EVALUATIONS

We have performed two application-specific experiments to investigate the performance of aforementioned XML Metadata Services. (General evaluations are extensively documented in [9]) First, we investigated the baseline-performance of both extended UDDI and hybrid WS-Context Services. Second, we analyzed the scalability of the hybrid WS-Context XML Metadata Service.

We tested the prototype implementation of the system by using a linux server (gf6.ucs.indiana.edu) and a desktop machine (kilimanjaro.ucs.indiana.edu) located at our facilities. We ran the XML Metadata Services on the linux server. The client applications for the both experiments were running on the windows machine. These experiments were performed separately on different dates. The server was equipped with Intel® Xeon™ CPU (2.40GHz), 2 GB RAM and ran Linux kernel 2.4.22. The desktop machine ran Windows XP and was equipped with Intel Pentium 4 CPU (3.4 GHz) and 1 GB RAM. We wrote all our code in Java, using the Java 2 Standard Edition. In the experiments, we used Tomcat Apache Server with version 5.5.8 and Axis software with version 1.2beta3 as a service deployment container.

In the first experiment, we investigated three different testing cases: a) a single client publishes metadata to a hybrid WSContext Service, b) a client publishes metadata to an extended UDDI Service, and c) a client publishes metadata to a dummy service where the round trip message is extracted to and from container but no processing is applied. At each testing case the client sends 100 sequential publication requests and average response time was recorded. We repeated these tests in five different test sets. We used ~1.1KByte-size metadata in the test cases. The result of this experiment is depicted in Figure-2.

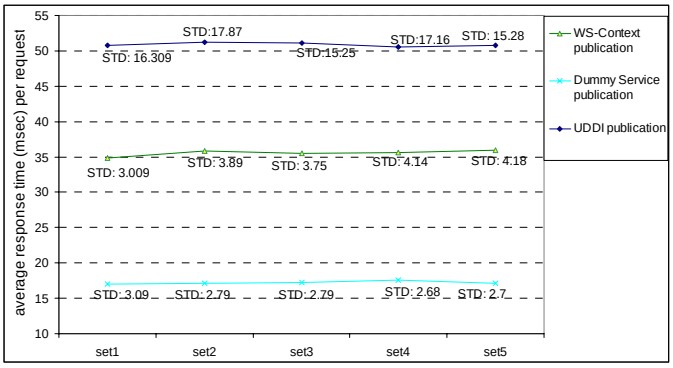

Figure 2. Average Response Time Graph for Publication Requests.

In the second experiment, we investigated the scalability of the system as a Context-store. As mentioned earlier, a Context-store is used for optimizing Web Service communication performance in mobile environment. In this experiment, a single client sends 100 sequential publication requests with varying message sizes to a hybrid WS-Context Service. We measured the time to finish a Context-store request message processing (i.e. T_{axis} + T_{wsctx}) and a time to process setContext() operation (i.e. T_{wsctx}). The result of this experiment is depicted in Figure-3.

Based on the results from first experiment (see Figure-2), we observe that hybrid WS-Context Service publication function performed with 30% performance increase compared to UDDI-publication function. The hybrid WS-Context Service employs a built-in cache mechanism for primary storage which in turn improves the performance. We also observe that the network latency is considerably high, although the context data size is very small and the performance measurements were taken on a tight cluster. However, in a wide area network, one could expect the network latency to be a bottleneck for system performance.

Figure-3 shows the T_{axis} + T_{wsctx} time compared with T_{wsctx} time. We observe that T_{axis} + T_{wsctx} increase linearly while Twsctx does not change as the size of context increases (This is of course for the context size ranging from 1.2 to 2.2 KByte). The overhead of the Axis container includes the time spent for sending and receiving a SOAP message as well as the time for XML processing. Based on the results, we observe that as the context size gets increased, the container overhead is also increased. Thus we think that if the time consumed for container processing (i.e. T_{axis}) was reduced, the throughput would increase. Let's consider a mobile-environment scenario in which the two web services are communicating via SOAP message exchanges (within a session) and utilizing a Context-store for optimization.

- Let N to be number of simultaneous sessions
- Let T_{wsctx} to be the time to process setContext() operation
- Let T_{axis} to be the time spent by Axis container
- Let $T_{session}$ to be the length of a given session

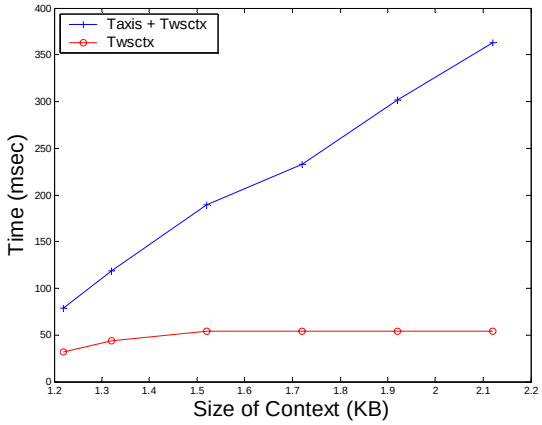

Figure 3. Comparison between (T_{axis} + T_{wsctx}) and T_{wsctx}

Based on the optimized communication model discussed in [11], each participant Web Service makes one access to the Context-store at the beginning of a session. Once the session is completed, another access is made by one of the participants to finalize the session. (There are two services exchanging messages. Thus we have three Context-store accesses per session.) Let's consider N simultaneous sessions happening during the time period of $T_{session}$. One can conclude that, per second, there exist $\frac{3N}{T_{session}}$ times Context-store accesses for N simultaneous sessions. A publication access to the Context-store takes (T_{axis} + T_{wsctx}) execution time. Thus we can formulize the calculation of maximum number of supported simultaneous sessions as following:

$$3N \times (T_{wsctx} + T_{axis}) \le T_{session} \qquad N \le \frac{T_{session}}{3 \times (T_{wsctx} + T_{axis})} \qquad (1)$$

Provided with the formula at (1) and our measurements from second experiment (see Figure-3), we can calculate the maximum number of simultaneous sessions supported by the Context-store. Let's say we have a session with a length of 10 minutes i.e. $T_{session}$ = 600 seconds. Then, using the formula and T_{axis} + T_{wsctx} time (from Figure-3) for a 1.2 KByte-size context, we can calculate the number of maximum simultaneous sessions as following.

$$N \le \frac{600\,\text{sec}}{3 \times (0.079\,\text{sec})} \qquad N \le 2532$$

Thus if we have 1.2 KByte-size contexts, the Context-store can support maximum 2532 sessions in mobile-environment optimized service communication. Please note that this illustration only indicates the optimal upper-bound (not the practical case) for a small XML message. Furthermore, this is based on the assumption that Tomcat server and the Context-store can handle this many simultaneous connections.

VII. Conclusion and Future Work

We examined XML Metadata Services as an important tool to knowledge and information grids. We focused on the semantics and identified the base elements of the architecture: data semantics and semantics for XML API sets such as publication, inquiry, security and access control. With this identification made, we discussed our approach and experiences in designing "semantics" for XML Metadata Services. Also, we outlined a real-life application use scenarios to identify a way and reason of using XML Metadata Services in Grids.

We implemented centralized versions of both extended UDDI and hybrid WS-Context XML Metadata Services as open-source software which have been used successfully in varying types of Grids: collaboration, earth science and so forth. We briefly discussed the prototype implementation of the proposed approach. We are currently working on implementing fault tolerance by using replication as a technique. We plan on investigating scalability and performance limitations of the system when it is decentralized and comprised of widely distributed nodes.

Acknowledgement: This work was supported in part by the U.S. National Aeronautical and Space Administration's Advanced Information Systems Technology program. The authors would like to thank Prof. Gordon Erlebacher for his critique on the WS-Context project and Community Grids Laboratory graduate research assistants who have been using XML Metadata Services in their applications.

References

[1] Zhuge, H., China's E-Science Knowledge Grid Environment, IEEE Intelligent Systems, 19 (1), (2004) 13-17

[2] Bellwood, T., et al. UDDI Version 3.0.1: UDDI Spec Technical Committee Specification. Available from http://uddi.org/pubs/uddi-v3.0.1-20031014.htm.

[3] Bunting, B., et al. K. Web Services Context (WS-Context), available from http://www.arjuna.com/library/-specs/ws_caf_1-0/WS-CTX.pdf

[4] V. Dialani. UDDI-M Version 1.0 API Specification. University of Southampton – UK. 02.

[5] Ali ShaikhAli, el al. UDDIe: An Extended Registry for Web Services. Proc. of the Service Oriented Computing: Models, Architectures and Applications, SAINT-2003 IEEE Comp. Society Press., USA

[6] Czajkowski, K., et al. 2004. The WS-Resource Framework. http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[7] Alexander, J., et al. 2004 The Web Service Transfer (WS-Transfer) http://msdn.microsoft.com/library/en-us/dnglobspec/html/wstransfer.pdf

[8] Ballinger, K., et al. 2004 The Web Services Metadata Exchange http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[9] Aktas, M. S., Fox, G. C., Pierce, M. Fault Tolerant High Performance Information Services for Dynamic Collections of Grid and Web Services FGCS Special issue from SKG2005 Beijing China November, 2005.

[10] Aktas, M. S., Fox, G. C., Pierce, M., Managing Dynamic Metadata as Context. The 2005 Istanbul International Computational Science and Engineering Conference (ICCSE2005), Istanbul, Turkey.

[11] Oh, S., Aktas, M. S., Pierce, M., Fox, G. C., Optimizing Web Service Messaging Performance Using a Context Store for Static Data, 5th WSEAS Int. Conf. on Telecommunications and Informatics, Turkey, 05

[12] Gelernter, N. C. a. D. (1989). "Linda in Context." Commun. ACM, 32(4): 444-458.

[13] Coleman, R., et al., MicroSpaces software with version 1.5.2 available at http://microspaces.sourceforge.net/. May 2004.

[14] Fox, S. P. a. G. (2003). NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro, Brazil.