XML Metadata Services

Mehmet S. Aktas[1,2], Sangyoon Oh [1,2], Geoffrey C. Fox[1,2,3], Marlon E. Pierce[1]

[1]Community Grids Laboratory
[2]Department of Computer Science
[3] Department of Physics
Indiana University
{maktas, ohsangy, gcf, mpierce}@cs.indiana.edu

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 a hybrid Information Service 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". We report results from a prototype of the system that is applied to mobile environment for optimizing Web Service communications.

1. 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.

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 a hybrid 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 approach and give an overview of implementation. In addition, we discuss a motivating application scenario and the way that the hybrid system is being used. We report results from a prototype that has been applied to mobile environment for optimizing Web Service communications.

2. Background

We broadly classify service metadata into two categories: interaction-dependent and interaction-independent. Interaction-dependent service metadata is the session metadata generated by one or more services as a result of their interactions. Interaction-independent service metadata is rarely changing information describing the characteristics of services.

Another way of classifying service metadata could be based on its characteristics such as prescriptive (functional) and descriptive (non-functional). The prescriptive characteristics are directly related with functionality of the service. For instance, the Open Geographical Information Systems (GIS) Consortium defines standards for prescriptive characteristics of services as an auxiliary capability file defining the data coverage of geospatial services. The descriptive characteristics are the non-functional properties associated with services. The non-functional properties of services may include availability (such as temporal, spatial availability), service quality (such as throughput, number of max supported clients), security and so forth.

Locating resources of interest is a fundamental problem in resource intensive environments. An effective methodology to facilitate resource discovery is to provide and manage information about resources. Here, a resource corresponds to a service and information associated to it refers to metadata of a service. Thus, we see a greater need for metadata management solutions to make such metadata available in peer-to-peer/grid environments. Having identified the two-metadata types, that is interaction-independent and interaction-dependent; we survey standardizations for metadata management under two categories: a) managing interaction-independent service metadata and b) managing interaction-dependent service metadata.

2.1. Managing interaction-independent service metadata

As the Service Oriented Architecture (SOA) principles [4] have gained importance, an emerging need has appeared for standardization of XML metadata services that provide programming interface to access and manipulate service metadata. Here, we investigate the existing specifications/ standardizations defining the interaction-independent service metadata. In our investigation, we mainly focus on metadata requirements of Geographical Information Systems (GIS), as they provide very useful problems in supporting "virtual organizations" and their associated information systems.

Web Registry Services: The Web Registry Service [5], introduced by the Open GIS Consortium (OGC) [6] is an approach to standardize the metadata management problem particularly for GIS domain. The OGC is an international organization providing specifications to integrate geospatial data and geo-processing resources into mainstream computing. It leads efforts to provide a) standardized protocols for accessing geospatial information and services and b) standardized service metadata such as "capabilities.xml" documents. The OGC introduced a) the Catalog Specification [7] and b) the Web Registry Service (WRS) Specification [5]. The OGC Catalog Specification is an abstract specification, which was introduced to create a conceptual model to allow the creation of implementation specifications for discovery and retrieval of metadata that describes geospatial data and geo-processing services. The Web Registry Service (WRS) Specification is an implementation specification of the OGC Catalog Specification, which was introduced to define a standard way to discover/publish service information of geospatial services and presents a domain-specific registry capability for geospatial information. The WRS Specification adopts the OGC Registry Information Model, which is based on the ebXML registry information model (ebRIM) [8, 9]. The WRS Specification uses ebRIM to support/integrate service entries with metadata and provide metadata management for geospatial domain. An example prototype [10] of the WRS Specification is implemented by LAITS group in George Mason University. This prototype is primarily based on Metadata Catalog Service (MCS) [11], a stand-alone metadata catalog service with an Open Grid Service Architecture (OGSA) [12] service interface. The prototype implementation provides a mapping between the OGC Registry Information Model and the MCS data model.

The WRS approach is limited to the GIS domain. As it was designed as a GIS domain-specific solution, it supports neither the information model nor the programming interface that could facilitate a generic metadata management.

The Universal Description, Discovery, and Integration (UDDI): The UDDI Specification is the most prominent and widely used standard that is based on a XML-based protocol that provides a directory and enables services advertise themselves and discover other services. UDDI is domain-independent standardized method for publishing/discovering information about Web Services. It offers users a unified and systematic way to find service providers through a centralized registry of services. As it is WS-Interoperability (WS-I) [13] compatible, UDDI has the advantage being interoperable with most existing Grid/Web Service standards.

We observe that the adoption of UDDI Specification in various domains such as Geographical Information Systems is slow, since existing UDDI specification has following limitations. First, UDDI introduces keyword-based retrieval mechanism. It does not allow advanced metadata-

oriented query capabilities on the registry. Second, UDDI does not take into account the volatile behavior of services. Since Web Services may come and go and information associated with services might be dynamically changing, there may be stale data in registry entries. Third, since UDDI is domain-independent, it does not provide domain-specific query capabilities such as geospatial queries. Thus, UDDI should be extended to overcome these limitations.

OGC use of UDDI Registries: In order to remedy some of these limitations, various solutions have been introduced. For an example, OGC has proposed a set of design principles, requirements and spatial discovery methodologies for discovery of OGC services through UDDI interface [14]. The proposed methodologies have been implemented by various organizations such as Sycline [15] and Galdos [16]. The Syncline experiment focuses on implementing a UDDI discovery interface on an existing OGC Catalog Service data model so that UDDI users can discover services registered through OGC Registries. The Galdos experiment focuses on turning OGC Service Registry into a UDDI node by utilizing JAXR API to map UDDI inquiry interface to the OGC Registry Information Model [14]. Briefly, these methodologies showed that it is possible to do spatial discovery and content discovery through UDDI Specification.

Existing UDDI approaches by OGC community are designed for and limited to geospatial specific usage. Services such as the Web Map and Web Feature service, because they are generic, must provide additional, descriptive metadata, such as Quality of Service attributes, in order to be useful. OGC approach does not define a data model rich enough to capture descriptive metadata that might be associated with service entries. That is, their approach does not put the descriptive metadata in the UDDI Registry. Therefore, it is still an open problem how to make these geospatial services distinguishable from others based on their qualities. Thus, we see the need for extensive metadata-oriented query capabilities in addition to geospatial query capabilities. We also note that the discovery methodologies (introduced by OGC community) extend the UDDI interface; however, they do not introduce an extension to existing UDDI information model.

UDDI-Extensions: The UDDI-M [17] and UDDIe [18] 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. UDDI-MT [19, 20] improves the metadata representation from attribute name-value pairs into RDF triples. A similar approach to leverage UDDI Specification was introduced by METEOR-S [21] project which identifies different semantics when describing a service, such as data, functional, quality of service and executions. Another approach, Grimories [22] is also an implementation of UDDI Specification. The Grimories Registry extends the functionalities of UDDI to provide a semantic enabled registry designed and developed for the MyGrid project [23]. It supports third-party attachment of metadata about services. The Grimories represents all published metadata in the form of RDF triples and allows the published metadata reside either in a database, or in a file, or in a memory. These approaches have investigated a generic and centralized metadata service focusing on the domain-independent metadata management problems. However, these solutions, as they are generic, do not solve the domain-specific metadata management problems. (How can registries facilitate geo-spatial queries on a metadata catalog for GIS domain?) We note that, the Grimories approach utilizes a caching mechanism for the RDF triple store. Although, this is a promising approach, we find following limitations.

Firstly, the performance of the system is bounded by the performance of the triple store (The Grimories uses the Jena software toolkit to operate on the RDF triple store. Thus, the limitations of Jena implementation may cause a performance bottleneck). Secondly, if the memory was chosen as the primary storage, the Grimories registry would give away from persistency as the snapshots of memory is not backed-up by the system. If the database was chosen as the primary storage, the system would give away from performance, as the system has to make disk access to publish a metadata. Thirdly, the Grimories's memory built-in storage does not provide mutual exclusive access to the shared data.

In our approach, we investigated methodologies, compatible with widely used standards, for discovering services based on both general and domain-specific search criteria. An example for domain-specific query capability is Xpath and RDQL queries on the auxiliary and domain-specific metadata files stored in the UDDI Registry. Another distinguishing aspect of our investigation is the support for session metadata. We took as a requirement that our hybrid Information Service should support not only quasi-static, stateless metadata, but also more extensive metadata requirements of interacting systems. Similar to existing solutions (UDDI-M and UDDIe), our design uses name-value pairs to describe characteristics of services and extend UDDI's Information Model to associate metadata with service descriptions. This approach has its own merits in the simplicity of design and implementation. Our system explored a caching mechanism that would provide persistency, performance and data sharing capabilities all together. UDDI-MT, METEOR-S, Grimories are example projects that utilize semantic web languages to provide better service matchmaking in retrieval process. This research has been investigated [19-22] and so not covered in our investigation. We view dynamic and domain-specific metadata requirements of sensor/GIS and collaboration Grids as higher priority.

2.2. Managing interaction-dependent service metadata

Often Web Services are assembled into short-term service collections that are gathered together into a meta-application (such as a workflow) and collaborate with each other to perform a particular task. For example, an airline reservation system could consist of several Web Services, which are combined together to process reservation requests, update customer records, and send confirmations to clients. As these services interact with each other they generate session state which is simply a data value that evolves as result of Web Service interactions and persists across the interactions. As the applications, employing Web Service oriented architectures, need to discover, inspect and manipulate state information in order to correlate the activities of participating services, an emerging need appeared for the technologies and specifications that would standardize managing distributed session state information. We can broadly classify existing solutions that define the stateful interactions of Web Services under two categories: a) point-to-point and b) centralized.

Point-to-Point methodologies to enable service communication: Point-to-point methodologies provide service conversation with metadata from the two services that exchange information. There are varying specifications focusing on point-to-point service communication, such as Web Service Resource Framework (WSRF) [24] and WS-Metadata Exchange (WS-ME) [25]. WSRF specification, which is proposed by Globus alliance, IBM and HP, defines conventions for managing state, so that collaborating applications can discover, inspect, and interact with

stateful resources in standard and interoperable ways. The WS-ME provides a mechanism a) to share information about the capabilities of participating Web Services and b) to allow querying a WS Endpoint to retrieve metadata about what to know to interact with them. Point-to-point methodologies provide service conversation with metadata only from the two services that exchange information.

Centralized methodologies to enable service communication: Communication among services can be achieved with a centralized metadata management strategy. The Web Services Context Specification (WS-Context) [3] is a promising example of this trend. It was introduced as a part of the Web Services Composite Application Framework (WS-CAF) [26] which is a suite of three specifications, WS-Context, WS-Coordination Framework (WS-CF) [27], and WS-Transaction Management (WS-TXM) [28]. The WS-Context defines a simple mechanism to share and keep track of common information shared between multiple participants in Web Service interactions. WS-CF defines a coordinator to which Web Services are registered to ensure messages and results are communicated correctly. The coordinator provides the notification of outcome messages to Web Services participating in an activity. WS-TXM defines three distinct transaction protocols: two phase commit, long running actions, and business process flows. These are used in the coordination framework to make existing transaction managers interoperable. The three specifications comprise a stack of functionality [26]. WS-Context is at the bottom and adding WS-CF and then WS-TXM.

The WS-Context is a lightweight storage mechanism, which allows the participant's of an activity to propagate and share context information. It defines an activity as a unit of distributed work involving one or more parties (services, components). In order an activity to extend over a number of Web Services, certain information has to flow among the participant of application. This specification refers such information as context and focuses on its management. The WS-Context Specification defines three main components: a) context service, b) context, and c) an activity lifecycle service. The context service is the core service concerned with managing lifecycle of context propagation. The context defines information about an activity and is referenced with a URI. It allows a collection of actions to take place for a common outcome. For an example, a participating application can discover results of other participants' execution, which is stored as context. The minimum required context information (such as the context URI) is exchanged among Web Services in the header of SOAP messages to correlate the distributed work in an activity. This way, a participant service obtains the identifier and makes a key-based retrieval on the context service. Thus, a typical search with the WS-Context is mainly based on key-based retrieval/publication capabilities. The activity of lifecycle service defines the scope of a component activity. Note that, activities can be nested. An activity may be a component activity of another. In this case, additional information (such as security metadata) to a basic context may be kept in a component service, which is registered with the core context service and participate in the lifecycle of an activity.

The WS-Context and UDDI introduce two different ways of managing service metadata. The WS-Context defines a standard way of maintaining distributed session state information associated to participating services. The UDDI is a standard way of publishing/discovering generic information associated to Web Services. Therefore, the two-metadata management solutions – UDDI and WS-Context – are comparable, as they, both deal with service metadata.

Firstly, the UDDI is concerned with the interaction-independent metadata space. The interaction-independent metadata is rarely changing information describing functional or non-functional properties of Web Services. On the other hand, the WS-Context is concerned with the interaction-dependent metadata space. The interaction-dependent metadata is highly updated and dynamic information describing information associated to Web Service activities. Thus, the two-metadata services define different functionalities to meet the requirements of the two different metadata domains. Secondly, in the WS-Context approach, the members of an activity should be notified of the distributed state information such as when it is created or deleted. This way, the dynamism in the metadata is captured by the participating services of the activity. However, in the UDDI approach, the interaction-independent metadata is rarely changing and may not necessarily require a notification mechanism. Thirdly, the WS-Context is intended for activities that are comprised of modest number interacting Web Services. However, the UDDI is intended for the whole Grid. Thus, the UDDI requires a degree of complexity in inquiry operations to improve the selectivity and increase the recall and precision in the search results. Fourthly, the WS-Context is intended to correlate activities of Web Services that participate to an activity. Thus, it supports loose coupling of services by employing synchronous callback facilities. However, the UDDI is a synchronous Web Service and provides an immediate response to a query. Fifthly, the WS-Context approach should be lightweight for allowing multiple Web Services to share a common context. Thus, it requires high performance and scalability in numbers for concurrent accesses. However, in the UDDI approach, Web Service metadata entry can only be updated by its publisher and is not shared, thus concurrency is not a high priority.

We find various limitations in WS-Context Specification in supporting stateful interactions of Web Services. First, the context service, a component defined by WS-Context to provide access/storage to state information, has limited functionalities such as the two primary operations: GetContext and SetContext. However, traditional and Semantic Grid applications present extensive metadata needs which in turn requires advanced search/access/store interface to distributed session state information. Second, the WS-Context Specification is only focused on defining stateful interactions of Web Services. It does not define a searchable repository for interaction-independent information associated to the services involved in an activity. However, there is a need for a specification, which can provide an interface not only for stateful metadata but also for the stateless, interaction-independent metadata associated to Web Services. Third, the WS-Context defines a centralized context service for managing dynamically propagated context. This in turn creates a single-point of failure problem and performance bottleneck for a given activity, as the number of its participants may get increased and be widely distributed. However, there is a need for fault-tolerant, high-performance and decentralized context service architecture for workflow activities in which the participants are widely distributed.

Among the existing specifications, which standardize service communications, we believe that the WS-Context Specification is the most promising to tackle the problem of managing distributed session state. Unlike the other service communication specifications, WS-Context models a session metadata repository as an external entity where more than two services can easily access/store highly dynamic, shared metadata. Thus, we took as a design requirement that the system should employ/extend WS-Context Specification to manage dynamically generated session metadata. In order to remedy the limitations of WS-Context, the proposed approach supports a fault-tolerant, high-performance hybrid XML Metadata Service that can be used as the

context service and provide a uniform programming interface to both stateless, interaction-independent and stateful, interaction-dependent service metadata.

3. Abstract Data Models

We have designed and built a novel architecture [29, 30] for an hybrid Information 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).

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 so that it can easily be operated with future releases of these specifications.

3.1. Extensions to UDDI Abstract 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.

3.2. WS-Context Abstract 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 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.

3.3. 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 Grid/Web Information 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.

3.3.1. Extended UDDI Inquiry API

We introduced various APIs representing inquiries that can be used to retrieve data from the hybrid 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 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.

3.3.2. 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 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 service.

3.3.3. 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.

3.3.4. 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.

3.4. 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.

3.4.1. 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.

3.5. 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.

3.5.1. WS-Context 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 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 issues the static portion ("metadataBag") of the query on the extended UDDI MySQL database, while it issues dynamic portion ("contextBag") of the query on the WSContext MySQL database. Further design documentation on hybrid XML Metadata Service is available at http://www.opengrids.org.

4. An Application Usage Scenario: Handle Flexible Representation (HHFR) System

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 system is used.

4.1. An Overview of the Service Oriented Architecture for HHFR System

A novel Web Service architecture, Handheld Flexible Representation (HHFR) is developed for optimizing Web Service performance in mobile computing which is physically constrained and requires an optimized messaging scheme to prevent performance degradations. Despite its important role in distributed computing, mobile computing hasn't reached its full potential because of the limited availability of high speed wireless connections (e.g. third generation cellular technology) as well as shortened the device's use time when it is connected to a faster channel and do more computation. Thus, applying current Web Service communication models to mobile computing may result in unacceptable performance overheads caused by the encoding and decoding of verbose XML-based SOAP messages.

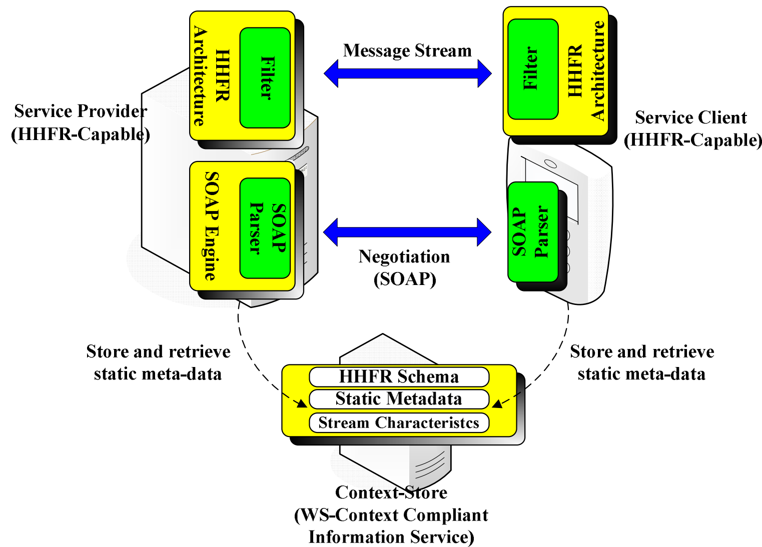

Figure 1 Overview of HHFR Architecture

HHFR let two end-points exchange information in a form of a message stream which is achieved on both semantic and protocol level by which mobile devices are able to overcome possible communication overheads caused by many factors: not only encoding/decoding overhead stated above, but also high latencies of mobile connection using HTTP. It provides a framework to negotiate characteristics of the message stream and representation between a mobile application and a corresponding service node.

Message streaming on semantic level is achieved by two methods. First, HHFR provides a scheme to separate the semantics of a message and its representation. Using Data Format Description Language (DFDL)-style data description language, HHFR describes the data format in Simple_DFDL language and the stub in the framework converts data from and to a preferred representation. The other method used by HHFR for building a message stream is storing unchanging/redundant message parts to hybrid information service (i.e. Context-store).

Let's consider a scenario where 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.

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 XML Metadata Service is being used as the Context-store [31]. 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 interacts with the hybrid service to retrieve the context associated with the URIs listed in the SOAP message.

4.2. Usage of the Context-store

The Context-store archives the static context information from a SOAP negotiation message, such as unchanging or redundant SOAP headers, a Simple_DFDL document as a message representation, and the character of the stream. By archiving, the Context-store can serve as a meta-data repository for the participating nodes in the HHFR architecture (i.e. database semantic). The hybrid metadata service is essential to HHFR architecture since the service provides a method to store message parts and sharing them between messages in the stream makes them related each other and ultimately reduce the size and complexity of them. The Context-store in the architecture also guarantees a semantically persistent recovery from disconnections which is more common in intermittent mobile domain than conventional computing domain.

We integrate a hybrid metadata service with HHFR through a direct serialization of the SOAP request message and a parsing SOAP without Axis SOAP-Java binding. It is because Axis version for mobile environment is not developed yet. We use the kSOAP library for those processes.

4.3. Performance Improvements in HHFR with the Context-store

To show the effectiveness of using the hybrid metadata service (i.e. Context-store), we measured a round trip time of both the full SOAP message and the optimized message with Context-store usage. As stated, the size of the message can be reduced and the performance of the messaging can also be increased.

The experiment uses various sizes of SOAP example documents as message; the round trip times for a large message are collected using a WS-Security headers and a sample message with WS-Addressing headers is used for a medium size. In this experiment, the messages between mobile devices and services are exchanged over HHFR. The result show we save 83% of message size on average and 41% of transit time on average by using the Context-store. The results are shown in Table 1 and the configuration of the mobile devices and the service provider machine is given in Table 2 and Table 3 respectively.

Message Size	Full SOAP Message		Optimized Message	
	Ave.±error	Stdev	Ave.±error	Stdev
Medium: 513byte (sec)	2.76±0.034	0.187	1.75±0.040	0.217
Large: 2.61KB (sec)	5.20±0.158	0.867	2.81±0.098	0.538

Table 1 Summary of the Round Trip Time

Service Client: Treo 600	
Processor	ARM (144MHz)
RAM	32MB total, 24MB user available
Network Bandwidth	14.4Kbps (Sprint PCS Vision)
OS	Palm 5.2.1.H
Java Version	Java 2, Micro CLDC 1.1, MIDP 2.0

Table 2 Summary of Mobile Device Configuration

Service Provider: Grid Farm 8	
Processor	Intel® Xeon™ CPU (2.40GHz)
RAM	2GB total
Network Bandwidth	100Mbps
OS	GNU/Linux (kernel release 2.4.22)
Java Version	Java 2 platform, (1.5.0-06)

Table 3 Summary of Service Provider Machine Configuration

5. 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 an 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 Hybrid 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. 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 [32] 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 [33]. 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. As the system keeps all metadata keys in memory, if the Expeditor Module can not find the result, 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) [34] 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.

6. EVALUATION

We have performed experiments to investigate the performance and scalability of aforementioned hybrid service.

Machine Configurations	
Processor	Intel® Xeon™ CPU (2.40GHz)
RAM	2GB total
Network Bandwidth	100Mbps
OS	GNU/Linux (kernel release 2.4.22)
Java Version	Java 2 platform, (1.5.0-06)

Table 4 Summary of Linux Machine Configurations

We tested our code using various nodes of a cluster located at the Community Grids Laboratory of Indiana University. This cluster consists of eight Linux machines that have been setup for experimental usage. The cluster node configuration is given at Table 5. 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 2 as a service deployment container.

Firstly we applied a performance experiment. The primary interest in doing this experiment is to understand the baseline performance of the implementation of hybrid Information Service. The performance evaluation of the service is done for inquiry and publication functions under normal conditions, i.e., when there is no additional traffic.

In this experiment, we particularly investigate performance of our caching methodology for WS-Context standard operations. We conduct following testing cases: a) A client sends queries to a dummy service. The dummy service receives a message and then sends it back to the client with no processing applied. b) A single client sends inquiry/publication requests to a hybrid Information Service where the system grants the request with memory access. c) A single client sends inquiry/publication requests to a hybrid Information Service where the system grants the request with database access.

This experiment studies the effect of various overheads that might affect the system performance. To do this, a dummy service, which is simply an echo service that returns the input parameter passed to it, is being used. This service helps measuring various overheads such as the network communication, client application initialization and container processing. By comparing and contrasting the results from the dummy service and the actual hybrid service, the actual time spent for pure server side processing can be observed. In this experiment, we use the same Web Service container engine (Apache Axis with version 2) for all testing cases.

In our investigation of system performance, we conducted the testing cases when there were 5000 metadata published in the system. At each testing case, the client sends 200 sequential requests for either publication or inquiry purposes. We record the average response time. This experiment was repeated five times. Figure 3 illustrates the system performance when the inquiry function was executed, while Figure 4 illustrates the same when the publication function was executed. The detailed statistics corresponding to these testing cases is listed in Table 6 and Table 7.

We also conduct an experiment where we investigate the best possible backup-interval period to provide fault-tolerance and high performance at the same time. Based on this experiment, we observe the trade-off in choosing the value for backup-time-interval. If the backup frequency is too high such as every 10 milliseconds, then the time required for a publication function is around 10.2 milliseconds. If the backup frequency is every 10 seconds or lower, we find that average execution time for publication operation stabilized to 7.46 milliseconds. Therefore, we choose the value for backup frequency as every 10 seconds in our experiments.

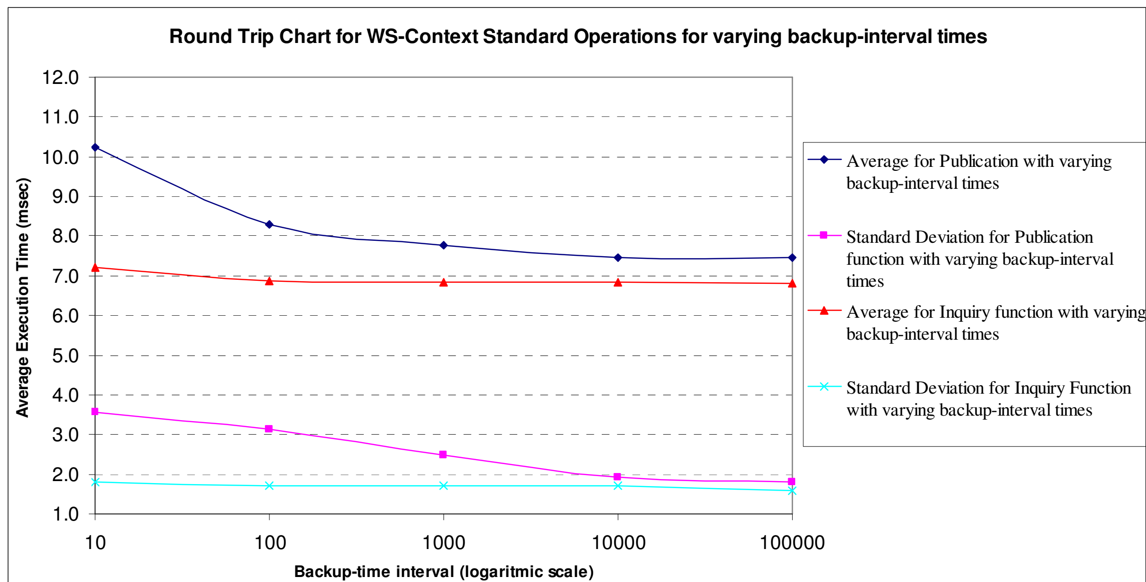

Figure 2 Test results for backup frequency investigation

	Publication Function		Inquiry Function	
Kbytes	mean	stdev	mean	stdev
0.01	10.24	3.57	7.20	1.80
0.1	8.29	3.13	6.86	1.71
1	7.76	2.48	6.85	1.70
10	7.46	1.94	6.85	1.71
100	7.46	1.82	6.81	1.60

Table 5 Statistics for the Figure 2

Figure 3 shows the performance results of inquiry function, while Figure 4 shows the performance results of publication function. The empirical results show that a) for inquiry function, we gain around 47% performance increase and b) for publication function, we gain around 30% performance increase by employing a cache mechanism in our design. This experimental study indicates that one can achieve noticeable performance improvements in metadata management for standard inquiry and publication operations by simply employing a memory built-in caching mechanism (the Expeditor Module), while preserving a certain fault-tolerance level as the contexts have to be backed up offline in at most N time unit. Based on our investigation on backup frequency, we choose the value of N to be 10 seconds.

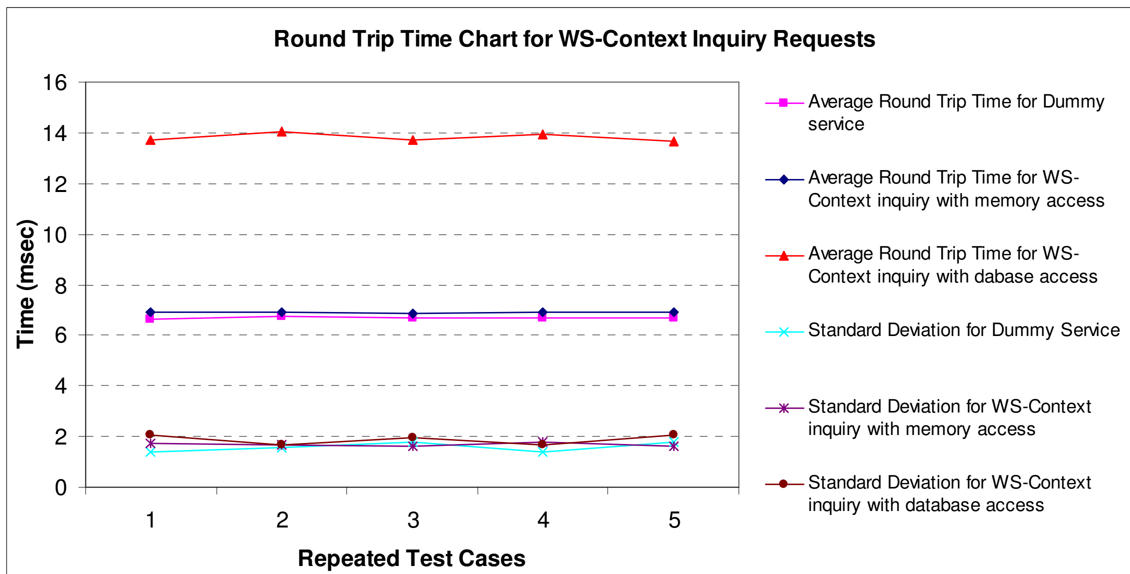

Figure 3 Round Trip Time Chart for Inquiry Requests

Statistics for the first test set from different publication request testing cases		
	mean	stdev
Test-1	6.64	1.40
Test-2	6.92	1.70
Test-3	13.73	2.08

Table 6 Statistics for the first test set from each testing cases conducted to test inquiry operation performance. (Test-1: Dummy service testing case, Test-2: WS-Context inquiry with memory access testing case, Test-3: WS-Context inquiry with database access testing case). The time units are in milliseconds.

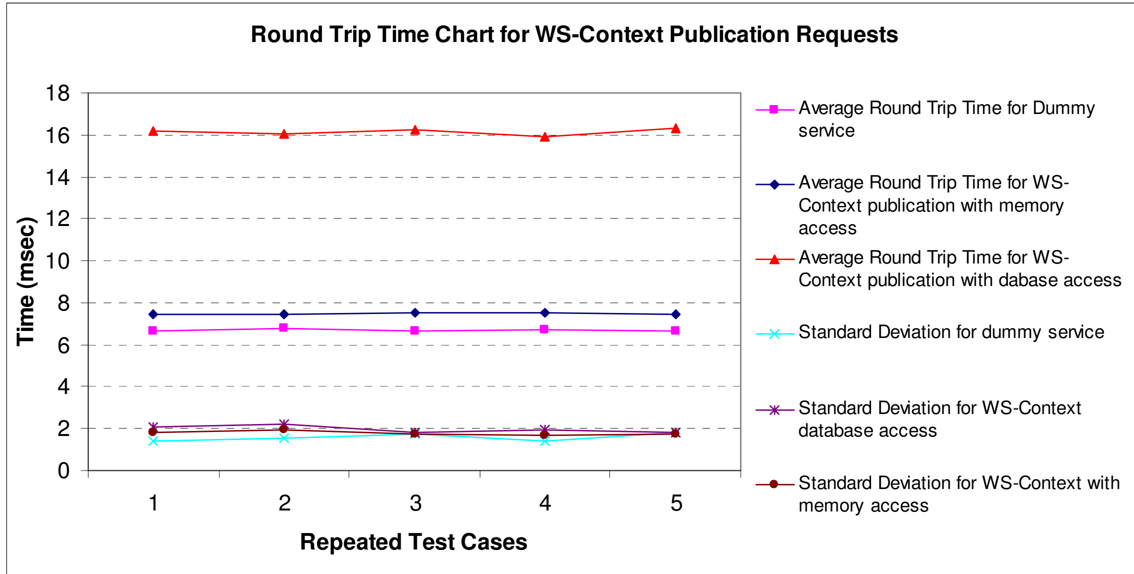

Figure 4 Round Trip Time Chart for Publication Requests

Statistics for the first test set from different publication request testing cases		
	mean	stdev
Test-1	6.64	1.82
Test-2	7.46	1.40
Test-3	16.19	2.06

Table 7 Statistics for the first test set from each testing cases conducted to test publication operation performance. (Test-1: Dummy service testing case, Test-2: WS-Context publication with memory access testing case, Test-3: WS-Context publication with database access testing case.) Time units are in milliseconds.

By comparing the results of inquiry and publication functions when the request is granted with database access, we observe that publication again requires more time compared to inquiry. This performance difference is the effect the database commit that must take place for publication operations.

Secondly, we conducted two testing cases on the system to investigate the scalability to answer following two questions: a) how well does the system perform when the context payload size gets increased, b) how well does the system perform when the message rate per second gets increased.

In the first testing case, our goal is to quantify the degradation in response time when contexts, with larger sizes, published/retrieved into/from the hybrid Service. We have done this by increasing the context sizes until the response time degrades. In this experiment round trip time was recorded at each inquiry/publication request message. The results are depicted in Figure 5 and Figure 6. The detailed statistics corresponding to this experiment is listed in

Table 8 and Table 9.

In the second testing case, we want to determine how well the number of users anticipated can be supported by the system for constant loads. Our goal is to quantify the degradation in response time at various levels of simultaneous users. In order to understand such performance degradation, we evaluate standard hybrid Service functionalities with additional concurrent traffic. We have done this by ramping-up the number of messages sent per second until the system performance degrades. In this testing case, messages were sent of randomly within a second. We again recorded the round trip time at each inquiry/publication request message and applied this test for both publication and inquiry standard operations. The results are depicted in Figure 7. The detailed statistics are given in Table 10.

Based on the results, we note that WS-Context standard operations performed well for small-size context payloads. For example, Figure 5 indicates that the cost of inquiry and publication operations remains the same, as the context's payload size increases from 100Bytes up to 10KBytes. Figure 6 indicates the system behavior for the message sizes between 10Kbytes and 100Kbytes. Based on these results, we observe a linear increase for the time required to complete a publication operation. By comparing the results from a dummy service, which returns the same size message with no processing applied, and hybrid service, we observe that the pure server processing time remains the same as the size of the messages vary. In our implementation, we keep all available metadata identifiers in memory as well as the modest size metadata values. Thus, if the metadata value is above a certain value, which could be specified in the configuration files, then the system makes a database access to store/retrieve corresponding metadata. This way, the system is able to keep high number of modest size messages in memory. Figure 6 also indicates the system performance behavior when the size of value requires a database access for the context payload sizes ranging from 10 Kbytes to 100 Kbytes. We observe an increase of ~7 ms when the publication operation requires a database access.

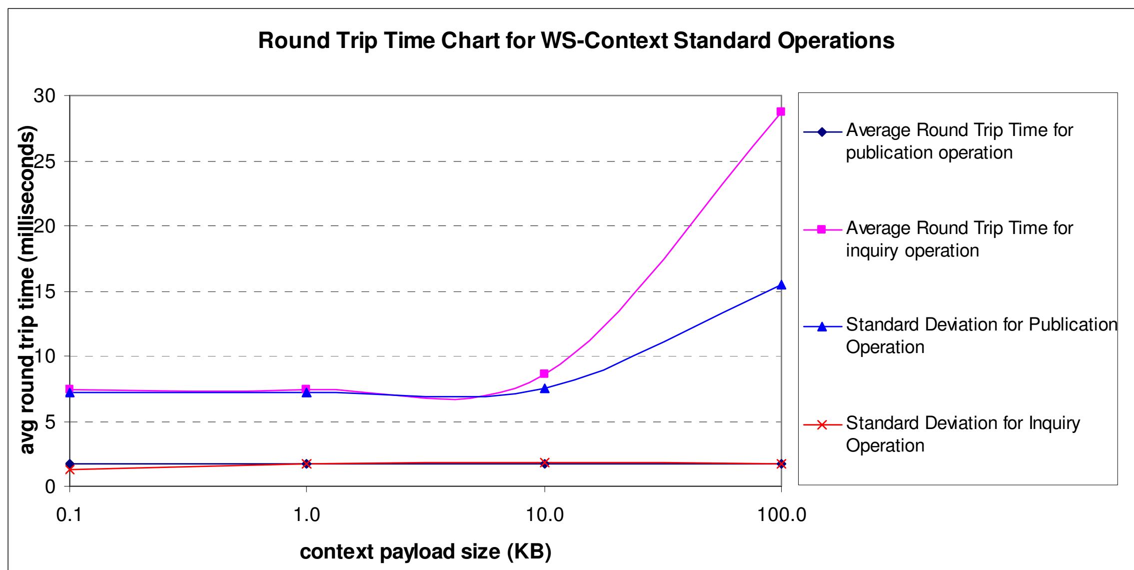

Figure 5 Logarithmic scale round trip time chart for inquiry and publication requests when context payload size increases

	WS-Context inquiry operation		WS-Context publication operation	
Kbytes	mean	stdev	mean	stdev
0.1	7.18	1.34	7.38	1.70
1	7.17	1.73	7.43	1.75
10	7.50	1.79	8.58	1.67
100	15.50	1.77	28.76	1.75

Table 8 Statistics of Figure 5 for hybrid Information Service - WS-Context inquiry and publication operations with changing context payload sizes. Time units are in milliseconds.

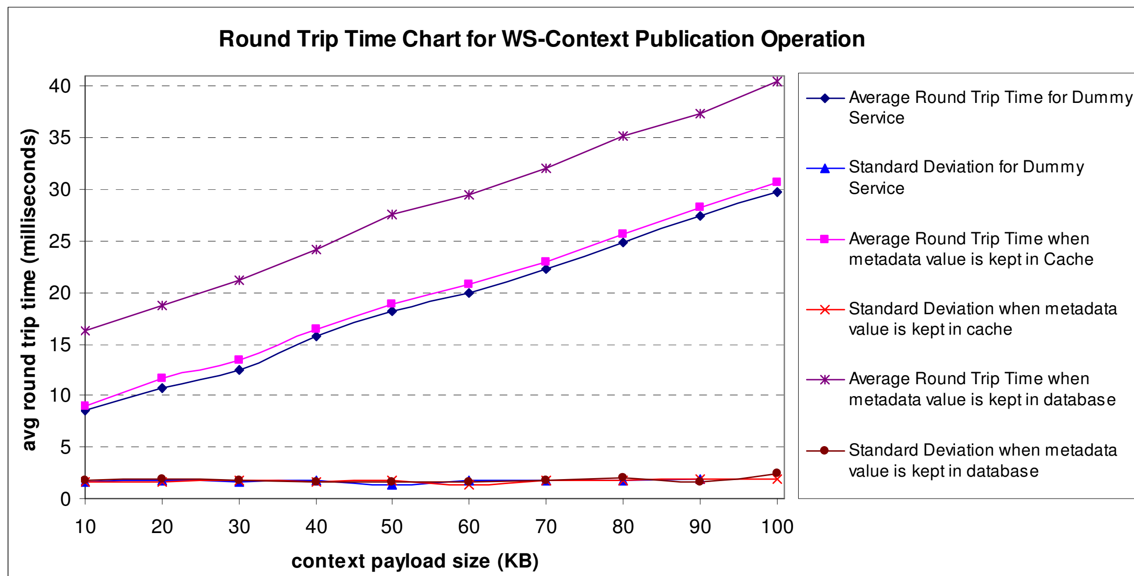

Figure 6 Round Trip Time chart for publication requests when context payload size increases from 10Kbytes to 100Kbtyes

	Dummy service		Hybrid Service with memory access		Hybrid-Service with database access	
Kbytes	mean	stdev	mean	stdev	mean	stdev
10	8.58	1.67	8.93	1.68	16.33	1.79
20	10.78	1.66	11.68	1.67	18.78	1.86
30	12.52	1.72	13.50	1.74	21.23	1.76
40	15.72	1.67	16.42	1.67	24.12	1.62
50	18.17	1.73	18.87	1.75	27.57	1.65
60	19.94	1.41	20.73	1.40	29.43	1.68
70	22.29	1.76	22.98	1.76	31.98	1.72
80	24.85	1.83	25.70	1.83	35.17	2.05
90	27.38	1.83	28.29	1.84	37.37	1.58
100	29.73	1.94	30.64	1.93	40.51	2.42

Table 9 Statistics of Figure 6 for hybrid service - WS-Context publication operations with changing context payload sizes. Time units are in milliseconds

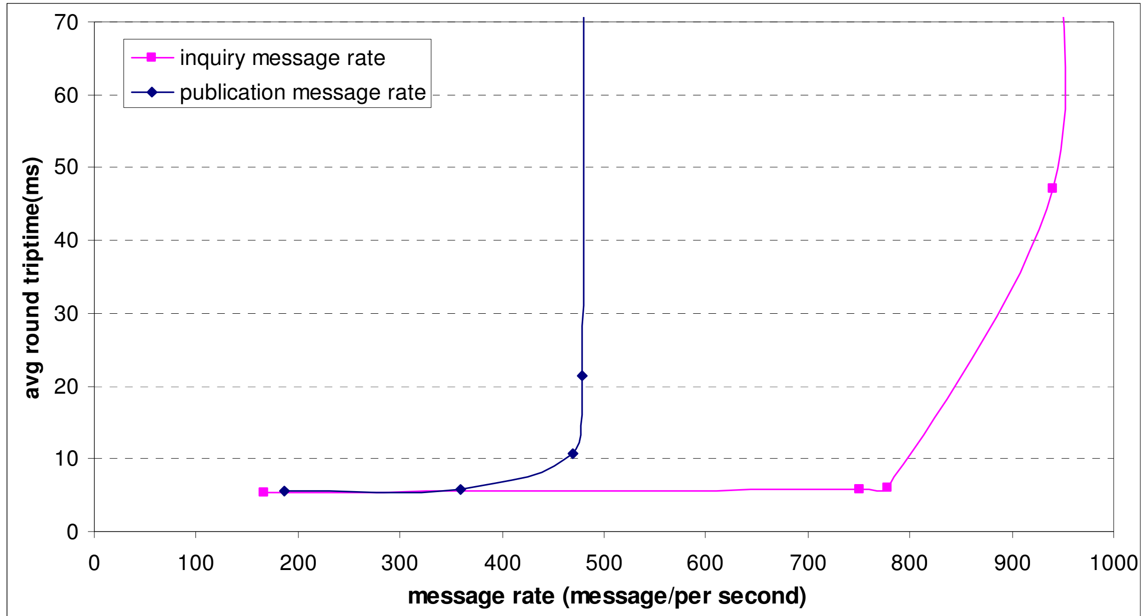

Figure 7 Average hybrid service - WS-Context inquiry and publication response time chart - response time at various levels of message rates per (simultaneous users) per second

WS-Context inquiry operation		
messages/second	mean	stdev
167	5.45	0.65
751	5.84	0.97
778	5.9	0.91
940	47.05	33.52
942	92.25	45.13
WS-Context publication operation		
messages/second	mean	stdev
186	5.65	10.31
359	5.86	39.49
469	10.69	53
479	21.36	203.65
480	70.57	300.56

Table 10 Statistics of the experiment depicted in Figure 7. These measurements were taken with the service when the inquiry and publication request is granted with memory access. Time units are in milliseconds.

Based on the results depicted in Figure 7 and listed in Table 10, we determine that large number of concurrent inquiry requests may well be responded without any error by the system and do not cause significant overhead on the system performance. We observe a threshold between 800 and 940 inquiry messages per second, for the maximum number of concurrent inquiries that can be handled by the system within a second. This threshold is mainly due to the limitations of Web Service container, as we observe the similar threshold when we test the system with an echo service that returns the input parameter passed to it with no message processing is applied.

Based on the results depicted in Figure 7 and listed in Table 10, we also determine that a significant number of concurrent publication requests may well be responded without any error

by the system and do not cause big overhead on the system performance. We observed a threshold, ~ 360 - 480 publication messages per second, for the maximum number of concurrent publication requests that can be handled by the system within a second. This threshold is mainly due to the fault-tolerance level. As the publication message rate is increased, the number of updated/newly written contexts (within a unit time interval) in the ContextSpaces (TupleSpaces) is also increased. In turn, the time required for writing the larger number of updates into MySQL database is increased. Thus, we see higher fluctuations in the response times for increasing number of simultaneous publication requests by examining the standard deviations results listed in Table 10. This experimental study points out the inevitable trade-off between the fault-tolerance and scalability. The lower the fault-tolerance level, the higher the scalability numbers would be for publication operations.

7. Conclusions

We examined the hybrid Grid Information Service as an important tool for 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 hybrid service. Also, we outlined a real-life application use scenario to identify a way and reason of using hybrid information service in Grids. We implemented the system as open-source software which has been used successfully in varying types of Grids: collaboration, earth science and so forth. We discussed the prototype implementation design and presented the results of the proposed approach.

Work remains to further extend the software design in such a way that it can provide support to schemas other than WS-Context and extended UDDI. With this approach, we aim to provide abstraction levels which would separate implementation from specification details.

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 Hybrid Information Service project and Community Grids Laboratory graduate research assistants who have been using the system in their applications*.

8. References

1. Zhuge, H., *China's E-Science Knowledge Grid Environment.* IEEE Intelligent Systems, 19(1), (2004) 13-17, 2004.
2. Bellwood, T., Clement, L., and von Riegen, C., *UDDI Version 3.0.1: UDDI Technical Committee Specification* http://uddi.org/pubs/uddi-v3.0.1-20031014.htm. 2003.
3. Bunting, B., Chapman, M., Hurley, O., Little M,, Mischinkinky, J., Newcomer, E., Webber, J., and Swenson, K. , *Web Services Context (WS-Context) ver 1.0* http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf. 2003.
4. Booth, D., et al. *W3C Web Service Architecture Document, available at* http://www.w3.org/TR/ws-arch/. 2003.

5. *Open GIS Consortium Inc., OWS-1 Registry Service available at http://www.opengeospatial.org/docs/01-024r1.pdf*. 2002.
6. OGC, *The Open Geospatial Consortiom (OGC), web site available at http://www.opengis.org*.
7. *Open Geospatial Consortium Inc., OpenGIS Catalog Specification available at http://portal.opengeospatial.org/files/index.php?artifact_id=901*.
8. *OASIS ebXML Registry Technical Committee, OASIS/ebXML Registry Information Model v2.0 Approved Committee Specification available at http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf*.
9. ebXML, *ebXML Registy Specification, web site available at http://www.ebxml.org*.
10. Zhao, P., Chen, A., Liu, Y., Di, L., Yang, W., Li, P. *Grid metadata catalog service-based OGC web registry service*. in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*. 2004. Washington DC, USA
11. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Manohar, M., Patil, S. and Pearlman. L. . *A Metadata Catalog Service for Data Intensive Applications*. in *SC'03*. November 15-21, 2003. Phoenix, Arizona, USA.
12. Foster, I., Kesselman, C., Nick, Jeffrey M., Tuecke, S. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. in *Open Grid Service Infrastructure WG, Global Grid Forum*. 2002.
13. *Web Service Interoperability (WS-I) Organization, web site is available at http://www.ws-i.org*.
14. *Open GIS Consortium Inc., OWS1.2 UDDI Experiment. OpenGIS Interoperability Program Report OGC 03-028 available at http://www.opengeospatial.org/docs/03-028.pdf*. 2003.
15. Sycline, *Sycline Inc., web site available at http://www.synclineinc.com*.
16. Galdos, *Galdos Inc., web site available at http://www.galdosinc.com*.
17. Dialani, V., *UDDI-M Version 1.0 API Specification*. 2002, University of Southampton – UK. 02.: Southampton.
18. ShaikhAli, A., Rana, O., Al-Ali, R., Walker, D. *UDDIe: An Extended Registry for Web Services. Proceedings of the Service Oriented Computing: Models, Architectures and Applications*. in *SAINT-2003 IEEE Computer Society Press.* . 2003. Orlando Florida, USA.
19. Miles, S., Papay, J., Dialani, V., Luck, M., Decker, K., Payne, T., and Moreau, L. *Personalized Grid Service Discovery*. in *Nineteenth Annual UK Performance Engineering Workshop (UKPEW'03)*. 2003. University of Warwick, Coventry, England.
20. Miles, S., Papay, J., Payne, T., Decker, K., Moreau, L. *Towards a Protocol for the Attachment of Semantic Descriptions to Grid Services*. in *In The Second European across Grids Conference*. 2004. Nicosia, Cyprus.
21. 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.* Journal of Information Technology and Management.
22. *Grimoires. A Grid RegIstry with Metadata Oriented Interface, more info. available at http://www.grimoires.org*.
23. *MyGrid. MyGrid is a bioinformatics Grid project led by the University of Manchester. more available at http://www.mygrid.org.uk/*.
24. Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W., *The WS-Resource Framework, available at http://www.globus.org/wsrf/specs/ws-wsrf.pdf*. 2004.

25. Ballinger, K., et al., *The Web Services Metadata Exchange* http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf. 2004.
26. Bunting, B., Chapman, M., Hurley, O., Little M,, Mischinkinky, J., Newcomer, E., Webber, J., and Swenson, K., *Web Service Composite Application Framework (WS-CAF) Ver 1.0.* http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf. July 2003.
27. Bunting, B., Chapman, M., Hurley, O., Little M,, Mischinkinky, J., Newcomer, E., Webber, J., and Swenson, K, *WS-Coordination Framework (WS-CF) ver 1.0* http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CF.pdf. July 2003.
28. Bunting, B., Chapman, M., Hurley, O., Little M,, Mischinkinky, J., Newcomer, E., Webber, J., and Swenson, K. , *WS-Transaction Management (WS-TXM) ver 1.0* http://www.arjuna.com/library/specs/ws_caf_1-0/WS-TXM.pdf. July 2003.
29. Aktas, M., G. Fox, and M. Pierce, *Fault Tolerant High Performance Information Services for Dynamic Collections of Grid and Web Services.* the International Journal of Grid Computing: Theory, Methods and Applications, Future Generation of Computer Systems (FGCS) Special issue from 1st International Conference on SKG2005 Semantics, Knowledge and Grid, Beijing, China, 2005.
30. Aktas, M., G. Fox, and M. Pierce. *Managing Dynamic Metadata as Context*. in *the proceedings of the Istanbul International Computational Science and Engineering Conference (ICCSE2005* http://www.iccse.org/ *)*. 2005.
31. Oh, S., et al. *Optimizing Web Service Messaging Performance Using a Context Store for Static Data*. in *the 5th WSEAS Int.Conf. on TELECOMMUNICATIONS and INFORMATICS (TELE-INFO '06)*. 2006 Istanbul, Turkey.
32. Carriero, N., Gelernter, D., *Linda in Context.* Commun. ACM, 32(4): 444-458, 1989.
33. Coleman, R., Bhardwaj, A., Dellucca, A., Finke, G., Sofia, A., Jutt, M., Batra, S., *MicroSpaces software with version 1.5.2 available at* http://microspaces.sourceforge.net/. 2004.
34. Pallickara, S. and G. Fox. *NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*. in *Lecture Notes in Computer Science*. 2003: Springer-Verlag.