SP2A: a Service-oriented Framework for P2P-based Grids *

M. Amoretti, F. Zanichelli, G. Conte
Distributed Systems Group
Dipartimento di Ingegneria dell'Informazione
Parco Area delle Scienze, 181A
I-43100, Parma, Italy
amoretti, mczane, conte@ce.unipr.it

ABSTRACT

Service-Oriented Architectures (SOAs) are rapidly becoming the key approach for achieving new levels of interoperability and scalability in the development of Grid applications. Within SOA solutions, current approaches for advertising service providers and for allowing prospective clients to discover them are mostly based on centralized registries. Envisioning Virtual Organizations in which all participants are both resource providers and consumers, in a peer-to-peer fashion, seems to be an appealing approach.

In this paper we propose the Service-oriented Peer-to-Peer Architecture (SP2A), a framework enabling peer-to-peer resource sharing in Grid environments. Resources are not directly exposed but can be accessed through Resource Provision Services, whose semantically enriched interfaces are published in the network. The framework has been implemented as a Java API, which currently supports a number of important technologies such as JXTA (peer-to-peer routing), Web Services (service deployment), and OWL-S (semantic description of services).

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; D.2.11 [**Software Architectures**]: Patterns

Keywords

Service-oriented Architecture, Grid, Peer-to-Peer

MGC '05, November 28-December 2, 2005 Grenoble , France

1. INTRODUCTION

Virtual Organizations (VOs) are transitory communities made of individuals and institutions coupled together by advanced communications technologies. Most institutions have typically a limited amount of resources, which are contended by many users. But it is very likely that in these same institutions there are hundreds of workstations that remain idle most of the time. Grid architectures [7, 10] have emerged to minimize this waste of resources, augmenting the processing power available to users, and to address the challenging problem of large-scale VOs requiring resource sharing with demands for high performance.

Unfortunately, most Grid architectures still rely on a rigid client/server approach which has many drawbacks. A few members of the VO are resource providers publishing their services within centralized indexes, while most VO members are resource consumers. Thus, in traditional Grids a considerable wealth, namely the whole amount of user resources (CPU cycles, storage, etc.), remains unused. Moreover, the centralized index approach for resource publishing and discovery is neither efficient nor robust. When the indexing service has to sustain a demanding load, its response time may become too high. Finally, it constitutes a single point of failure for the whole system.

To achieve good load balancing and scalability, all participants should be active contributors to the distributed functionalities of the Grid, without unflexible distinction between service providers and service consumers. Robust resource sharing mechanisms should be provided, avoiding single points of failure. Moreover, support to participant ranking, based on their capabilities and trustiness, should be granted. Last but not least, security mechanisms (key management, authentication, admission control, authorization, privacy) should be provided, not limited to user-to-service interactions but also for service-to-service interactions.

The Peer pattern [2] is a viable solution to this problem. In a Peer-based distributed system, *i.e.* peer-to-peer system, all nodes have the same structure and are not only potential users but also potential resource providers. The nature and availability of resources that each peer provides can be different: storage, CPU cycles, content, real-time sensor data, etc.

We applied the Peer pattern to realize the *Service-oriented Peer-to-Peer Architecture (SP2A)* [5], a lightweight frame-

work for the development of service-oriented peers for efficient and robust Grid environments. SP2A allows to cope with the requirements of applications with a large number of users dynamically connecting to the system, and provides high levels of scalability, decentralization and interoperability. The framework is being implemented as an open-source Java API, which supports important technologies for peer-to-peer message routing, service description and deployment.

The paper is organized as follows. Section 2 introduces SP2A-based Grids, providing an overview of the framework. The Java API under development is presented in Section 3, with particular emphasys on supported technologies. Section 4 discusses related work on service-oriented architectures. Finally, an outline of SP2A open issues concludes the paper.

2. SP2A-BASED GRIDS

The service-oriented architecture (SOA) paradigm has received significant attention within the software design and development industry in recent times, resulting in many conflicting definitions. The *SOA Reference Model* proposed by OASIS [12] is the one we considered in writing the functional and technical specification of the Service-Oriented Peer-to-peer Architecture (SP2A), our lightweight framework enabling service-oriented peer-to-peer based Grids.

> *A service is a set of functionalities provided by one entity for the use of others. It is invoked through a software interface but with no constraints on how the functionality is implemented by the providing entity.*

A service is opaque in that its implementation is hidden from the service consumer except for (1) the *data model* exposed through the published service interface, and (2) any information included as *metadata* to describe aspects of the service which are needed by service consumers to determine whether a given service is appropriate for the consumer's needs. Consistent with the axiom of opacity, a service consumer cannot see anything behind the service interface and does not know if one service is actually consuming and aggregating other services.

2.1 Service-oriented Peers (SOPs)

SP2A is a realization of the Peer pattern [2], which defines the basic modules for building service-oriented peers (SOPs). In this section we briefly illustrate them.

The *Message Handler* allows peers to create a virtual network overlay on top of the existing physical network infrastructure. This module supports different transport protocols allowing message transmissions, potentially traversing firewalls or NATs.

A pool of *Resource Provision Service*s implements resource management mechanisms for local and remote control of Peer's resources (*e.g.* computational power, storage, multimedia objects, sensors). The same resource may be used in several ways, depending on sharing restrictions (*i.e.* when, where, and what can be done). These constraints can dynamically change, based on the nature of permitted access, and on participants to whom access is allowed.

On the other hand, peer's resources provide enquiry mechanisms allowing analysis of their structure, state, and capabilities. The *Resource Monitor (RM)* relies on these facilities to gather and report information about local resources. The Resource Monitor is responsible for providing a list of available resources, maintaining related information, identifying and reporting failures.

The *User Interface* is the entry point for users to interact with the system of peers. The module is responsible for managing user commands, which are opportunely translated in local control operations or messages to be propagated in the network. The User Interface allows to query the Resource Monitor and to configure local resources, to express complex queries for remote resources and to operate on them.

The *Router* defines the rules for addressing, filtering, sending, and receiving messages. Most of the peer operations, such as publication, search and delivery of resources, rely on the Router.

The *Scheduler* is responsible for managing task execution requests, based on information provided by the Resource Monitor. Since many peers can provide the same resources, their Scheduler modules must be able to cooperate for workload distribution.

The *Security Manager* is mainly responsible for protecting shared resources. It relies on mechanisms to safeguard integrity and authenticity of data, to ensure privacy and confidentiality in communications, and to provide means for user authentication and authorization. Moreover, the Security Manager allows peers to establish their identity within a group. In general, the Security Manager should define the capabilities of the peer within a peergroup, *i.e.*, in increasing order of importance (and trustworthiness required): message forwarding, resource discovery, resource publishing and delivery, group monitoring, group management (*e.g.* voting for changing the rank of another group member), subgroup creation.

The *State Manager* provides facilities to check and change the peer state. *E.g.* in a peergroup the peer could be a supernode (with all its capabilities activated), while in another peergroup it could be a leaf node (unable to perform certain actions). Peers' state changes can be user-driven, or dependent on the configuration of the peergroup.

2.2 SOP Networks

Figure 1 shows a common SP2A-based system, emphasizing intra- and inter-SOP module interactions.

The User Interface is aware of one or more Resource Provision Services, either local or remote. Moreover, it must be able to query the Resource Monitor and the State Manager, to collect information about the system and to provide it to the user. Finally, the User Interface uses the Router when it publishes its Resource Provision Services or searches fors remote ones.

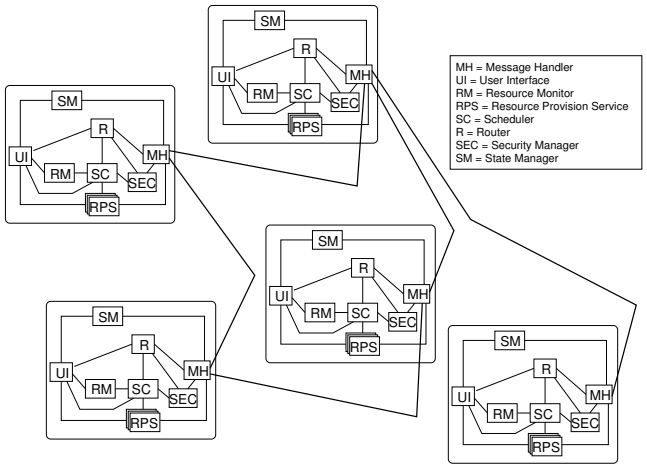

Figure 1: Example of SP2A-based system. For each peer, interactions among internal modules are illustrated.

The State Manager can modify the state of the peer as a consequence of a user command received from the User Interface, or after the delivery of an event by the Message Handler. For this reason the State Manager is referred by both the User Interface and the Message Handler.

The Router computes the destination of messages constructed by the User Interface. Messages can be distributed using the supported transport protocols accessed through one Message Handler.

When a remote request for service provision is received, the Scheduler queries the Resource Monitor for available resources. On positive answer, the Scheduler activates the appropriate Resource Provision Service based on the call handler.

The Security Manager interacts with the Scheduler, Router, and Message Handler, to modify the behaviour of the peer in the current peergroup.

2.3 Service Description

To use a resource, a consumer must know if the related Resource Provision Service (RPS) exists and is available; if it performs a certain function or a set of functions; if it operates under a specified set of assumptions, constraints, and policies; and if it can be invoked through a specified means, including inputs that the service requires and outputs that will form the response to the invocation. The second objective, that capturing service functionalities, is the most difficult to achieve. This aspect needs to be expressed in a way that is generally understandable by service consumers, but able to accomodate a vocabulary that is sufficiently expressive for the domain for which the service provides its functionalities. This may include, among other possibilities, a textual description intended for human consumption, or identifiers/keywords referenced to specific machine-processable definitions.

All SP2A RPSs have a *name*, a short textual *description*, and an uniquely identified *owner*. In addition, they expose an interface which specifies how to access its functionalities. This information is synctactically represented in one or more standard, referenceable formats.

> *The service interface prescribes what information needs to be provided to the service in order to exercise its functionalities and/or the results of the service invocation to be returned to the service requester [12].*

The logical expression of the set of information items associated with the consumption of the service is referred as the service's data model.

The semantics of a service are the shared expectations associated with the service. A service consumer perceives the semantics as broken in three main parts: the data model, the process model, and the behaviour. The latter is the intended real world effect of using a service.

2.4 Service Sharing and Discovery

Locating services dispersed across the system, in response to a discovery query, is a central topic for Grid Computing. A discovery system should be scalable, efficient, autonomic, and fault tolerant.

Central-server based approaches have received large attention for their effectiveness since they guarantee discovery of registered services. Nevertheless, they exhibit many drawbacks: they are neither scalable nor efficient, being performance bottlenecks for the system, and they are not fault tolerant, having a single point of failure.

Peer-to-peer approaches seem the more suitable solutions for Grid Service infrastructures, which are often already organized as federation of entities without any distinct centralized service. At discovery time, each peer is also charged with the duty of propagating and responding to the queries it receives in order to cooperate to provide a distributed search service without any centralized server. While coping with traditional problems of centralized systems listed before, P2P architectures do not guarantee that a request will find a suitable service even if available in the system as there is not guarantee that a query will be broadcast to the whole set of peers. As our analysis is mainly targeted to dynamic communities, P2P still remains the most suitable solution for our architecture. SP2A supports both pure and hybrid peer-to-peer networks. In the first case, all SOPs are equally responsible for routing messages. In the second case, only supernodes, *i.e.* SOPs with higher bandwidth and process capacity, have such duty.

The User Interface allows to perform both simple attribute/value search, and semantic search. The latter is driven by the ontology obtained from the service description provided by the user, *i.e.* text description and Input/Output/Precondition/Effect (IOPE) description if known. Discovered RPSs are ranked by a Semantic Matcher, based on their measured similarity with the reference ontology. w

2.5 Resource Provision

The most interesting scenario is resource delivery, involving the shared Resource Provision Services and five SOP modules (figure 2).

- The Message Handler delivers a message to the Security Manager, communicating resource request and credentials of the requester (another Peer).
- If the requester's credentials are valid, the Security Manager delivers the resource request to the Scheduler.
- The Scheduler interacts with the Resource Monitor to check the availability of the requested resource.
- If the requested resource is available, the Scheduler invokes its allocation on the corresponding Resource Provision Service.
- If the resource is not available, or by its nature can be provided concurrently by many Peers, the Scheduler starts searching for remote resources, transparently to the user. The Router computes the destination(s) and sends request message(s).
- In the meantime, the Resource Provision Service provides the available local resource to the requester, through the Message Handler.

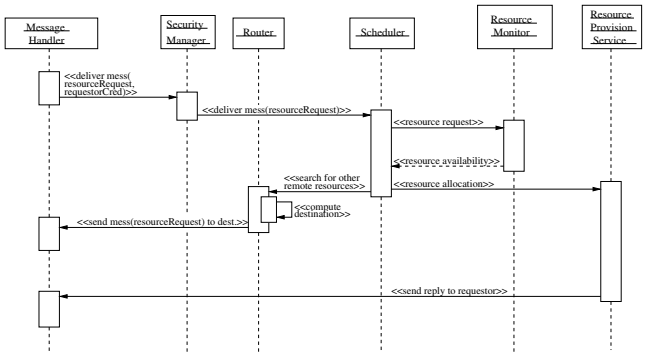

Figure 2: How a SOP provides its resources.

3. SP2A JAVA API

We are implementing the framework as a lightweight Java API, including four branches: group, rps, security and state. In the following we summarize the supported technologies, and we illustrate how SP2A implementats three of the previously described modules.

3.1 Supported Technologies

SP2A currently supports three state-of-the-art technologies: Web Services, OWL-S and JXTA. These technologies complement each others: WSs provide a framework for service design; OWL-S supplies service providers with a core set of markup language constructs for describing the properties and capabilities of their services in unambiguous, computer-intepretable form; JXTA operates at the lower level providing P2P functionalities.

3.1.1 Web Services

The main goal of Web Services is to allow machine-to-machine interaction while traditional Web applications are instead human-to-human oriented. The Open Grid Service Infrastructure (OGSI) [17] specification introduced Web Services (WSs) in Grid systems. The refactoring of OGSI was proposed in January 2004 as WS-Resourse Framework (WSRF) [18], aiming at exploiting new Web Services standards. WSRF retains essentially all the functional capabilities present in OGSI, while changing some of the syntax and also partitioning OGSI functionality into five distinct, composable specifications. Major benefits of this new specification are the explicit distinction between the service and the stateful entities (resources) acted upon by the service.

3.1.2 OWL-S

The OWL-based Web Service Ontology (OWL-S) [11] supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form. OWL-S markup of Web services will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring.

SP2A provides a Semantic Matcher implemented with the OWL-S API [14] (version 1.1-beta) for programmatic comparison of OWL-S service descriptions.

3.1.3 JXTA

Project JXTA [16], a Sun Microsystems open initiative, is the most supported (and advanced) peer-to-peer middleware currently available. The main effort of JXTA developers is to standardize a common set of protocols which define the minimum required network semantics, allowing peers to form and join an unstructured peer-to-peer network based on supernodes. The Java API includes endpoint and pipe modules, implementing a Peer's Message Handler, and rendezvous and resolver modules, implementing a Router.

3.2 The User Interface

The User Interface module is realized by two interfaces which define the basic functionalities required to the SOP, which are group management and RPS deployment, sharing and discovery.

```
public interface RPSManager {
  public void addToRPSList(ResourceProvisionService rps);
  public RPSList getRPSList();
  public void shareRPS(Group group,
                       ResourceProvisionService rps);
  public void findRPS(Group group, ...);
  public DiscoveredRPSVector getDiscoveredRPSVector();
  public void joinRPS(ResourceProvisionService rps);
  public RPSList getJoinedRemoteRPSList();
  public void leaveRPS(ResourceProvisionService rps);
  public void startServiceDiscoveryListener(Group group);
  public void stopServiceDiscoveryListener(Group group);
}

public interface GroupManager {
  public DiscoveredGroupVector getDiscoveredGroupVector();
```

```
  public void shareSubGroup(Group subGroup);
  public void findSubGroup(String attribute, String value);
  public Group createSubGroup(.., SecurityManager secMan);
  public void joinSubGroup(Group subGroup);
  public void leaveSubGroup(Group subGroup);
  public GroupList getJoinedGroupList();
  public void startGroupDiscoveryListener(Group group);
  public void stopGroupDiscoveryListener(Group group);
}
```

SP2A Java API provides JXTA-based classes, and will provide other implementations based on other peer-to-peer middlewares such as JXME [15] and DKS [13].

3.3 The State Manager

The StateManager interface defines some important internal functionalities of the peer, such as supernode behavior activation/deactivation, and asynchronous discovery listeners management.

```
public interface StateManager {
    public Group startPeer();
    public String getPeerID();
    public void printPeerInfo(Group group);
    public void startSuperNodeFunction(Group group);
    public void stopSuperNodeFunction(Group group);
    public boolean isSuperNode(Group group);
    public void checkConnectedPeers(Group group);
}
```

SP2A Java API provides a JXTA-based StateManagerImpl class.

3.4 The Security Manager

It is quite evident that providing security to peergroups is rather a difficult goal, because interactions are not just user-to-service, but also service-to-service on behalf of the peers, thus requiring delegation of rights from peers to services and dynamical instantiation of services. SP2A provides a SecurityManager interface for security policy enforcement in peergroups. An implementation example is the SCSecurity-Manager class, which enables role-playing based on secure credentials. Peers can have different ranks, corresponding to the actions they are allowed to perform within the group. Only peers with the highest rank can be members of the Group Authority and provide *Group Membership Certificates (GMCs)*, which state membership and define the capabilities of each peer on behalf of the group. We developed an enhanced version of JXTA routing services, to enable authorization control in the supernode network.

3.5 RPS Deployment and JXTA-SOAP Based Messaging

We assume that resources are shared as a Resource Provision Services (RPSs). The ResourceProvisionService inteface provides the following methods: setInterfaceDescription(), getInterfaceDescription(), and invokeOperation(). Furthermore, developers should realize RPS specific functionalities in the related implementation classes. As an example, consider the following:

```
public class MathService
    extends ResourceProvisionServiceImpl {
  public void add(Integer a) { ... }
  public void subtract(Integer b) { ... }
  public int getValueRP() { ... }
}
```

Service deployment is transparent to the user, which only has to invoke the shareRPS() method of the RPSManager. This encapsulates the creation of a JXTA service advertisement, and the creation of a service instance in a new thread.

```
//// [1] Service construction
MathService math = new MathService();
math.setName("MathService");
math.setVersion("1.0");
math.setOwner(accSvc.getPeerID());
math.setDescription("A very useful math service.");
math.setClassName("MathService");
math.setGroupID(mainGroup.getID());
math.setGroupName(mainGroup.getName());
math.setGroupDescription(mainGroup.getDescription());
math.setSecurity(false);
math.setInterfaceDescription("WSDL", WSDLFile);
rpsManager.addToRPSList(math);

//// [2] Service activation and publication
rpsManager.shareRPS(math);
```

Messaging is based on JXTA-SOAP [6], a JXTA-related project whose ownership we have recently obtained. Current JXTA-SOAP component release is based on Axis JAX-RPC, an API for the invocation of Web Services across heterogeneous platforms. A JAX-RPC client can use stubs-based, dynamic proxy or dynamic invocation interface (DII) programming models to invoke a heterogeneous Web Service endpoint. JAX-RPC requires SOAP over HTTP for interoperability. JAX-RPC uses SAAJ API which provides a standard Java API for constructing and manipulating SOAP messages with attachments. Moreover, JAX-RPC provides support for SOAP message processing model through the SOAP message handler functionality. This enables developers to build SOAP specific extensions to support security, logging and any other facilities. In particular, JXTA-SOAP uses JXTA bidirectional pipes to carry out SOAP messages between active service instances and remote client peers.

Using the math example, we illustrate how a SP2A-based peer performs attribute/value search and service interaction, using the RPSManager and the methods which all RPSs have in common.

```
rpsManager.findRPS(mainGroup, "Name", "MathService");

... // search results filtering

ResourceProvisionServiceImpl math =
(ResourceProvisionServiceImpl)
rpsManager.getDiscoveredRPSVector().getElementAt(..);

String WSDLbuffer =
```

```
math.getInterfaceDescription("WSDL");

... // interface parsing

Integer a = new Integer(..);
math.invokeOperation("add", new Object[] { a });
```

We are working to enhance JXTA-SOAP to allow WSRF deployment directly in the peer's JVM. Currently, WSRF services are deployed in containers based on Globus Toolkit 4 [1] and bridging with SOP's plain WSs is achieved at the expense of service response time, although the overhead is quite negligible if the SOP and its containers run on the same machine.

4. RELATED WORK

The convergence of Grid and Peer-to-Peer environments is clearly depicted in [9], envisioning systems with a larger number of stable nodes than in today's P2P networks and resources with highly diverse types and characteristics. A more recent Global Grid Forum paper [4] outlines the importance of peer-to-peer technologies to enable larger-scale, higher-performance grid systems, and attempts to develop a set of requirements for the Open Grid Service Architecture (OGSA) [7] to be used to build peer-to-peer applications.

SP2A current prototype can be compared to WSPeer [8], a framework for deploying and invoking Web Services in a peer-to-peer environment, although SP2A is designed to be independent from a particular service description and implementation technology. Moreover, SP2A is concerned with stateful services for Grid environments, *i.e.* services for resource management (deployment, sharing, reservation and use). Another interesting system is OurGrid [3], which is released as a set of components enabling peer-to-peer sharing of computational power. The main difference between OurGrid and SP2A is that the former associates each peer to an istitution, while the latter can run on single user machines. Moreover, SP2A is service-oriented, which allows to address several security issues using standard mechanisms.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described the current status of the SP2A project, a framework enabling service-oriented Grids based on the peer-to-peer paradigm. We focused on the current Java API, providing an example of service deployment, sharing and use.

Next step in the development of the SP2A framework is the creation of a benchmark for testing and improving the ontology-driven service discovery algorithm. Enhancements to the JXTA-SOAP infrastructure will be introduced, in particular for WSRF support. We are studying other peer-to-peer technologies, such as JXME for mobile ad-hoc networks, which will be supported by the framework.

6. ACKNOWLEDGMENTS

This work has been supported by the "WEB-MINDS" FIRB project of the National Research Ministry.

7. REFERENCES

[1] The Globus Alliance. http://www.globus.org.

[2] M. Amoretti, M. Reggiani, F. Zanichelli, and G. Conte. Peer: an Architectural Pattern Enabling Resource Sharing in Virtual Organizations. In *The 12th Pattern Languages of Programs (PLoP) 2005, Monticello, Illinois, USA*, September 2005.

[3] N. Andrade, L. Costa, G. Germóglio, and W. Cirne. Peer-to-peer grid computing with the OurGrid Community. In *Proceedings of the 23rd Brazilian Symposium on Computer Networks*, 2005.

[4] K. Bhatia. Peer-To-Peer Requirements On The Open Grid Services Architecture Framework. Technical report, Global Grid Forum (GGF), July 2005.

[5] Distributed System Group. Service-oriented Peer-to-Peer Architecture (SP2A) homepage. http://dsg.ce.unipr.it/research/SP2A.

[6] Distributed Systems Group. JXTA-SOAP project. http://soap.jxta.org/.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), March 2001.

[8] A. Harrison and I. Taylor. Dynamic Web Service Deployment Using WSPeer. In *The 13th Mardi Gras Conference, Baton Rouge, Louisiana.*, February 2005.

[9] A. Iamnitchi and I. Foster. A Peer-to-Peer Approach to Resource Location in Grid Environments. In *Grid Resource Management*. Kluwer Publishing, 2003.

[10] Z. Nemeth and V. Sunderam. Characterizing Grids: Attributes, Definitions, and Formalisms. *Journal of Grid Computing*, 1(1), Oct. 2003.

[11] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, 2002.

[12] S. O. A. Reference Model TC. Service Oriented Architecture Reference Model. Technical report, OASIS, June 2005.

[13] Royal Institute of Technology and Swedish Institute of Computer Science. Distributed K-ary System. http://dks.sics.se/.

[14] E. Sirin. OWL-S API. http://www.mindswap.org/2004/owl-s/api/.

[15] Sun MicroSystems. JXTA for J2ME. http://jxme.jxta.org.

[16] B. Traversat, M. Abdelaziz, and E. Pouyoul. A Loosely-Consistent DHT Rendezvous Walker. *Project JXTA*, March 2003.

[17] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. *Open Grid Services Infrastructure (OGSI) Version 1.0*, June 2003.

[18] Web Services Resource Framework (WSRF) TC. The WS-Resource Framework. Technical report, OASIS, March 2004.