ENABLING PEER-TO-PEER WEB SERVICE ARCHITECTURES WITH JXTA-SOAP

Michele Amoretti, Francesco Zanichelli, Gianni Conte
Information Technology Department, University of Parma
Via Usberti 181/a, 43100 Parma, Italy

Matteo Bisi
Sinfo Pragma Spa
Via Benedetta 77/a, 43100 Parma, Italy

ABSTRACT

Service oriented architectures (SOA) are nowadays the de-facto standard for IT infrastructure integration in the business and commerce domain. To better cope with fundamental issues such as scalability, flexibility and fault-tolerance, large-scale SOAs envision arbitrary pairs of peer application entities communicating and providing services directly with each other in a peer-to-peer to fashion.
In this paper we describe JXTA-SOAP, an important extension to the well-know peer-to-peer JXTA technology to the purpose of peer-to-peer sharing of Web Services, *i.e.* deploying, publishing, discovering and invoking Web Services in a network of JXTA peers. The JXTA-SOAP API has been exploited to build peer-to-peer SOAs in the e-logistics domain.

KEYWORDS

Service-Oriented Architectures, Peer-to-Peer, Virtual Communities, Ubiquitous Computing

1. INTRODUCTION

Service-oriented computing [Bichler06] defines an architectural style whose goal is to achieve loose coupling among interacting software entities. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software entities on behalf of their owners. Service-Oriented Architectures (SOAs) [He03] achieve loose coupling among interacting software entities by employing two architectural constraints: a small set of simple and ubiquitous *interfaces*, and descriptive *messages* constrained by an extensible schema delivered through the interfaces. No, or only minimal, system behaviour is prescribed by messages, and a schema limits the vocabulary and structure of messages. Services are passive, *i.e.* they wait for incoming messages before doing any work. Services expose one or more endpoints where message can be sent. Each endpoint consists of an address which specifies where to send messages, a binding which describes how to send messages, and a contract which describes what the message should contain. Consumers need to know this information before they can access a service.

In this context, Web Service technologies [Nezad06] provide standard, simple and lightweight mechanisms for exchanging structured and typed information between services in a decentralized and distributed environment. By focusing solely on messages, the Web Service model is completely language, platform, and object model-agnostic. As long as the contract that explains service capabilities and message sequences and protocols it expects is honoured, the implementations of Web Services and service consumers can vary independently without affecting the application at the other end of the conversation.

Traditional SOAs based on the client/server paradigm are characterized by a server application, hosted by an always-on end system, which provides services to many other client applications, which can be hosted by sometimes-on end systems. The server has a fixed, well-known IP address. Often in a client/server application, a single server host is incapable of keeping up with all the requests form its clients. For this reason, clusters of hosts sometimes referred as server farm are often used to create a powerful virtual server

in client/server architectures. In this context, providers typically publish service interfaces on index services which provide white/yellow pages functionalities.

A novel approach to large-scale SOAs envisions arbitrary pairs of peer application entities communicating and providing services directly with each other. In a *peer-to-peer SOA*, none of the participant hosts is required to be always on; moreover, a participating host may change its IP address each time it comes on. Churn rate is a measure of the number of peers moving into or out of the system over a specific period of time. If shared resources and services are evenly distributed among all participants, an high churn rate does not affect the performance and the consistency of the system. On the other hand, because of the highly distributed and decentralized nature of peer-to-peer SOAs, they can be difficult to manage. A service can be offered or withdrawn by a peer at any time, and consumers have to discover whether and where a service is provided.

In the context of JXTA project [JXTA03], the design and implementation of the JXTA-SOAP API is being carried out with the purpose to enable peer-to-peer sharing of Web Services. JXTA's peer-to-peer underpinnings provide efficient scalable alternatives for Web Service discovery and communication. Currently, JXTA-SOAP is a provided as a Java API which allows to deploy, publish, discover and invoke Web Services in a network of JXTA peers. It is important to say both JXTA and Web Service technologies are XML-based, and programming language independent, thus JXTA-SOAP could be written *e.g.* in C++, rather than Java.

Section 2 provides the description of the JXTA-SOAP API, with particular emphasis on service deployment, lookup and invocation. Security mechanisms are also illustrated. Section 3 illustrates an application example in the field of e-logistics. Section 4 provides an overview of related works in service-oriented architectures based on the peer-to-peer paradigm. Section 5 concludes the paper, illustrating future plans for the improvement of JXTA-SOAP.

2. INSIDE JXTA-SOAP

JXTA [JXTA03] is Sun MicroSystem's set of open, generalized peer-to-peer protocols that allow a vast class of networked devices (smart phones, PDAs, PCs and servers) to communicate and collaborate seamlessly in a highly decentralized fashion. JXTA aims at reducing the complexity otherwise required to build and deploy peer-to-peer applications and services by providing an open source software platform and deployed virtual network. The JXTA platform provides core building blocks (IDs, advertisements, peergroups, pipes) and a default set of core policies, which can be replaced if necessary. Its Java implementation is currently the most advanced (and supported) toolkit in the peer-to-peer middleware panorama.

The open source community that has formed around JXTA technology is very diverse, including members from ISVs, corporate IT, academia, as well as many independent developers and consultants. Our research group leads the JXTA-SOAP project, whose design goals are wrapping Web Services into JXTA services, and using JXTA for locating Web Services and transporting SOAP messages.

In the following we describe the JXTA-SOAP API, and the internal mechanisms of the package. During the design process many *remoting patterns* [Volter05] have been taken into account, in particular those which are not directly implemented in JXTA and Axis, which are the basic building blocks of JXTA-SOAP.

2.1 Service Deployment

In order to deploy its services, a peer has to instantiate and configure the related `SOAPService` objects (one for each hosted service), and to advertise the service interfaces in the network. The class diagram in figure 1 illustrates the classes and the relationships among them which are involved in this tasks. The `Peer` class represents the generic peer application implemented by the developer, and relies on JXTA-SOAP's API which provides all the other classes represented in the diagram. In particular, `SOAPService` is the class which must be associated to every service implementation, providing a common initialization method. Each service implementation contains a `ServiceDescriptor`, providing basic information which uniquely identifies the service, and implements the `ServiceLifecycle` interface, which is used to pass a

`Context Object` [Krishna05] (constructed by the user application) to the Web Service class. Finally, a security `Policy` is associated to each service implementation.

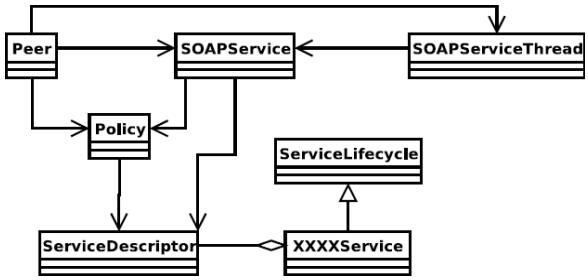

Figure 1 - Class diagram for service deployment.

At start-up, the typical `Peer` object bootstraps the JXTA platform, configuring basic connectivity settings, such as TCP/IP and HTTP ports, and joins the public JXTA peergroup. To bridge to existing common membership and access technologies, JXTA introduces the concept of `MembershipService`, which allows the peer to establish an identity within a peergroup. Each `MembershipService` implementation is responsible for its own protocol definition. JXTA's implementation of Public Key Infrastructure (PKI) is named *Personal Security Environment (PSE)*, and since JXTA v2.3 the default choice for the main peergroup is the `PSEMembershipService`. This service also initializes the *PSE KeyStore*, an object that acts as a secure datastore for peer certificates and keys. Once the `Peer` has authenticated itself to PSE, it is ready to deploy its service instances. These tasks are illustrated by the sequence diagram in figure 2, and detailed in the following of this section.

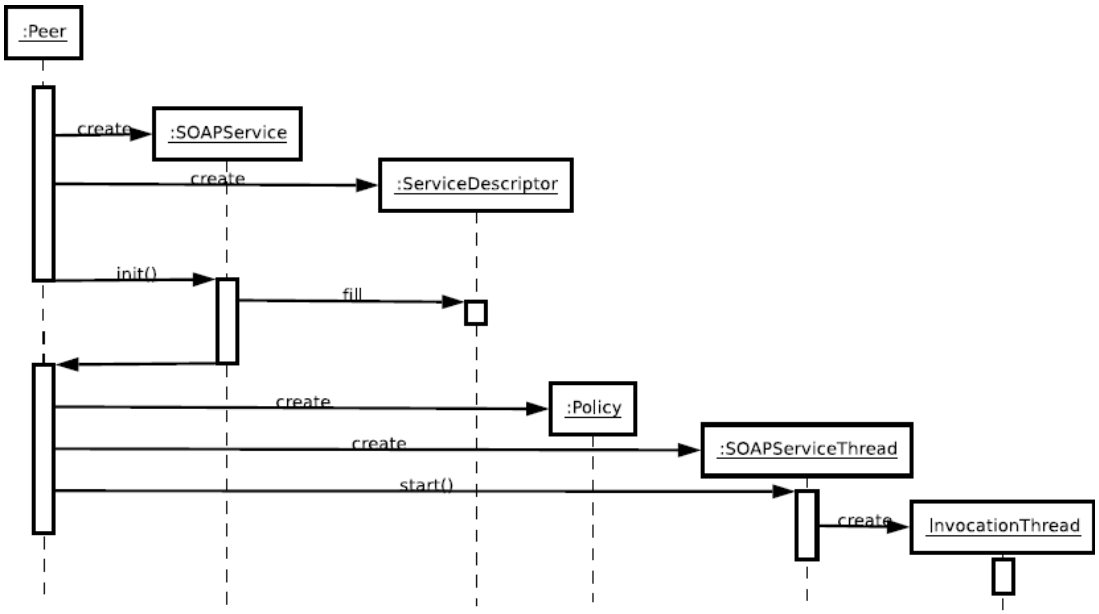

Figure 2 - Sequence diagram for service deployment.

For each service to be deployed, a couple of `SOAPService` and `ServiceDescriptor` is instantiated. The `ServiceDescriptor` is filled with service-specific information, such as the service name, a brief description, the implementation class name, the peergroup ID, the security tag. Then, the "Parm" section of the service advertisement is filled with the service WSDL. Next step is to create the context object, whose parameters are stored in a `HashMap`, and to pass it to the `SOAPService` instance.

The `SOAPService`'s initialization method is then called, to set the `ServiceDescriptor`, bootstrap Axis engine, create and publish the public pipe and the advertisement of the service. The

ServiceDescriptor is then filled with some parameters which are needed by the Axis engine (see below, Service Invocation).

For each service implementation, the Peer spawns a SOAPServiceThread which waits for other peers' connections to that service, on the service public pipe, and creates a pool of InvocationThreads to efficiently execute service invocation (the Threadpool pattern is illustrated in [Pyarali01]).

The Axis engine, in which the SOAPService is deployed, implements the JAX-RPC API, one of the standard ways to program Java services. According to JAX-RPC specification, if the service implements the ServiceLifecycle interface, the JAX-RPC runtime system is required to manage the lifecycle of the corresponding service endpoint instances. The lifecycle of a service endpoint instance is realized through the implementation of the init and destroy methods of the ServiceLifecycle interface. After loaded and instantiated, the JAX-RPC runtime system is required to initialize the service instance before any requests can be serviced. The init method is invoked, and the service instance uses this method to initialize its configuration and setup access to any external resources. The context parameter in the init method enables the service instance to access the context provided by the underlying JXTA-SOAP based runtime system. The init method implementation should typecast the context parameter to an appropriate Java type. For services deployed in a JXTA-SOAP based runtime system, the Java type of the context parameter is defined by the developer that is using the JXTA-SOAP API, and passed to the SOAPService through the SOAPService.setContext method.

Once a service instance has been initialized, the Axis engine may dispatch multiple remote method invocations to it. After that, when the Axis engine determines that the service instance needs to be removed from service of handling remote invocations, it invokes the destroy method. In the implementation of the destroy method, the service class performs cleanup and releases its resources.

2.2 Service Publication and Lookup

The main enhancement of JXTA-SOAP with respect to traditional Web Service frameworks is the adopted distributed approach for service advertising and lookup. JXTA-SOAP allows to encapsulate WSDL interfaces in particular JXTA documents, the so-called ModuleSpecAdvertisements, which can be spread into the network using many different routing policies which can be inserted in JXTA protocol stack. In general, a ModuleSpecAdvertisement describes a module specification. Its main purpose is to provide references to the documentation needed in order to create conforming implementations of that specification. A secondary use is to make running instances remotely usable, *e.g.* by publishing a pipe advertisement associated to the ModuleSpecAdvertisement. Once the ModuleSpec-Advertisement has been filled with the service WSDL and the secure pipe tag, a context object is created and passed to the SOAPService instance. Moreover, the Peer spawns a SOAPServiceThread which waits for other peers' connections to the deployed service, on the public pipe associated to the ModuleSpecAdvertisement.

Service publication is a distributed process, which uses network nodes as a distributed repository (on the contrary, traditional UDDI registries are centralized interface description repositories). As for publication, service lookup is a distributed process, which can be conceptualized as message exchange between low-level JXTA modules.

JXTA's default message routing protocol for advertisement sharing and discovery is called SRDI. Its description is out of the scope of this paper, thus we suggest to read the paper of Traversat *et al.* [Traversat03] which provides a deep analysis.

2.3 Service Invocation

SOAP is the most widely used Web Service protocol for message enveloping. SOAP message management in JXTA-SOAP is based on Axis, which is based on two remoting patterns: the *Requestor*, and the *Invoker* [Volter05]. For remote service invocation, consumer peers only interact with JXTA-SOAP's extension of Axis' Call class. The latter implements the Requestor pattern, which constructs a message from the absolute reference of the remote service, the operation name, its arguments and return type. Axis' default

`Call` uses URLs as absolute service references, while JXTA-SOAP's `Call` extension uses the `ServiceDescriptor` and the public pipe advertisement of the service. To create instances of the `Call` class, JXTA-SOAP (as Axis) provides a Factory Method [Gamma95] `createCall()`.

We previously described the tasks which are performed when a Web Service is deployed by a peer, and we mentioned that some parameters are put in the `ServiceDescriptor` for further use by the Axis engine. In particular, one of these parameters is the Web Service Deployment Descriptor (WSDD). When the WSDD is sent to the Axis engine running in the peer, the Invoker is informed that it supports the new Web Service. Thus, when an invocation reaches the peer, the Invoker looks up the class which implements the service, and lets the instance handle the request message. In details, the Invoker reads incoming messages and demarshals the parameters inserted by the consumer peer's Requestor (absolute reference of the service, operation name, arguments, return value) and dispatches the message to the targeted service.

Figure 4 illustrates Associations among classes which are involved in the service invocation task, performed by a generic `Peer`. The latter, once it has discovered the `ModuleSpecAdvertisement` and the WSDL interface of the required Web Service, can create a `SOAPTransportDeployer`, which manages the transmission of SOAP messages to and from the service using its pipe. Then the `Peer` creates a `ServiceDescriptor`, which is used by the `CallFactory` to instantiate the `Call` object.

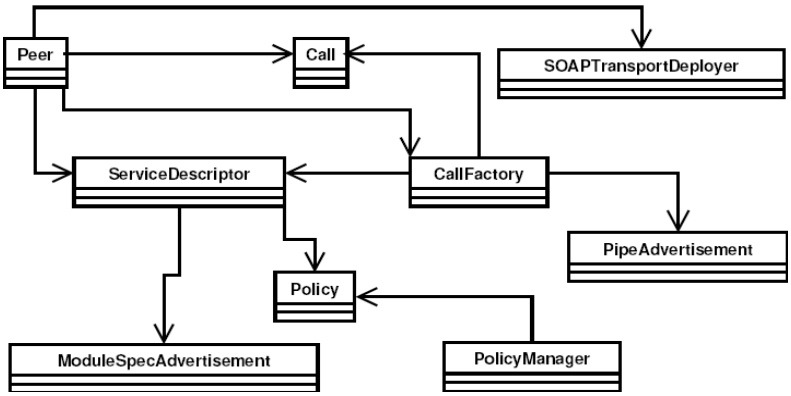

Figure 4 - Class diagram for service invocation.

2.4 Secure Service Invocation

The high degree of decentralization which characterizes the peer-to-peer approach introduces several security issues. Currently, JXTA-SOAP supports secure service invocation by means of two orthogonal mechanisms. The first one, *transport-level security*, allows to create a TLS-based secure channel which guarantees the integrity and confidentiality of exchanged information, by means of mutual authentication between parties (using X509 certificates), and data encoding. The other approach is WSS-based *message-level security*, for which SOAP messages sent by service consumers contain security parameters (tokens) which are extracted by service providers to check for consumers' compliance with the security policy of the invoked service.

Among `net.soap.jxta.security` packages, `policy` provides two important modules, *i.e.* `Policy` and `PolicyManager`. The latter is a class which allows to associate a service with a security policy (currently: TLS, WSS, or both). The `Policy` interface provides methods which are commonly implemented by all policy classes. Its current implementations are `DefaultTLSPolicy`, based on `JXTAUnicastSecure` pipes which subsume default (insecure) `JXTAUnicast` pipes, and `DefaultWSSPolicy`, which uses Apache's WSS4J to provide SOAP messages with security headers and fill them with tokens. Policies are associated to services by means of two extensions of the `<Parm>` field of the `ModuleSpecAdvertisement`. The first extension is the `<security>` tag, whose value (`true` or `false`) indicates whether a service is secured or not. The other extension is the `<InvocationCharter>`, a XML document which is inserted in the `<Parm>` field of the `ModuleSpecAdvertisement` if the

<security> tag is set to true. The secure invocation model is illustrated in figure 6 (TLS-based case), with particular emphasis on multithreaded handling of concurrent invocations.

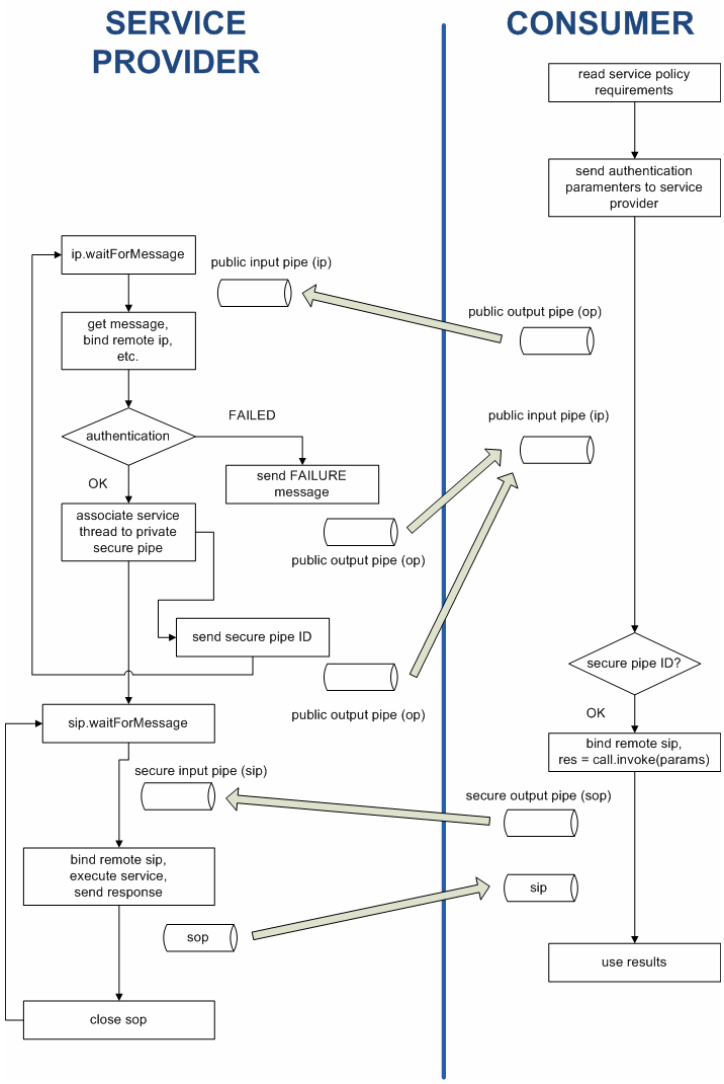

Figure 6 - TLS-based secure invocation model.

3. APPLICATION EXAMPLE: E-LOGISTICS

JXTA-SOAP is being used in a number of applications, ranging from network control [GIGAMANP2P] to collaborative knowledge-sharing systems. In this section we illustrate how JXTA-SOAP has been used for the development of a prototype of the service-oriented architecture proposed within the STIL project, whose purpose is to enable digital logistics in regional inter-firm networks.

Logistics can be defined as the geographical repositioning of raw materials, work in process, and finished inventories where required at the lowest cost possible, through the integration of information, transportation, inventory, warehousing, material handling, and packaging. In STIL, logistics is considered in the context of interactions between providers and consumers of products and services in the e-market place. STIL's final goal is to create a region-wide Virtual Logistic Pole providing Enterprise Application Integration (EAI) for business to business (B2B) applications to manufacturing firms, transportation carriers and logistic hubs. The

concept of value-chain is applied not only to the manufacturer-carrier chain, but also to public interest, with particular emphasis on process observation and optimization for environment and quality of life safeguard.

Figure 7 illustrates a typical STIL scenario, with two kinds of actors, namely manufacturers' logistic operators and carriers, and two types of services which are critical for the value-chain:

- the `TransportationBroker` which accepts transport orders from manufacturers' logistic operators, requests and retrieves mission plans, interacts with carriers to obtain their quotations and finally orders the best one;
- the `MissionPlanner` which is invoked by the `TransportationBroker` to compute optimized mission plans considering different transportation systems.

In this scenario, each kind of service may be offered by multiple competing providers with different costs and implementations. In the prototype, emphasizing decentralized and direct e-business interaction, JXTA-SOAP provides to each node the basic layer for peer-to-peer service sharing and interaction.

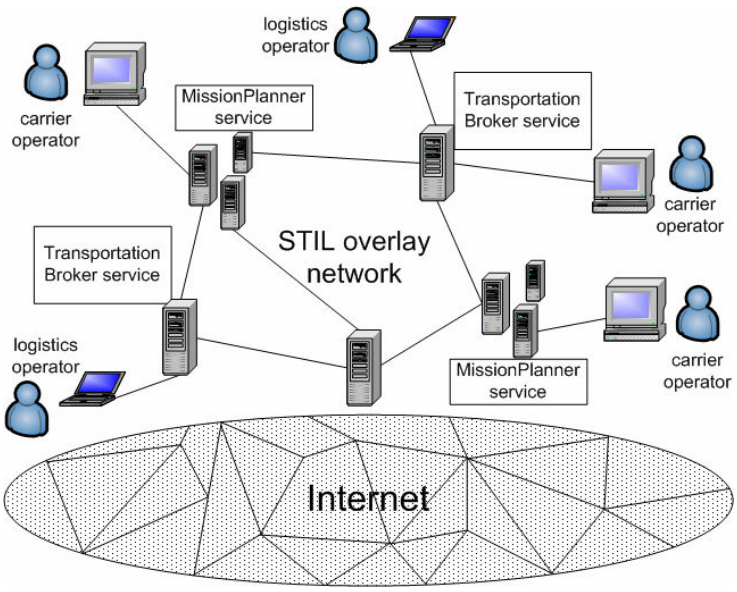

Figure 7 - Example STIL scenario: aggregation of demand for transportation services.

4. RELATED WORK

JXTA-SOAP current distribution can be compared to WSPeer [Harrison05], a framework for deploying and invoking Web Services in a peer-to-peer environment. Recently, WSPeer has been enhanced with Globus Toolkit core libraries to support the WS Resource Framework (WSRF), which defines a generic and open framework for modeling and accessing stateful resources using Web Services. To the same purpose, JXTA-SOAP will adopt the Apache Muse library.

WSPDS [Kashani04] is a distributed discovery service implemented as a cooperative service. Networked WSPDS servents collaborate to resolve discovery queries raised by their peers. Each servent is composed of two engines, communication engine and local query engine, standing for the two roles that a servent plays: communication and query management. WSPDS provides a primitive layer which implements the Gnutella protocol (*i.e.* message flooding) for keyword-matching queries. WSPDS also integrates a semantic layer, providing a discovery mechanism for which queries are routed to the neighbours that have the most similar services to the target service. The solution appears to be quite inefficient, since semantic matchmaking is performed on top of the keyword-matching mechanism.

A recent Global Grid Forum paper [Bhatia05] outlines the importance of peer-to-peer technologies to enable larger-scale, higher-performance service-oriented Grid systems, and attempts to develop a set of

requirements for the Open Grid Service Architecture (OGSA) [OGSA01] to be used to build peer-to-peer applications based on Web Services.

5. CONCLUSIONS AND FUTURE WORK

The increasing adoption of service-oriented architectures calls for efficient, scalable alternatives to the traditional client/server model for service discovery and interaction. This paper described a result of our activities in the emerging field of middleware for peer-to-peer SOAs, namely the JXTA-SOAP API which offers lightweight albeit efficient mechanisms to deploy Web Services in JXTA networks. JXTA-SOAP has already proved its value for several applications, *e.g.* in the context of decentralized service sharing and interaction for the STIL e-logistics project.

As further work, JXTA-SOAP will be enhanced to include Web Service Resource Framework (WSRF), in order to provide peers the ability to access and manipulate state, *i.e.* data values that persist across, and evolve as a result of, Web service interactions. Moreover, we are working on a J2ME-based version of JXTA-SOAP, which uses kSoap in place of Axis.

ACKNOWLEDEGMENT

This work has been partially supported by the Italian Ministry for University and Research (MIUR) within the project PROFILES under the PRIN 2007 research program.

REFERENCES

Bichler06 - M. Bichler, K.-J. Lin, 2006. Service-Oriented Computing. *In IEEE Computer*, Vol. 39, No.3, pp 49-65.

He03 - H. He, 2003. What is Service-Oriented Architecture. http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

Nezad06 - H. R. Motahari Nezad, B. Benatallah, F. Casati, F. Toumani, 2006. Web Services Interoperability Specifications. *In IEEE Computer*, Vol. 39, No. 5, pp 24-32.

JXTA03 - B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Poyoul, B. Yeager, 2003. Project JXTA 2.0 Super-Peer Virtual Network. *Sun Microsystems Technical Report*.

Traversat03 - B. Traversat, M. Abdelaziz, and E. Pouyoul. A Loosely-Consistent DHT Rendezvous Walker. Project JXTA, March 2003.

Krishna05 - A. Krishna, D. C. Schmidt, M. Stal, 2005. Context Object: A Design Pattern for Efficient Middleware Request Processing. *Proceedings of the 12th Pattern Language of Programming Conference*, Allerton Park, Illinois, USA.

Volter05 - M. Volter, M. Kircher, U. Zdun, 2005. Remoting Patterns, Wiley.

Pyarali01 - I. Pyarali, M. Spivak, R. Cytron, D. C. Schmidt, 2001. Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA. *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, Snowbird, Utah, USA.

Buschmann96 - F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, 1996. Pattern-Oriented Software Architecture. John Wiley & Sons.

Gamma95 - E. Gamma, R. Helm, R. Johnson, J. Vlissides, 1995. Design Patterns. Addison-Wesley, 1995.

GIGAMANP2P - GigaMAN P2P project homepage. http://gigamanp2p.inf.ufrgs.br

Harrison05 - A. Harrison, I. Taylor, 2005. Dynamic Web Service Deployment Using WSPeer. *Proceedings of the 13th Mardi Gras Conference*, Baton Rouge, Louisiana, USA.

Kashani04 - F. Banaei-Kashani, C. Chen, C. Shahabi, 2004. WSPDS Web Services Peer-to-peer Discovery Service. *Proceedings of the International Symposium on Web Services and Applications*, Las Vegas, Nevada, USA.

Bhatia05 - K. Bhatia, 2005. Peer-To-Peer Requirements On The Open Grid Services Architecture Framework. *Global Grid Forum (GGF) Technical Report*.

OGSA01 - I. Foster, C. Kesselman, J. Nick, S. Tuecke, 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *In International Journal of Supercomputer Applications*, Vol.15, No.3.