Web Services in Natural Language:  
Towards an Integration of Web Service and Semantic Web Standards in the JXTA Peer to Peer Environment

P. Contreras, D. Zervas and F. Murtagh, CS, RHUL (2005 May 18)

Summary

We survey the context of separation of concerns in business logic and its implementation in Web Services, and then look at the orchestration and choreography of Web Services. We pursue an approach to Web Service design and implementation in order to support ontologies and through them natural language queries. This implies support of Web Service Description Language (WSDL) and Simple Object Access Protocol (SOAP) in regard to Web Services. For ontology support, we have XML (eXtensible Markup Language) and want to leave the way open for support of Ontology Web Language, a semantic web standard. The open source JXTA platform offers many advantages as a basis for this work, such as search and discovery functionality. In this report, we review other work in this area, and describe our prototype.

1. The Business Problem, Orchestration and Choreography

There is a need to extend current businesses to support the integration of Web Services, both internal and external. However, existing methods for creating business processes are not designed to interact with components that cross organisational boundaries. These methods are also not flexible enough to handle the technical interfaces (e.g., SOAP and WSDL) introduced by Web Services. Processes being built today need the business agility to quickly adapt to customer needs and market conditions. This would include incorporating new customers, partners, or suppliers used in a process.

With the introduction of Web Services, terms such as “Web Services composition” and “Web Services flow” were used to describe the composition of Web Services in a process flow. More recently, the terms orchestration and choreography have been used to describe this. Orchestration describes how Web Services can interact with each other at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional, multi-step process model. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners. Choreography is typically associated with the public message exchanges that occur between multiple Web Services, rather than a specific business process that is executed by a single party.
There is an important distinction between Web Services orchestration and choreography. Orchestration refers to an executable business process that may interact with both internal and external Web Services. For orchestration, the process is always controlled from the perspective of one of the business parties.

Choreography is more collaborative in nature, in which each party involved in the process describes the part they play in the interaction. The term Web Services orchestration is used here to describe the creation of business processes, either executable or collaborative, that utilize Web Services.

There are a number of important technical requirements that must be addressed when designing business processes involving multiple Web Services running over a long duration. Knowledge of these requirements will help in positioning the various standards that have been introduced for Web Services orchestration.

The ability to invoke services in an asynchronous manner is vital to achieving the reliability, scalability, and adaptability required by today’s computing environments. With asynchronous support, a business process can invoke Web Services concurrently rather than sequentially in order to enhance performance. For example, a purchasing system may want to interact with multiple supplier Web Services at the same time, looking to find the supplier that can offer the lowest price or earliest shipment date. Asynchronous support can be achieved through Web Services through various correlation techniques. Orchestrated Web Services that are long-running must also manage exceptions and transactional integrity.

2. Choreography versus Orchestration, WS-CDL versus BPEL

Steve Ross-Talbot is co-chair of the Web Services Choreography working group of W3C and also the chair of Web Services Coordination, and the following is based on his views (Bharti, 2005).

As companies increasingly use service-oriented architecture, the Business Process Execution Language (BPEL) is becoming an important means to weave Web Services into meaningful business processes. While BPEL allows existing services to be orchestrated into composite services, the Web Services Choreography Description Language (WS-CDL) goes a step further and describes the relationships between services in a peer-to-peer scenario.

Any real business transaction is not just one function call, it is a sequence of function calls and it may be lots of things in parallel between different services that occur. BPEL is about “how do I construct Web Services out of existing Web Services”. In other words, BPEL is about the orchestration of existing services to yield another service.

Choreography is complementary to that. WS-CDL is an unambiguous way of describing the relationships between the services in a peer-to-peer collaboration.
without necessitating any orchestration at all. That is important because if one business and another business are talking together, it is clear that neither partner would accept orchestrating the other's services.

On the orchestration side of it, which is what BPEL does, that comes in on both sides of the firewall in order to reuse services. In an orchestration a service from elsewhere may be used and presented as one’s own service; however that is a case of really calling out to another service. With BPEL one is, in a sense, the conductor in the pit.

In any dance, however, you never see somebody on stage telling the dancers what to do. The dancers have a choreography description. They learn their dance and they learn the “touch points” where they interact with others, the same way one does in a choreography description.

Can both Web Services choreography and Web Services orchestration be specified using a common language? No, because orchestration has more to do with the recursive composition of services so that it will yield another service. Orchestration gets realized as an executable, so you can identify the conductor in the pit and realize that he/she is as much an actor as the violinist.

Choreography, on the other hand, is a description. The choreographer writes the descriptions down and gives it to the dancers and works with them to make sure they learn their parts, but he/she is not there as an “executable actor” when it is happening. So choreography is purely a description, but it is a description that can be used to generate the behaviour of the dancers. It can be used to generate, but not to execute.

3. Orchestration through XLANG (Microsoft), WSFL (IBM), and WSCI

Two proposed XML grammars for describing the orchestration of Web Services – Microsoft's XLANG, and IBM's WSFL (Web Services Flow Language) – were expected to have merged into a joint World Wide Web Consortium (W3C) submission. That did not happen. Meanwhile, Sun, BEA, SAP, and Intalio introduced a third candidate: WSCI (Web Service Choreography Interface). The relationships among these three proposals – and others, including Intalio's BPML (Business Process Markup Language) and ebXML's BPSS (Business Process Schema Specification) – are not clear.

XLANG, WSFL, and WSCI talk about two different orchestration layers. One layer deals with the public protocols of business collaboration, which WSFL and WSCI call the global model. The other layer describes private protocols, which WSFL calls the flow model. XLANG addresses both layers, but less explicitly. Ideally, a W3C recommendation will emerge that splits these apart neatly, but it is not yet clear how to do that.

All three XML grammars define standard programming language constructs for sequences, loops, spawning, conditional execution, and exception handling. XLANG and WSCI have roots in a formal algebra called pi-calculus (developed by Robin
Milner, Computing Laboratory, University of Cambridge), which is used to model the kinds of parallel, message-driven computations that make orchestration a uniquely difficult problem.

XML is a very poor syntax for programming languages, and is not programmer-friendly. XML could be used to specify service choreography but it ought not be used to implement it. A conventional syntax can do this more naturally (using, e.g., Java or C#). Java implements the conversational Web Services defined in BEA's WebLogic Workshop, and Collaxa's ScenarioBeans mixes workflow tags with Java, creating a JSP (JavaServer Pages)-like metaphor that aims to be intuitive for developers.

Shared public protocols, however, must be language-neutral, which mandates XML. At this level, all three proposals – XLANG, WSFL and WSCI – explore grammars that describe dynamic interplay among services offering static WSDL (Web Services Description Language) interfaces. With respect to loosely coupled, asynchronous services, this is where the real test arises. Problems that far-flung service orchestration forces us to confront include message correlation, long-running transactions, and human usable abstractions.

SOAP (Simple Object Access Protocol), as such, is not directly relevant. Orchestration, in the general case, is a many-to-many conversation, and one in which not every participant will necessarily even speak SOAP.

Even when everybody does speak SOAP, the nature of the message fabric is unclear. SOAP routing seems a likely substrate, but is not a necessary building block.

The example illustrated in the XLANG, WSFL, and WSCI specifications is an airline ticketing scenario involving a traveller, an agent, and an airline. XML can represent this scenario in a way that enables diverse software systems to collaborate. But no human reader (or writer) of these documents can hope to understand the dynamic behaviour in terms of the XML.

It is likely that a standard diagramming system, perhaps based on UML (Unified Modelling Language), will play an important role in orchestration. But diagrams have their limitations. There is additionally a perceived need for more background information – ultimately more use cases.

Having looked at scope and limitations of orchestration and choreography in a Web Services context (see also Englmeier, 2005), we will now turn attention to the JXTA open source peer to peer environment. Based on this, we will proceed to look at how JXTA can support both Web Services and also emerging semantic web standards. By means of the latter we will target distributed user interaction based on natural language.
4. A Short Overview of the JXTA P2P Platform

P2P computing is a network model where possibly any node (peer) may be interconnected. A peer may also operate as a consumer or provider, unlike the traditional client/server architecture.

JXTA (JXTA, 2003) is an open source project (created by Sun Microsystems) that defines the following set of six language independent protocols (Protocols, 2002) to provide a P2P infrastructure:

- Peer Discovery Protocol (PDP) Peer/Resource discovery
- Peer Resolver Protocol (PRP) Generic query service
- Peer Information Protocol (PIP) Net monitoring
- Pipe Binding Protocol (PBP) Addressable messaging
- Rendezvous Protocol (RVP) Propagation service
- Endpoint Routing Protocol (ERP) Routing

These protocols establish a ubiquitous, secure and pervasive virtual network (Traversat et al., 2002) on top of Internet and non-IP networks, allowing peers to directly interact and be organized independently of their network locations, even behind firewalls and NATs (network address translation).

JXTA has been designed with the objective of making it ubiquitous. Any device including mobile phones, PDAs, pagers, electronic sensors through to desktop computers and servers, has the potential of being JXTA-enabled. In order to maximise its platform independence and interoperability, widely accepted standards like XML form the basis of JXTA communications. Security (Security, 2001) is considered through the use of Transport Layer Security (TLS), Digital Certificate and Certificate Authorities.

JXTA protocols may be used to build different types of application, e.g. instant messaging, file sharing, data search, remote sensor communication, agent based systems (Babaoğlu et al., 2002). It can also be used to implement a wide range of services. Thus, this technology provides the ability to integrate heterogeneous information sources in a decentralized, self-organized and secure way.

4.1 JXTA Architecture

Figure 1 shows an abstraction of the JXTA platform architecture (Wilson, 2002).
4.2 Core Layer
This layer contains elements that are essential to a P2P solution. The elements of the core level are listed below:

- Peers
- Peer Groups
- Network transport (pipes, endpoints messages)
- Advertisements
- Entity naming (identifiers e.g. peer IDs)
- JXTA protocols
- Security and authentication

4.3 Services Layer
The services layer provides network services that are considered to be desirable but not totally necessary to all P2P solutions. Such services provided include:

- Searching for resources on a peer
- Sharing documents from a peer
- Performing peer authentication

An example of a JXTA service is the Content Management System (CMS, 2002) that allows the sharing and retrieval of content within a peer group

4.4 Applications Layer
The dividing line between the services layer and the application layer can be hard to distinguish. Usually anything in this layer builds upon the functionality provided by the service layer. Instant messaging is an example of something that lies in the application layer.

JXTA uses asynchronous uni-directional pipe communication channels known as pipes for sending and receiving messages. The presence of peers, peer groups and
pipes are all advertised through the use of XML advertisements. Sun has provided an open source Java implementation of the JXTA protocols as well as some core services to provide extra functionality. Examples include CMS, and JXTA Search that provides a more efficient searching mechanism than a traditional P2P search that sends requests to all known peers, hence creating excessive network traffic.

To serve as both a proof of concept and to make some evaluation of JXTA as a development platform a series of additions/modifications to the JXTA platform, CMS and InstantP2P (InstantP2P, 2002) user interface were made in order to set up a Simple Image Service that would supply a user with some image retrieval and processing functionality in a P2P manner (see Johnstone et al., 2005). InstantP2P (also known as myJXTA) is an application freely available from the JXTA site that gives the user a GUI (graphical user interface) giving access to peer discovery, group membership, file sharing and chat functionality through a window-based interface.

5. JXTA versus Web Services

P2P has a broad definition but Web Services are intended to promote interoperability, and extensibility among various applications, platforms and frameworks, by means of externalizing and modularizing application functionalities as sets of interoperable Internet services. It can be admitted that currently P2P – JXTA in our case – and Web Services belong to two quite distinct areas. Different identification and search (or location) functionalities are used in these two areas. Let us look a little closer at these distinctions (Qu and Neijdl, 2004).

A Web Service architecture is typically composed of service provider, service registrar, and service requestor. Associated with these are three types of operation: publish, find, and bind. These entities and associated operations work in concert to provide a loosely coupled computing paradigm, described in such standards as: SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discover and Integration) and WSDL (Web Services Definition Language). SOAP provides the means for communication between Web Services and client applications. UDDI is used to register and publish Web Services and their characteristics so that they can be found by potential client applications. WSDL is used to describe the interface of Web Services. Within the Web Services world, service providers, service requestors and service registrars leverage the UDDI-based centralized discovery mechanism to locate each other and further achieve various functionality interactions.

In JXTA there is no clear differentiation between the service provider, service requestor and service registrar. A JXTA peer typically implements one or more of the JXTA protocols, and may play multiple roles at the same time. JXTA mainly depends on decentralized discovery mechanisms for searching and locating peers and P2P services, although it does not essentially exclude other discovery mechanisms such as the centralized ones. However, as the current decentralized discovery mechanism of JXTA is implemented based on the JXTA Search peer discovery protocol, which is not compatible with UDDI, JXTA P2P services cannot directly be advertised or discovered in the Web Services world. Likewise, the UDDI-based centralized Web Services registry cannot directly be integrated into JXTA and used by JXTA peers for discovering and binding Web Services.
Next there is a problem with perspectives on the transport layer. At the highest abstraction level, JXTA is nothing more than a set of protocols, each of which is defined by one or more messages exchanged among participants. With the focus on decentralization and ubiquitous computing, JXTA neither mandates the transport protocol nor mandates how the messages are specified or propagated. In contrast, although theoretically Web Services do not also mandate the transport protocol, since they principally depend on SOAP for XML based messaging, they practically use HTTP (Hypertext Transfer Protocol) for transport. This implies that there might exist the transport layer incompatibility between a Web Services node and a JXTA peer, which could lead to communication blockage during the interaction. In addition, with regard to the message protocol, although the current JXTA specification adopts XML format to define messages, the definition is not compliant with SOAP. This makes it rather difficult to achieve the direct SOAP RPC (Remote Procedure Call) between JXTA and Web Services.

Another difficulty is service description. Web Services use WSDL to describe services and their invocation details, whereas JXTA uses XML structured service advertisements to describe P2P services. In JXTA, a P2P service is generally described by three types of JXTA advertisements: the module class advertisement, module specification advertisement and module implementation advertisement, which jointly describe the same JXTA service at different abstraction levels. Although all these abstraction levels can find their conceptual mappings in WSDL, the description formats of both are not compatible. Moreover, as JXTA and Web Services adopt different entity identification systems and also use different protocols for the service invocation, the JXTA service advertisements cannot provide a Web Services node with necessary information for locating the access point of P2P services and further invoking these services. Likewise, the SOAP RPC information provided by WSDL also makes no sense for JXTA peers that wish to invoke Web Services, as the SOAP RPC is not yet supported in JXTA.

In regard to security there are differences also. The security of the JXTA platform is implemented based on the so-called “web of trust” security model. A JXTA peer can loan out its credentials to another peer and a community is bootstrapped by linking trusted peers. As a Web Services node does not conduct any JXTA configuration, it has no chance to participate in an JXTA community and sequentially interact with JXTA peers and P2P services.

To overcome these incompatibilities between Web Services and JXTA, Qu and Neijdl (2004) propose Web Service proxies. A Web Service proxy is a JXTA peer, which also possesses a SOAP implementation module responsible for mediating the SOAP RPC between JXTA and Web Services. In order to be identified outside of JXTA, unlike usual JXTA peers, the Web Services/JXTA proxy has to expose its IP address besides its usual JXTA ID. It actually plays a double role, being able to directly address a few of aforementioned challenges such as the entity mapping and the security management. The proxy is a JXTA node that “speaks” Web Services.
6. JXTA Search for Flexible Resource Discovery

P2P and Web Services are complementary technologies. While Web Services provide the universal information architecture to solve the functional integration problem, P2P provides the distributed network architecture that takes us directly and securely to the sources of information on the Internet. The interaction between the two platforms may potentially combine advantages of both, leading to a promising distributed computing architecture converging P2P and Web Services.

Resource location is at the heart of any middleware service. JXTA Search is a decentralized P2P search engine in which information providers publish a description of queries they are willing to answer. JXTA Search uses specialized peers, called hubs, that can register with other hubs as information providers.

The JXTA Search discovery and access model uses a query routing protocol (QRP) which is text based, i.e. XML-based (Waterhouse et al., 2002). Therefore JXTA Search implementations are platform and language independent. Registration caching, indexing, and other tasks, are strictly implementation dependent. Opportunities for extending JXTA Search are manifold. For example, the JXTA Search platform supports arbitrary XML so it can integrate other standards such as the resource description framework, RDF, for describing metadata. One can plug categorisation, classification, or relevance approaches for refining searches into the basic framework.

Qu and Nejdl (2004) discuss the exposing of existing JXTA services as Web Services; and also integrating Web Service enabled content providers into JXTA. Service layer interaction is discussed, linking JXTA and more general Web Services through so-called Web Service/JXTA proxies. The JXTA-based P2P semantic web application, Edutella, is used. Edutella accommodates heterogeneous resource metadata repositories through use of RDF. It is assumed that all resources can be described in RDF, and queries are mediated through RDF statements.

Hajamohideen (2003) discusses the use of SOAP services as Web Services in JXTA. The SOAP service is a customization of the peer group service with ability to send and receive SOAP messages via a JXTA transport. In addition to modules that advertise the existence and implementation of a SOAP service in JXTA, there is also a set of supporting classes needed for SOAP communication. Since the SOAP service uses SOAP over JXTA, it can be realized as a Web Service so long as the discovery mechanism is equipped with a Web Service description such as WSDL.

The JXTA model specifies how services other than the core services are published using the module advertisements and discovered using the discovery service, but not the issue of how they are invoked. The service description regarding how to invoke a service, the details of the operation, and parameters, are all assumed by the client peer that wants to access the service. The JXTA client does not require prior knowledge of a Web Service or the invocation framework of any specific Web Services toolkit. The implementation relies on "just-in-time" creation of dynamic proxies generated from a WSDL document. The proxy uses SOAP, on the wire, to communicate with the Web...
Service, but the client does not need to understand SOAP. The proxy is implemented as part of a JXTA service. This can be done without any change to the module advertisement framework and allows a peer in the JXTA network to access a Web Service through a JXTA service.

7. System Design and Rationale for Prototype

In this section we will discuss our near future objectives at a general level.

We will establish two JXTA peer nodes, one of which will act as a hub. It will contain some simple Web Service functionality – record update/delete/add – described in WSDL. Thus the JXTA hub will also act as a Web Service proxy.

Communication in JXTA is by means or pipes. JXTA services – at hubs – are defined in terms of XML. By constraining the description of such a service to be in SOAP, we will support JXTA-based access to a Web Service respecting the WSDL description standard.

Currently many practical Web Services use standard, off-the-shelf, software libraries and components. Examples of these include: Apache (open source web server), Tomcat (enabling J2EE; an alternative is JBoss which is both openly and commercially available, and uses Tomcat), and struts. These middleware layers support the provision of dynamic content and forms-based server-user interaction. As is usual with middleware of this type, the overall goal is to separate business logic from code, against the background of supporting dynamic content. The middleware layers above adhere to – or can be made to adhere to – description and access standards such as UDDI, SOAP and WSDL. As already noted, JXTA services can easily accommodate any self-describing services that use these description standards.

Let us address the reasons why we use JXTA.

1) For Web Service and JXTA coexistence and cooperation, we have pointed in particular to such work as Hajamohideen (2003) and Qu and Nejdl (2004), both of which clearly show how the JXTA native-mode XML description can function seamlessly with SOAP and WSDL as used by Web Services.

2) In line with some other peer-to-peer systems JXTA supports and favours stable ontologies. (Cf. P2P systems for music delivery and the ontologies used in such areas.)

3) Search and resource discovery functionality for Web Services is supported, through self-description of the Web Services in XML, and by implication SOAP and WSDL.

4) Again through XML self-description, free text – hence natural language – querying is supported.

5) JXTA is based on open source, Java code, and has a large developer community.
6) JXTA supports decentralized peers and thus is flexible and robust.
7) In light of the above we have great hopes for JXTA’s ease of handling the ontologies defined by other WS-Talk partners; and comprehensively supporting WSDL-based Web Services.
8) The one area where JXTA comes off second best compared to, for example, the OGSA Grid standard is in certificate-based user authentication (but cf. Security, 2001). However we have found that OGSA authentication is somewhat top-heavy and onerous in the early stage’s of a project’s lifetime. In addition, a common role played by OGSA is to manage access to, and operational use of, important, distributed computational resources. This is not our problem in WS-Talk. Finally, in WS-Talk, we can of course easily set up Web Services whose role is precisely to handle authentication.

Our objective (see Elenius, 2005) is to merge (i) Web Services for description, (ii) semantic web for search and support of ontologies, and through ontologies, natural language, and (iii) peer-to-peer networking because it provides us with a convenient transport layer. Therefore the protocols used will be WSDL for Web Service description; DAML which has now become OWL for ontology description; SOAP, Simple Object Access Protocol, for access and transfer; all based on JXTA using pipes or sockets as a transport layer.

8. Prototype

In this section we describe our on-going work aimed at realizing the viewpoints described in the previous sections.

8.1 JXTA Web Services Interaction

To show integrating viability between Web Services and JXTA we have designed a simple prototype as a proof of concept.

Our prototype is based on two peers, one that act as a Client or service requester, and another that acts as Server or service provider. More details can be found in the following section.

Client
- Start up peer, join (or create) a peer group using JXTA.
- Search and discovery services, described in a WSDL file.
- Invoke services through JXTA-SOAP.

Server
- Start up peer, join (or create) a peer group using JXTA.
- Publish a service through a module advertisement described in a WSDL file.
- Wait for calls.
- If a service call is received answer it through JXTA-SOAP.
The prototype is being developed with the Java programming language in conjunction with a series of open source libraries such as Apache Ant, Axis (from Apache), and JBuilder 2005 Foundation.

8.2 JXTA and Simple Access Object Protocol (SOAP)

Figure 2 describes an UML sequence diagram to represent our approach to integrating JXTA and Web Service technologies.

![Figure 2. JXTA / Web Services integration](image)

SOAP is used pretty much in the same way as a Remote Procedure Call (RPC), with the advantage that its content is described as an XML file.

8.3 Web Services Description Language (WSDL)

WSDL is used to describe the service’s programming interface. Figure 3 shows a generic representation of a WSDL file, which defines the service’s port (method or functions to be invoked), their messages (input/output parameters), parameter types, and binding to the communication protocol (in this case SOAP). Following this representation we have created a simple “Hello” WSDL file shown in Table 1 that is used in our prototype.

---

1 This is an adaptation of the model originally designed by Kevin Burton (http://soap.jxta.org), 2002. More recently it was taken and updated by Daniel Elenius.
Figure 3. WSDL file format.
8.4 JXTA Start Up

To initialise a peer, to function either as a client or a server, a registration process is necessary. This process allows the peers to “see” each other, and this is done just one time. This registration process involves the creation of a peer name, peer group, and password. Further information such as the specific rendezvous peer can be also specified. Once the process is finished our peer will be available with a unique number to be identified, as well as its membership in a peer group. Figure 4 shows a standard JXTA registration window. The information collected in this registration process is stored in a folder called “.jxta” which is read every time a peer starts.

Table 1. “Hello” WSDL file representation.
8.5 Current Prototype Status

At the moment we have implemented the following tasks in our prototype

Client
- Start up peer, join (or create) a peer group using JXTA.
- Services described in a WSDL file
- Deploy SOAP

Server
- Start up peer, join (or create) a peer group using JXTA.
- Publish a service through a module advertisement described in a WSDL file.
- Deploy SOAP
- Wait for calls
9. Integration with Other WS-Talk Components

9.1 The WS-Talk Architecture

In AKRA report Demostrator_draft V1.5, p. 9, there is a schema that shows a JXTANode, and JXTANode proxy. A JXTA service can be plugged in there. We expect that in this way a Web Service will be interoperable with a JXTA peer.

9.2 Current and Future Work

JXTA pipes vs. JXTA Sockets: When an advertisement is generated a pipe is used to communicate with other peers. This is the standard way in which JXTA peers communicate with each other. Since pipes are unidirectional and asynchronous we are exploring and testing other communication methods like JXTA Sockets. If these can be shown to be more reliable than unidirectional (or bidirectional) pipes we are willing to use them instead.

Discovery and Searching mechanism in JXTA: Further research is needed regarding the discovery and searching mechanism in JXTA. At the moment we have encountered a series of difficulties when looking at and retrieving services, due mainly to the asynchronous way in which JXTA works. Looking into more detail as to how exactly this mechanism is implemented in JXTA will improve significantly the performance of our prototype.

Implementation of a rendezvous peer: When a peer starts, a registration process is carried out. This allows the peers to create new groups to join, as well as assigning a unique identification number (UID) for communication proposes. At the moment we are relying on Rendezvous peers (servers) set up by Sun Microsystems regarding registration. We are considering the option of implementing this registration functionality ourselves, since WS-Talk will need its own registration method.

Integration with HTTP server (Apache Tomcat): Regarding potential problems encountered that have not been thought through interoperability testing is necessary. That is mainly between Web Services deployed as JXTA services and Web Services deployed as an HTTP service (with Apache Tomcat).

Ontology languages and protocols: Once a stable prototype has been designed and the communication between peers has proven reliable, ontology integration will be our goal. This will lead to other areas such as orchestration and choreography of Web Services as described earlier in this paper. On this basis, then, we will support natural language use of Web Services.
Acknowledgements

Research outlined in this paper is part of the project WS-Talk that is supported by the European Commission under the Sixth Framework Programme (COOP-006026).

Acronyms

BPEL – Business Process Execution Language
BPML – Business Process Markup Language
BPSS – Business Process Schema Specification
CMS – Content Management System
DAML – DARPA Agent Markup Language
ERP – Endpoint Routing Protocol
GUI – Graphical User Interface
HTTP – Hypertext Transfer Protocol
IBM – International Business Machines Corp.
ID – identifier
IP – Internet Protocol
JSP – Java server pages
NAT – network address translation
OGSA – Open Grid Services Architecture
OWL – Ontology Web Language
P2P – peer to peer
PBP – Pipe Binding Protocol
PDP – Peer Discovery Protocol
PDA – portable digital assistant
PIP – Peer Information Protocol
PRP – Peer Resolver Protocol
QRP – Query Routing Protocol
RDF – Resource Description Framework
RVP – Rendezvous Protocol
RPC – Remote Procedure Call
SOAP – Simple Object Access Protocol
TLS – Transport Layer Security
UDDI – Universal Description Discovery and Integration
UML – Unified Modelling Language
W3C – World Wide Web Consortium
WSCI – Web Service Choreography Interface
WSDL – Web Services Definition Language
WS-CDL – Web Services Choreography Description Language
WSFL – Web Services Flow Language
WS-Talk – Web Services Communicating in the Language of Their User Community
XML – eXtensible Markup Language
Bibliography


InstantP2P Project Home. Website retrieved on September 12 2003 http://instantp2p.jxta.org/servlets/ProjectHome


