Position Papers for the World Wide Web Consortium (W3C) Workshop on Web Services

W3C Web Services Team
Contact: Harumi Kuno
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-73
April 3rd, 2001*

E-mail: hkuno@hpl.hp.com

Web-services, W3C, E-services, transactions, electronic commerce

This technical report is a collection of position papers that HP submitted to the World Wide Web Consortium (W3C) Workshop on Web Services. The W3C Web Services Workshop represents a community interested in XML-based Web service solutions and standardization of components thereof, including both solution providers and users. The goal of this workshop is to advise the W3C about which further actions (Activity Proposals, Working Groups, etc.) should be taken with regard to Web services.

Introduction

The W3C Web Services Workshop represents a community interested in XML-based Web service solutions and standardization of components thereof, including both solution providers and users. The goal of this workshop is to advise the W3C about which further actions (Activity Proposals, Working Groups, etc.) should be taken with regard to Web services.

In order to present a coordinated position, researchers from HP Labs (both Palo Alto and Bristol) and representatives from both the E-Speak Organization and the E-process Operation collaborated closely (by telephone and email) to produce nine position papers reflecting HP's position on topics ranging from conversational interfaces for web-services to electronic contracts to service management. This technical report is a collection of the position papers that HP submitted to the World Wide Web Consortium (W3C) Workshop on Web Services. (The order in which they appear reflects the infrastructure stack sketched in the "HP Web Services Architecture Overview" paper.)

Table of Contents

HP Web Services Architecture Overview

Kannan Govindarajan, Arindam Banerji

Email: kannang@hpl.hp.com, axb@hpl.hp.com

1. Abstract:

Web services are different from traditional distributed computing models. Web services architectures provide a framework for creating, and deploying loosely coupled applications. One of the consequences of the loose coupling is that any entity that a web service may interact with may not exist at the point of time the web service is developed. New web services may be created dynamically just as new web pages are added to the web and web services should be able to discover and invoke such services without recompiling or changing any line of code. In this position paper, we outline some of the high-level architectural requirements of a comprehensive framework for web services, we propose a layered approach to architecting web services that allows for pluggability and interoperability. In addition, we distinguish between infrastructure services and application specific frameworks.

2. Introduction:

The web service architecture allows businesses to expose business assets as services. Standardizing interactions amongst services has the added advantage that any enterprise can out-source parts of its operation that it does not have expertise in. In addition, since the vision of web services enables web services to dynamically find new web services that it can interact with, enterprises can find new providers for the service relatively quickly. A specific application of this dynamism is in the e-procurement arena. For example, the average sourcing/procurement cycle in enterprises is of the order of 3-4 months. Of this time, about 50% of the time is spent in identifying the appropriate suppliers, about 20% of the time in handling the RFQ (request for quotes) process, and an additional 10% of the time is spent in negotiating the appropriate deal. The ability to dynamically find suppliers can translate to significant time savings, and therefore to lowering of costs. Essentially, the procurement and fulfillment business process are modeled as services, and a hub is the aggregation point for the services. In such an architecture, finding a new supplier is the same as finding the fulfillment service of the supplier at the hub. HPs web services vision enables such a dynamic world by allowing business processes to be modeled as web services, by providing a platform for hosting such web services, by defining the technical conventions that enable the interoperability between web services, and by defining a hub, or aggregation mechanism for web services.

It is our position that fundamentally different component model is required for modeling web services. This is because the assumptions that are made by traditional distributed component models are violated by web services. In addition, we believe that a comprehensive web services platform has at least three related technologies:

1. The technologies that define the hosting platform that hosts services. Service providers typically will host their services on this hosting platform.
2. The technologies that define the hub that allows services to dynamically discover other services and establish trust in the context of the community. This hub technology potentially is compatible with other hub-like efforts such as UDDI (www.uddi.org).
3. The technologies that define the standard conventions that ensure that services can inter-operate with each other irrespective of their implementations.

The hosting platform provides, among other things, technologies that are required to model existing business asset/process as a web-service, allows clients of web services to invoke services, etc. The hub provides technologies for web services to be described, discovered, etc., and the standard conventions specify the things that have to be standardized so that web services hosted on various web service platforms inter-operate. We at HP have found these three technologies to be extremely relevant in the mobile services and electronic marketplace deployments. We now briefly provide an overview of these scenarios. The mobile service we consider provides a sales representative access to manuals from his cell phone and allows him to print the manuals on a printing service that may be determined by his current location. For example, an architecture of a mobile online printing solution is shown in Figure 1:

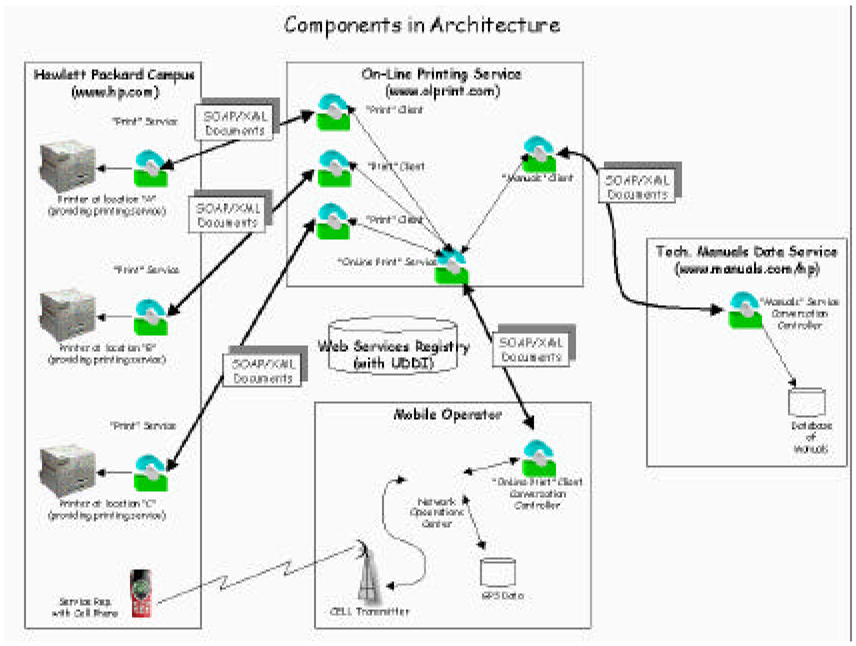

Figure 1 - Components of a mobile online printing solution.

The architecture of an electronic marketplace is sketched below in Figure 2. Note that the two architectures are quite similar -- the MNO in the mobile solution serves as the aggregating and intermediating entity, whereas the hub in the electronic marketplace serves as the aggregation point and may intermediate the access to services hosted on it. We suggest that W3C standardize the technical conventions that enable web services to inter-operate with other web services. The conventions that need to be standardized fall into two categories: infrastructural and domain specific. Infrastructural conventions should be separated into many layers that provide the functionality that essentially allow web service creation, deployment, interaction, and execution. In addition, specific web service infrastructures may also standardize common functionality that is applicable in specific domains. Two domains of particular interest in the web services space are electronic marketplaces, and mobile services. In addition, it is important for W3C to keep all the web services related conventions coordinated in order to enable a coherent platform.

3. *Infrastructure for Web Services*

The basic infrastructure for web services serves the same purpose that CORBA and COM serve for traditional distributed computing infrastructures. However, there is an important distinction between the infrastructure for web services as opposed to traditional infrastructures. The inter-operability problem among web services is more challenging than the inter-operability between traditional distributed enterprises. One reason for this is that web services may be separated by firewalls and the semantics of data being communicated between them may not be uniform at the communicating ends. Another important distinction between web services and traditional distributed component models is that the binding between carious web services is looser and can occur later than the binding between various components of a traditional distributed application.

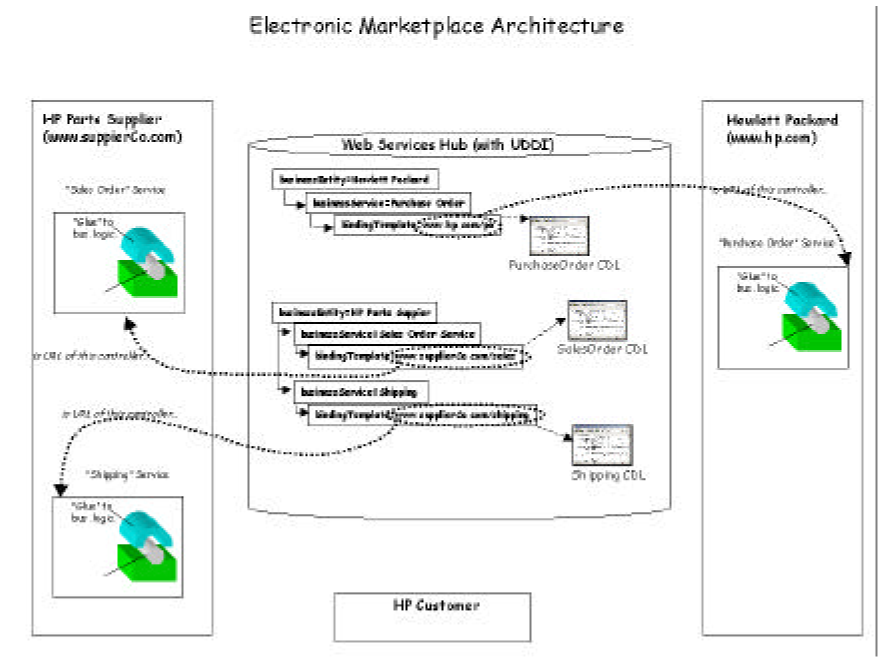

Figure 2 - An architecture for an Electronic Marketplace.

Assumptions

Note that one of the key differences between the traditional distributed computing models and web services infrastructure is that the message formats need standardization. This is in addition to standardizing the APIs needed to access the functionality. Standardizing just the APIs as the Java ™ distributed computing platform has done leads to a situation where there are many incompatible implementations. For instance, at the time of writing this position paper getting two JMS implementations to communicate was still an open issue. We believe that the message formats should all be characterized as XML messages and W3C guide their definition so that different web service infrastructure implementations can inter-operate. In addition, as mentioned before, web services are fundamentally different from traditional distributed models for the following reasons:

Web services are loosely coupled: Changes to a web service should not require re-installation of software components by the users of the web service.

Web services require dynamic binding: Typically, application designers bind software components to one another at development time. Web services, on the other hand, are likely to be implemented and provided by different service providers. In addition, we must enable easy changes to the services we are using, easy discovery of new services, of new capabilities of existing services, and of new binding or location information of services.

Web services communication is based on a Document Exchange Model: Traditional component frameworks support a network-object model of interaction in which objects of strictly defined types are transferred between components using a request-response interaction pattern. Cross-organizational business interactions do not fit this framework well for two reasons. The interfaces of services may need to be changed in ways that cannot be captured by simple extensions. This precludes the use of object inheritance to support the inter-operability in presence of change. Secondly, interactions can be long lived. Therefore, asynchronous exchange of XML documents is better suited for cross-organizational business transactions.

Web services in different enterprises are likely to use different semantics. The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data often tends to be similar or homogeneous contributing to tighter coupling.

Web services in different enterprises require a distributed model of security. Security responsibilities are split amongst the enterprises. Each enterprise manages its end of the security infrastructure independently.

Web services from different enterprises may be built using heterogeneous technology stacks. Each enterprise decides on the computing infrastructure independently taking into consideration many factors.

Web service interactions must be able to traverse corporate firewalls. Traditional distributed systems are tuned for applications that are deployed within the enterprise; Web services may be deployed from behind Firewalls. They may need to access other web services across enterprises. For example, in the J2EE™ architecture, the application are written to receive input from outside the enterprise, however, the applications do not often initiate outbound requests to applications in other enterprises. In the realm of web services, accessing applications in other enterprises is the norm rather than the exception.

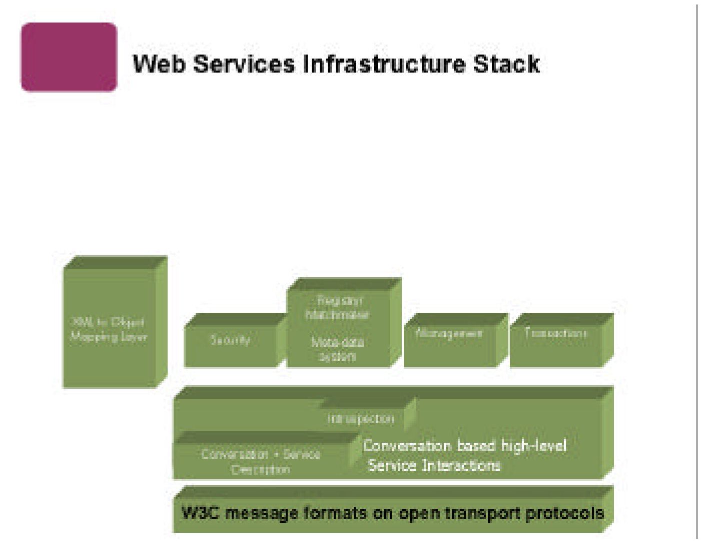
Web Services Infrastructure Stack
XML to Object Mapping Layer
Security
Registry Matchmaker
Meta-data system
Management
Transactions
Introspection
Conversation + Service Description
Conversation based high-level Service Interactions
W3C message formats on open transport protocols

Figure 3 - The layers of an infrastructure for web-services.

As shown in Figure 3, the infrastructure for web services is made up of the following layers:

The communication layer: The communication layer should be transport independent though bindings for common transport protocols such as HTTP, SMTP, etc., should be defined. Essentially, this layer answers the question: How do web services communicate with each other over standard protocols? The messaging layer provides the requisite properties of the messaging such as reliability, security, etc. In order for web services to communicate with each other they have to agree on the technology, standards and protocols for communication. They also need to agree on the syntax and semantics of data they are going to exchange. However, the data that they exchange can be classified into infrastructure parts that the infrastructure uses in order to route the message and application specific parts that the infrastructure does not inspect.

The Service definition layer. Essentially, this layer is concerned with defining how services expose their interfaces as well as the service invocation model. It addresses question such as do the web services expose a RPC like invocation model or do they expose a completely asynchronous document exchange model. The invocation model rests on the messaging model. Since the programming model for web services is more likely to be document exchange based, it is difficult to adopt an object interface definition language for the purpose of defining interfaces of web services. Another aspect to keep in mind when defining interfaces in terms of message exchanges is that the order of message exchanges plays a role in the interface definition. In addition, the security infrastructure has to provide mechanisms that allow service providers to easily determine whether the invoker has the authorization for invoking the service. It should be noted that the service definition layer provides the platform on which the other infrastructure services and functionality is built on. We at HP have invented CDL (Conversation Definition Language) as a means of expressing the interface of web services in terms of the documents that are exchanged with the service.

Introspection: The web service infrastructure has to define mechanisms that allow clients of web services to introspect web services in order to determine how they should interact with them. There are two conversations that allow services to get information about another service they are interested in. These two conversations support the dynamic interaction by allowing a functionality that is similar to the notion of introspection in traditional object systems. The ServicePropertyIntrospection conversation provides general information about the service, the conversation it supports, its provider, and how to access the service. The ServiceConversationIntrospection conversation returns the complete description of the conversations as CDL documents.

The mechanism for binding to services: This allows clients of services to determine the service they want to use in a declarative manner separate from the interface definition of the service. This essentially involves defining the meta-data of services and registry infrastructure for storing and querying the meta-data of services. Matchmaking is the process of putting service providers and service consumers in contact with each other. The matchmaker is where services that want to be dynamically discovered register themselves or their service offerings, and where services that want to find other services send their request for matches. Some of the services advertising themselves through the matchmaker will be simple end-providers, while others may be brokers, auction houses and marketplaces which offer a locale for negotiating with and selecting among many potential providers. The matchmaker is a very simple, foundational, service on which the rest of the service framework rests, and should be as neutral as possible. It is the web-spider search engine of the web services world. The matchmaker service depends on the meta-data definition system that is provided by the infrastructure. This meta-data system should be flexible and capable of defining the meta-data of a wide variety of services. Some of the requirements of the meta-data definition system are the following. It should be capable of defining the metadata for a wide variety of services. This means that if different vertical industries want to define their metadata for the services in their industry in different ways, the mechanism should allow that. It should allow the metadata to evolve in a gradual manner without changes to the backends or the enterprises that are using old versions of the metadata definition. This allows specific advertisers to differentiate their descriptions of the goods/services they sell with characteristics that enhance their advantage over their competitors. It should be compatible with existing metadata definition mechanisms so that existing web services can be advertised at matchmakers and discovered. It should identify portions of the metadata that are searchable

The security infrastructure: This enables web services to secure various aspects of the service. Traditional PKI infrastructures may not be suited for the web services space. In addition, the security infrastructure will have to provide not only secure transport, but also higher-level access control mechanisms. Web Services or E-Services pose a unique challenge to the design of a distributed security system. At the high level the security system performs the access control, accountability, system integrity and confidentiality.

Unfortunately security systems today cannot do these things well in a distributed environment that spans multiple, independently managed security domains. Some of the specific problems that must be solved are:

- Elimination of man in the middle attacks at boundaries of security domains. Frequently security must be compromised to convert from one security model to another at a border of one or more security perimeters.
- End to End authorization and confidentiality
- Accountability
- The system must not rely upon any central security domain. There are known problems with global name spaces, interoperability of dissimilar security domain infrastructures, the need for trusted third parties at boundaries, etc. The system's objective is to establish an authorization, accountability, and confidentiality agreement between a client and service without depending upon any centralized security domain.
- The system needs to be suitable (scalable?) for a wide range of service value. A service can range from a few cents to potentially millions (billions?) of dollars. While multiple security systems could be invented, tailored to the needs of low or high value services, the challenge is to develop a single security infrastructure that can be readily adapted to either extreme.

Transaction Support: The web services infrastructure should provide support for some notion of transactions so that web services can compose other web services in a transactional manner. Traditional two phase commit protocols may not be appropriate in the context of web services. In addition, transactionality in the web services world may be restricted to providing support for atomicity only. This is in contrast to the consistency, isolation, and durability properties guaranteed by traditional distributed transaction systems. Some of the questions that need to be answered by the support for transactions are:

- Do we allow resource locking?
- What is the model for achieving transactions?
- What protocols do we need to support in order to achieve atomicity?
- Do we make compensation time-bounded?
- Does atomicity apply to conversations or isolated invocations?
- Should we enable composition of different atomicity models?

Management Support: The web services infrastructure should specify how web services are to be managed so that various aspects of the interaction among web services can be managed. There are two interpretations of manageability for services. *Manageability from a management system's perspective* refers to whether a service provides sufficient information (events, measurements, and state) and control points (lifecycle control, configuration control, etc) so that a management system can effectively monitor and control the behavior of the service. There is a second notion of manageability – *manageability from a peer service's perspective*. When two services interact with each other, one of them initiates a request (client) and the other services the request (service). From a client's perspective, a service is manageable if the latter provides sufficient visibility and control over itself and over the transactions or conversations it executes. For example, a service that provides information about the progress of a client's ongoing transactions and an ability to escalate the speed of transactions in progress (perhaps by paying additional price) is more manageable than a service that does not. Similarly, a service that can be queried about how much quality of service (QoS) it can guarantee and that can provide response time measurements about its transactions is more manageable than a service that does not provide these features. From a service's perspective, a client is manageable if it can provide enough information about service usage back to the service. For instance, a client that can be queried about its perception or quality of experience with the service is more manageable than a client that cannot be. Similarly, a client that provides end-to-end response time measurements of its transactions back to the service is more manageable than a client that does not.

4. Domain Specific Infrastructure

We now turn our attention to some of the domain specific functionality that the web services infrastructure should standardize. There are two main domains that we believe web services have a major impact. These are: electronic marketplaces, and mobile web services.

5. Electronic Marketplaces

In the specific domain of integrating business applications across enterprises, web services can play a role in standardizing some of the interactions between businesses. The web services infrastructure provides facilities to find web services and interact with web services. However, in the business to business scenario, the negotiation and contract formation functionality has special relevance.

Negotiation: Standardizing the protocols required for various forms of negotiation such as auctioning, two-party negotiation, etc., allows web services to cleanly attach their own specific strategies while still being able to negotiate with a wide variety of parties in order to complete business transactions. The negotiation framework aims to provide infrastructure that allows two or more independent entities to interact with each other over time to reach agreement on the parameters of a contract. It is aimed primarily, though not exclusively, as a means to reach trade agreements. It can be used both by automated entities, and by users via appropriate software tools. Its value to negotiation participants is that it is a prerequisite to provide decision support or automation of the negotiation, and hence make the process more efficient. Furthermore, they can be confident that the basic rules of interaction in any negotiation are standardised, hence reducing the effort to automate many different kinds of business interactions. They are able to negotiate simple contracts, where only price is undetermined, and more complex contracts where many complex parameters depend on each other. Furthermore, the protocols provide the participants with trust guarantees, that no party has access to extra information or is able to forge false information. Its value to negotiation hosts such as auction houses and market makers is that it provides a standard framework that all potential customers can use to interact with them. However, it does not require a specific market mechanism, so allows the host to decide on an appropriate one. It not only provides standard off-the-shelf market mechanisms such as the English auction, but also allows custom mechanisms to be implemented for particular special needs such as the FCC auction for auctioning bandwidth.

Contract Formation and Business Composition: The central idea of the conceptual model for contracts is that the business relationship that motivates interactions that follow is captured explicitly in an electronic contract. An electronic contract is a document formed between the parties that enter into economic interactions. In addition to describing the promises that can be viewed as rights and obligations by each party, the e-contract will describe their supposed respective behavior in enough detail, so that the contract monitoring, arbitration and therefore enforcement is possible. The terms and conditions appearing in the e-contract can be negotiated among the contracting parties prior to service execution. In this way businesses with no pre-existing relationships can bridge the trust gap, be able to strike deals and take them to completion. Business composition is the ability for one business to compose e-services, possibly offered by different providers, to provide value-added services to its customers. Composition of e-services has many similarities with business process (workflow) automation. A web service virtualizes the customer interaction aspects of the business processes implementing a service. In order to specify how services should be composed, we must define the interaction process associated with a service as well as the flow of service and inter-service invocations.

Conversation Definitions: defining interfaces of web services

Kannan Govindarajan, Alan Karp, Harumi Kuno, Dorothea Beringer, Arindam Banerji

Hewlett Packard Company

10450 Ridgeview Court MS 49EL-FR

Cupertino, CA 95014

Abstract

In this position paper, we advocate a novel methodology for defining interfaces of web services. This novel methodology is motivated by the realization that web-services have unique characteristics not addressed by traditional methods of defining interfaces. In particular, our methodology addresses the fact that web-services are characterized by loose coupling amongst the participating entities, as well as a message oriented interaction model. We identify various characteristics of a desirable solution.

1 Problem Statement

Web services, or e-services are applications that interact over the open internet through the use of standard protocols. In order for web services in one enterprise to interact with web services in other enterprises, we must establish technical conventions for standardizing interactions with web services. These technical conventions range from the messaging formats to interaction definitions, to properties of the interactions such as security, transactionality, etc.

Traditional distributed object models use the concept of interfaces to model interactions. This is a useful technique because the language in which an interface is defined can be completely independent of any language used to implement that interface. However, most traditional distributed object infrastructures were designed for distributed systems whose deployments were limited to within a single enterprise. They are thus suited to deploying services across organizational boundaries due to the following inherent characteristics of the inter-enterprise web services:

Web services are loosely coupled: Changes to a web service should not require re-installation of software components by the users of the web service.

Web services require dynamic binding: Typically, application designers bind software components to one another at development time. Web services, on the other hand, are likely to be implemented and provided by different service providers. In addition, we must enable easy changes to the services we are using, easy discovery of new services, of new capabilities of existing services, and of new binding or location information of services.

Web services communication is based on a Document Exchange Model: Traditional component frameworks support a network-object model of interaction in which objects of strictly defined types are transferred between components using a request-response interaction pattern. Cross-organizational business interactions do not fit this framework well for two reasons. The interfaces of services may need to be changed in ways that cannot be captured by simple extensions. This precludes the use of object inheritance to support the inter-operability in presence of change. Secondly, interactions can be long lived. Therefore, asynchronous exchange of XML documents is better suited for cross-organizational business transactions.

Web services in different enterprises are likely to use different semantics. The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data often tends to be similar or homogeneous contributing to tighter coupling.

Web services in different enterprises require a distributed model of security. Security responsibilities are split amongst the enterprises. Each enterprise manages its end of the security infrastructure independently.

Web services from different enterprises may be built using heterogeneous technology stacks. Each enterprise decides on the computing infrastructure independently taking into consideration many factors.

Web service interactions must be able to traverse corporate firewalls. Traditional distributed systems are tuned for applications that are deployed within the enterprise; Web services may be deployed from behind Firewalls.

6. Characteristics of Solution

It is our position that in the web-services world, there needs to be a clear distinction between the public interface supported by a service and its private process that implements the public interface. This distinction enables flexible and dynamic inter-operability between Web services; for example, a service implementor can re-implement a service as long as it still supports the public interface. We base the definition of these public interfaces on the document exchange model, meaning that service interaction points are defined in terms of the documents that are exchanged with the service rather than method signature definitions.

In addition to defining service interaction points, an enterprise deploying a web service also needs to specify the valid sequences of message exchanges (interactions) that the service supports. For instance, RosettaNet PIPs define the interaction sequences that are specific to supply-chain like interactions amongst enterprises. However, RosettaNet does not support loosely coupled web services, and thus does not provide a generic mechanism for defining new interactions among services running in different enterprises. We identify that a generic, open ability to define the valid interaction sequences as part of the interface definition is necessary for enabling loose coupling amongst web-services. Such definitions would enable clients to determine dynamically the relative order in which the documents are to be exchanged to perform any unit of work with the service.

7. Outline of CDL

The Conversation Definition Language (CDL) is an XML schema for defining valid sequences of documents exchanged between web services. Conversations have been studied in the agent community, and provide a means of defining the interfaces of services in terms of the interactions that the service supports. The interaction definitions are at a level that is higher than the transport layer and allow high-level modeling of the interfaces that the service supports. For example, the details of the transport binding are the responsibility of a separate layer in our architecture and can use existing technologies such as WSDL [1] for this purpose. Essentially, CDL allows web services to model their public process in a lightweight manner. CDL also provides an extensional view of a web service in terms of the messages that are exchanged with it.

Conversation based interface definition enables loose coupling among web services by defining an asynchronous document exchange model, as opposed to a remote procedure call model, for interaction amongst web services. Web services can inter-operate with each other as long as they conform to the conversation definitions exposed by them irrespective of the implementation stack that supports the conversation definitions. CDL as described in this paper suffices to model interactions amongst two web services. We plan to extend it to support secure, mediated, and multi-party conversations.

The notion of interactions amongst services can be expressed in many ways.

One way is to express the concept of the shared interaction that abstracts away the entities in the interactions. CDL, however, specifies the view of the interaction from the viewpoint of the entities that are interacting. This means that a programmer who is programming a service can define the CDL for his service just as programmers today define the interface for the code that they write. In many vertical industries, there may be standard ways of defining conversations that can be found at an internet-wide registry. In this case, the programmer can select the conversation that she wants her service to support. This allows the clients of the service to interact with the service in a well-defined, though loosely-coupled, manner.

The design of CDL separates the mapping of external interactions between services from mappings to internal legacy business logic, which most service implementations will typically encapsulate. This latter mapping is described through business logic mapping layer documents that are outside the scope of this paper. Furthermore, the conversations that a service supports can become one of the pieces of information that is required at service registration, just as tModels for services are required when services are registered at UDDI registries today. Furthermore, the CDL definitions themselves can be listed in registries so that clients can discover the CDL definitions required to interact with services that support the CDL.

One can extend CDL in a variety of ways in order to capture various kinds of semantics. Some of the possible extensions include:

- Disconnected conversations: support for long-lived conversations with mobile entities.
- Multi-party conversations: support for conversations amongst multiple entities.
- Transactional conversations: support for atomicity and other desirable properties for interactions and conversations.
- Secure conversations: support for security.
- Mediated conversations: support for a third mediating entity that can monitor and control the execution of the conversation, like the hub-based messaging in RosettaNet specification.
- Mobile conversations: support for mobile clients in addition to the notion of disconnection.
- Support for events: the ability to define what the conversation definition will look like when support for events is added.

8. Relationship to existing standards

The conversation definition language defines the notion of interfaces and protocols of web services. Here we use the word "protocols" to denote the ordering of invocations. It presents an abstraction that is above the format of the XML message on the wire that other standards such as SOAP define.

For example, an interaction amongst the services could be a SOAP message, where the body of the SOAP message contains the document that represents the request from the client to the service. CDL defines conversations that take place on top of a messaging layer. However, CDL is independent of any particular messaging protocol.

The current version of WSDL (1.0) is an XML-based format that describes the interfaces and protocol bindings of web service functional endpoints. WSDL also defines the payload that is exchanged using a specific messaging protocol; SOAP is one such possible messaging protocol. However, neither UDDI nor WSDL currently addresses the problem of how a service can specify the sequences of legal message exchanges (interactions) that it supports. Like WSDL, CDL does not expose the service implementation. Both CDL and WSDL provide a language to define interactions amongst the entities. Unlike WSDL, CDL relies on a separate specification to define method dispatch, and relegates the binding to various message protocols to a lower layer in the stack.

Note that conversation definitions are not workflow definitions. The key difference between a conversation and a workflow is that a conversation models the externally visible interaction model of the web service. A workflow is one way to actually implement the web service.

9. References

[1] Web Services Definition Language (WSDL) available at http://msdn.microsoft.com/xml/general/wsdl.asp

Advertising and Discovering Business Services

Alan H. Karp, Kevin Smathers
Hewlett-Packard Laboratories
Palo Alto, California
Alan_Karp@hp.com
Kevin_Smathers@hp.com

Abstract

One of the key needs for businesses to operate effectively over the Internet is the ability to discover providers of services that they need. It is our position that the mechanisms used by existing marketplaces and *ad hoc* consortia do not fully meet this requirement because they lack the flexibility needed in the dynamic environment of the Internet. In particular, web-based services need to be able to describe themselves in new ways without undue delay. We believe that vocabularies are a representation of ontologies that are well suited to the business needs. A framework that allows for easy creation, dissemination, and evolution of vocabularies, while accommodating industry standard vocabularies better meets this need.

1. Business Need

As business services move to the web, it becomes increasingly important to automate the way buyers and sellers find each other. Advertising at the Super Bowl is not going to attract software. Clearly, some sort of automated advertising and discovery mechanism is needed. However, blind searching on keywords results in too many false misses as well as too many false hits. What is needed is a context that provides semantic meaning to the search terms.

An ontology provides semantic content to an attribute, allowing parties to agree on the meaning of the value associated with that attribute. For example, in an electrical engineering ontology, GATES might be an integer denoting the number of electronic gates per square inch. In a landscape architecture ontology, GATES might be a string denoting the style of gate. There are many ways to represent ontologies. Examples include hierarchical taxonomies and vocabulary expressed as name-value pairs. We feel that the latter best meets the needs of e-businesses.

Business interactions on the web use ontologies in a way that differs from many common uses, leading to a set of requirements on the infrastructure.

- Adaptability is key. However the ontologies are defined, it must be possible to describe new kinds of things without undue delay. Waiting for the next version of a standard ontology constitutes undue delay.

- Ontologies must be widely available. It doesn't do a business any good to describe itself in an ontology unknown to its potential customers.

- Translation from one ontology to another related ontology is critical. Simply translating between languages is enough to justify this requirement, but the ability to cross market segments that have their own way of describing the same thing enforces it.

- Fixed values registered in a catalog service may not be adequate. For example, advertised prices may depend on who is doing the search or when the search is done. Hence, *dynamic* attributes are needed.

- Complex descriptions and queries must be supported. A business is more than just its list of products and their prices. It is a combination of those factors plus its business practices and procedures.

- Delayed discovery, in which the searcher is notified when a suitable service is found whether or not the initial search succeeded, is needed in many scenarios.

2. Other Ontology Efforts

We have been discovering things on other computers as long as there have been networks, probably even before. The Internet and the growth of commerce on it have dramatically increased both the need for discovery and the difficulty in providing it. While a survey of all ontology efforts would be of value, it is beyond the scope of this position paper. However, understanding why these efforts fail to meet the needs of business being conducted on the Internet is important for understanding our position. We've picked some representative examples to illustrate the problem.

Probably the first efforts attempted to index the entire web. The Alta Vista search engine provides a full text search of every web page it has indexed. A problem with this approach is that you get no hits or 40,000, a phenomenon that has been facetiously called the *Alta Vista effect*. Yahoo also indexed the web, but with human indexers. However, searching requires following a fixed hierarchy, something better for humans than software. Other hierarchical indexing schemes, such as CoS Naming and LDAP, provide wildcarding that avoids some of these problems, but they don't provide any special mechanisms to support ontologies. You can always make a particular node in the hierarchy an ontology node in which points below it are defined in terms of that ontology. Unfortunately, the very feature that makes it easier for software to search, wildcarding, makes it impossible to enforce the use of the ontology.

Object systems, such as CORBA or Jini, provide a means to find objects that implement specific interfaces. However, a business service is more than the interfaces it supports. While Jini is built on top of JavaSpaces, a very general search engine that could be used to provide more complex description and discovery, there is no specific support for ontologies. VerticalNet supports a large number of trading communities. Each has a specialized ontology to enable businesses within that community to find each other. However, there is only one ontology per community, so that additions must wait for approval from a central authority.

More recent efforts to provide description and discovery frameworks are UDDI and ebXML. UDDI Version 1 allows businesses to describe themselves in one or more standard taxonomies, such as NAICS and UNSPSC. However, these taxonomies are not flexible enough for all situations. For example, it is difficult to decide if the category "computer service" is for hardware or software. Version 2 of UDDI, currently being designed, allows new taxonomies to be introduced. However, allowing the description of a business service in detail is not one of UDDI's goals. It is only intended to provide a first level filter; further discrimination is done in direct communication with the service provider. Another standard in the making, ebXML, is similar to UDDI Version 2 in that it allows categorization in different taxonomies. However, ebXML includes more of the other aspects of doing business on the web than does UDDI.

E-speak was designed for doing business in the dynamic environment of the Internet. Business services in an e-speak environment are constructed by specifying the job that needs to be done rather than how the job is to be done. Thus, a business process that needs a billing service describes the properties it is looking for rather than naming a specific billing service. Hence, a rich, flexible description and discovery mechanism was critical to making e-speak useful. It's not surprising that e-speak incorporates many of the features needed for discovery of business services on the web. Our position is strongly influenced by our experience with e-speak, both in understanding the most valuable features of e-speak vocabularies as well as in knowing what extensions would be most useful.

3. Dynamic Vocabularies as Discoverable Services

We believe that businesses will want to use industry standard ontologies, but they need a way to meet special needs on a time scale shorter than these standard ontologies can be updated. We propose a framework for defining *vocabularies* to meet this need. A vocabulary is a particular representation of an ontology that allows a business service to be described as attribute-value pairs. Further, a vocabulary can be advertised as a business service itself in another vocabulary. A very simple *base vocabulary* understood by all is needed to ground the recursion.

This mechanism is best illustrated by an example. Say that I am interested in finding a billing service to incorporate into my business processes. I find general business services vocabularies by doing a lookup in the base vocabulary. I use those vocabularies to find the ones related to business processes. I can then look for the exact service I'm interested in. Note that I may turn up more vocabularies, which I can then use to extend my search. More complex cases involving detailed business services might well go through more levels. However, at the end of the process, I will have found a vocabulary that is rich enough to describe the services in sufficient detail. Of course, the vocabulary creator decides how much detail is sufficient. If a business finds that it needs an extension, all it needs to do is create a new vocabulary and advertise it in the previous vocabulary.

The vocabulary framework must support certain features that fall into 4 categories.

Specification

- It should be possible to specify both the vocabulary and the query in XML. Using XML makes it easier for independently coded applications to use the vocabulary. It also means that a specification for doing translation is in place.
- The vocabulary mechanism must allow for user-defined attribute types. These types can be defined in terms of other user-defined types, but at some point the definition must end in a rich set of architected types. One such type should be a service description in terms of another vocabulary.
- Mandatory attributes, those that must be included in every service description, allow the vocabulary creator to enforce consistency standards.
- Multi-valued attributes are a must. It must also be possible to define attribute values as a range. These requirements can be combined by specifying the allowed attribute values as a constraint on the type.

Advertising and searching

- Businesses must be able to advertise in multiple vocabularies.
- The search language must support complex queries, including inequality and substring matching
- The search mechanism must provide support to express arbitrary boolean relations between terms from different vocabularies.

Evolution of vocabularies

- The vocabulary framework must permit the translation of one vocabulary to a related one.

- It should be possible to evolve a vocabulary without invalidating previous uses.
- Allowing inheritance from another vocabulary will make introducing new vocabularies easier.
- Often two or more equivalent vocabularies may be developed. It is important to distinguish between two references that refer to such equivalent vocabularies as opposed to independent references to the same vocabulary.

Creation and control

- The owner of a vocabulary must be able who can modify its definition.
- Controlling who can advertise in a vocabulary is one way that marketplaces can provide a degree of trust. Controlling who can search using a vocabulary provides the service provider with a degree of trust in the client.
- The vocabulary should carry information about the creator of the vocabulary in order for the user to have sufficient trust in the matching rules it contains.
- The creator of the vocabulary must be able to define the matching rules, and these matching rules should be definable per attribute, not just per value type.

This last point needs some explanation. Consider a string-valued attribute. What constitutes a match on a "less than" query? Is it collating sequence? In what language? Is it substring? Case sensitive? Starting at the beginning of the string? The creator of the vocabulary has the semantic knowledge to understand the meaning of the attributes and answer these questions, and the answers may be different for different attributes.

Once such a general vocabulary mechanism is in place, it can be used for other purposes. Events can be defined in terms of a vocabulary. Publishers and subscribers can find each other by advertising in the event's vocabulary. Event state can be specified as attribute values in the vocabulary and subscription filters can specified as constraint expressions in the vocabulary. A vocabulary can also form the basis of online negotiation and contract formation. Each multi-valued attribute can be treated as a clause in a contract, and negotiation can proceed to settle on values in the contract.

4. References

This section provides links to the pages of the technologies referenced in this document.

1. Alta Vista: http://altavista.com
2. CORBA: http://www.corba.org
3. ebXML: http://ebxml.org
4. E-speak: http://e-speak.net; http://e-speak.hp.com
5. Jini: http://www.sun.com/jini
6. LDAP: http://www.openldap.org
7. UDDI: http://uddi.org
8. VerticalNet: http://www.verticalnet.com
9. Yahoo!: http://www.yahoo.com

Requirements for Automated Negotiation

Claudio Bartolini[1], Chris Preist[1], Harumi Kuno[2]

Hewlett-Packard Labs

(1) Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK

(2) 1501 Page Mill Road, Palo Alto, CA 94304, USA

Email: claudio_bartolini@hp.com, chris_preist@hp.com, harumi_kuno@hp.com

1. Introduction

The increasing importance of business to business electronic trading has driven interest in automated negotiation to soaring heights. In recent times, web-service enabled electronic marketplaces have been displacing proprietary trading solutions. Looking at this trend, we foresee a need for a general software infrastructure that enables independent entities to interact using multiple forms of negotiation. This infrastructure would cover a variety of aspects, including defining a general protocol for negotiation (including the definition of the actors, roles and phases of negotiation); defining a taxonomy and a language for negotiation rules to cast the general protocol into one that embodies the desired market mechanism; defining a language to express negotiation proposals.

Negotiation has been for decades a central subject of study in disciplines such as economy, game theory, and management. When discussing negotiation, it is important to distinguish between *negotiation protocol* and *negotiation strategy*. The protocol determines the flow of messages between the negotiating parties, dictating who can say what, when and acts as the rules by which the negotiating parties must abide by if they are to interact. The protocol is necessarily public and open. The strategy, on the other hand, is the way in which a given party acts within those rules in an effort to get the best outcome of the negotiation for example, when and what to concede, and when to hold firm. The strategy of each participant is therefore necessarily private. In this document we concentrate on the requirements for architecting software enabling automated negotiation, therefore we discuss protocols and not strategy.

Existing approaches to architecting software enabling automated negotiation provide either ad-hoc software for particular market mechanisms or proprietary solutions. We take an open approach by defining a standard protocol for interaction among the participants in the negotiation process. Our protocol is defined independently from the negotiation rules embodying the particular market mechanism that the negotiation host wants to impose. Instances of negotiation rules will be used to cast the general protocol into a specific one that embodies the desired market mechanism. This approach is general with respect to a wide variety of market mechanisms, from one-to-one negotiation to auctions and double auctions.

2. Value proposition

Our position is that we need to design a standardized infrastructure that allows two or more independent entities to interact with each other over time to reach agreement on the parameters of a contract. This infrastructure is aimed primarily, though not exclusively, as a means to reach trade agreements. It can be used both by automated entities and by users via appropriate software tools.

The value of such a framework to *negotiation participants* is threefold. First, the framework frees participants from having to develop their own negotiation infrastructure, providing prerequisite services such as the provision of decision support and support for the automation of the negotiation process. Second, the infrastructure enforces the standardization of basic interaction rules, allowing participants to be confident that basic rules of interaction in any negotiation will be followed. For example, participants will able to negotiate simple contracts, where only price is undetermined, as well as more complex contracts involving multiple complex and interdependent parameters. Third, the protocols provide the participants with trust guarantees, ensuring that no party has access to extra information or is able to forge false information.

The value to *negotiation hosts*, such as auction houses and market makers, is that by providing a standard framework that is independent of any specific market mechanism, they will increase the number of potential customers who can interact with them. The infrastructure would allow the hosts to select an appropriate market mechanism. It would provide standard off-the-shelf market mechanisms (e.g. English auction), and also allow custom mechanisms to be implemented for particular special needs (e.g. the FCC auction).

3. Requirements

We identify the following requirements for a negotiation protocol that would meet our goals:

- Be sufficiently formal that automated entities can interact using it.
- Support negotiation about simple and complex objects.
- Be sufficiently general that a variety of different market mechanisms (e.g. 1-1 negotiation, combinatorial auctions, exchanges) can be expressed as specific instances of it.

- Support security mechanisms and protocols that enable participants to do business in a trusted way.
- Allow, but not require, the existence of a third party to arbitrate a given negotiation (e.g. an auctioneer in an auction.)
- Support existing ways of doing business, as well as permitting more radical approaches in the future.

4. What needs to be defined

4.1. A general protocol for negotiation that can support a wide variety of market mechanisms

A negotiation protocol provides a means of standardizing the communication between participants in the negotiation process by defining how the actors can interact with each other. We therefore base the definition of our protocol on conversation orchestration protocols such as CDL [1] or WSDL [2].

We propose that the protocol should be general enough to support a wide variety of market mechanisms. Therefore the protocol should be based on the common aspects of the various market mechanisms that it wants to support. That is, define the negotiation process as an exchange of negotiation proposals followed by a phase of agreement formation. The two phases will often be intertwined for some market mechanisms.

Designing the protocol requires the definition of:

1. The roles played by the actors involved in negotiation processes
2. The phases of the negotiation process (e.g. admission, proposal submission, agreement formation)

4.2. A taxonomy of rules of negotiation

As noted above, the protocol is defined independently from the negotiation rules embodying the particular market mechanism supported by the negotiation host. The rules for negotiation will then cast the general protocol into one that can be used to embody a particular market mechanism.

Examples of types of rules for negotiation would be rules for deciding on the well-formedness of a negotiation proposal. Another example is rules regulating the alternation of participants in submitting proposals. Again, there will be rules the dictating the visibility aspect of proposals in many-to-many negotiation, i.e. who among the participants is entitled to see a submitted negotiation proposal, and so on. Rules will be needed for supporting mechanisms such as zero knowledge proofs to avoid revealing private information in the course of negotiation.

4.3. A language to define rules of negotiation

The language to define the rules of negotiation should be standardized. The idea is to have a declarative language for expressing rules in a way that participants to negotiation can reason about them. The declarative layer would then be mapped to reusable software components implementing the logic expressed by the rules. These components would be plugged in the orchestration infrastructure for the protocol to be cast to embody a desired market mechanism.

4.4. A language to express negotiation proposals

The format to express negotiation proposals has to be standardized. The requirements to be satisfied by a candidate language would be:

- Support for ontology and namespaces
- High degree of expressiveness
- Ability of expressing less than fully bound specifications
- Ability of expressing constraints over ranges of possible values as well as definite values of a specification
- Loosely supporting types and some degree of inheritance
- Support for complex queries
- Support for complex matching

RDF [3] and DAML-OIL [4] are promising candidate languages

5. References

[1] Govindarajan K., et al. *Conversation Definitions: A way of defining interfaces for web services* – submitted to W3C Workshop on Web services, 11-12 April 2001, San Jose, CA

[2] http://msdn.microsoft.com/xml/general/wsdl.asp

[3] http://www.w3.org/RDF

[4] http://www.daml.org/2000/12/daml+oil-index.html

Towards the Electronic Contract

Michal Morciniec, Claudio Bartolini, Brian Monahan, Mathias Sallé

Hewlett-Packard Labs

Filton Road

Stoke Gifford

Bristol BS34 8QZ

UK

Email: michal_morciniec@hp.com, claudio_bartolini@hp.com, brian_monanhan@hp.com, mathias_salle@hp.com

1. Introduction

In recent years we have witnessed an explosion of business applications exploiting Internet as a communication medium. Initially, on-line catalogues and shop fronts have been deployed followed by auction sites and finally business-to-business (B2B) infrastructures [Sculley 1999]. Electronic marketplaces are e-commerce infrastructures that aggregate potentially large numbers of buyers and sellers and allow them to interact according to a variety of market mechanisms such as request for quotes, reverse auction or exchange resulting often in significant cost saving. They rapidly evolve towards trading of web services rather than commodity goods.

As each enterprise tries to maximize its goals, conflicts of interests are certain to appear. Possible concerns [Favier 2000] range from security of transactions, fairness of the market mechanism, anonymity, identity of business partners to service performance. We feel confident that adequate solutions can be provided for most of the technology related concerns, however, a technology gap exists in addressing concerns that have their roots in the societal aspects of business interactions.

In the real-world the contracting process together with the established institutions and adequate enforcement and trust mechanisms provide an answer to these concerns. We assert that reification of this process will be required in order to enable enterprises establish and manage dynamically changing business relationships that can be formed in the electronic marketplaces.

2. *Electronic contract*

A contract is a statement of intent that regulates behaviour among organizations and individuals. An electronic contract is its reification in software that can be instantiated as a set of obligations that are fulfilled between parties, refused or waived as future events occur. Because the contract parties are assumed to be autonomous and self-interested, conflicts will occur, and an appropriate resolution mechanism is required.

An electronic contract can be viewed through transformations that are applied to it during its lifecycle in the electronic marketplace. Further, there is an information viewpoint that shows (document) objects that can be observed at the beginning and termination of each stage. Conceptually, the lifecycle can be split into the three stages of contract drafting, formation, and execution.

Contract drafting phase – Given the contract template model, the drafter role constructs an instance of the template. In this phase the contractual roles, abstract business interactions and contractual situations are specified. Furthermore, if the drafter acts as a regulator, rules and constraints can be added which should be adhered to during the contract execution phase. The template typically has a number of free variables that are agreed upon in the next phase.

Contract formation phase - Participants assume contract roles and negotiate the details of their responsibilities. The negotiable variables of the contract (deadlines, order of actions) become fixed, and concrete business interactions are bound to the abstract ones defined in the template. The relationships between contract parties are created and are captured in the contract statements using the policy expressions that imply obligations and rights of parties.

Contract execution phase –Actual delivery of contract consideration happens. Typically this phase constitutes service or goods delivery, invoicing, bill calculation, presentment and payment. The interactions between the parties may be monitored for their conformance to the terms of the contract.

3. *Scope of the proposed work*

Standardization effort is required to enable enterprises to interact with each other and electronic marketplaces. The scope of the effort is broadly related to the provision of a contracting framework in the context of electronic marketplaces. More specifically it includes the following:

1. Definition of the structural model for electronic contract (means to capture contractual commitments) that includes relevant information objects related to the contract during its lifecycle

2. Definition of the main phases and roles involved in the contract lifecycle and transformations applied to it. In particular,

    - Contract formation (contract drafter, contract negotiator);
    - Contract fulfillment (service provider, service consumer);

3. Specification of the protocols used by the relevant roles to achieve these transformations. In particular,

    - Protocol used for contract formation;
    - Protocol used for contract fulfillment;

4. Interfacing of the contracting framework components with other components in the enterprise.

4. References

[Sculley 1999] Sculley A.B., William W., Woods A., 1999, "B2B Exchanges : The Killer Application in the Business-to-Business Internet Revolution", ISI Publication.

[Favier 2000] Favier J., 2000, "eMarketplaces Face the Law", the Forrester Report, October 2000, http://www.forrester.com

Security Requirements for Web-Services

Hewlett Packard Position Paper to the Worldwide Web Consortium Workshop on Web Services, April 11th and 12th.

Author: Mike Jerbic, Hewlett Packard Co. email: Mike_Jerbic@hp.com

Abstract:

As XML document exchange models become the de facto standard for users and programs to interoperate with web services across multiple security boundaries, end to end security becomes either meaningless or at best extremely difficult. Today's XML security "solutions" center around document encryption and signature: that is securing the confidentiality and integrity of the XML document itself. Missing from the solution suite is an interoperable standard to authorize access to services and a framework for accountability should one or more components of a service invocation fail to perform. The W3C should initiate an activity to create standards for defining, discovering, and exchanging authorization and accountability information within XML document conversations.

Problem Statement

HP has taken a position that XML document oriented conversations are appropriate and necessary to exchange information needed to discover, exchange necessary information, invoke, and receive electronic services over the Web. For XML conversations to become practical for the exchange of personal information, service invocation authorization, and service participant accountability, security must be designed into the conversation framework from the beginning.

Security services in use today do not satisfactorily address the needs of an open, web-oriented document exchange model for electronic service delivery. They are deficient today in these ways:

- Security services are single domain oriented. E-services by design cross multiple domains. Examples include service composition across enterprises and the overall Business to Business electronic commerce problem. Even though PKIs (such as Verisign) appear to span security domains, they by virtue of being a "trusted third party," simply establish their own domain. Service providers and consumers must be able to establish a security context independent of so called trusted third parties.
- Security systems require unnecessary information, the exposure of which is

a global privacy, public safety, and security issue. Many times names, contact information, and other non-essential personal information is required to support legacy identity oriented security services. Adversaries misuse this information routinely. A global e-services infrastructure must allow for communication of privacy policy, the sharing of sensitive information, and recourse in the event privacy agreements are not kept. Some specific industries that are facing critical privacy problems yet also have the business requirement to exchange private information over the web in the conduct of their e-services are:

- United States Health Care delivery (HIPPA regulation requirements).
- United States multinational enterprises who must comply with the European Union Privacy Directive for the protection and use of European citizen information
- Banking and Financial Services transactions
- B2B and B2C service invocation ant payment
- And many others…

Privacy and Security expectations must be part of the agreement [XML conversation] between service providers and service consumers. They cannot be ambiguous. XML conversations should have the flexibility of being private conversations.

- Security systems do not scale well and may actually prevent service providers from entering markets. Security systems often do not scale in
- Cost. Cost can be measured in several ways including legacy system integration or porting costs and the cost of ownership of central security servers or services.
- Time. Security servers that map names to authorizations, for example, may require several hours up to days to vet new entries or to make permission or role changes granting authorizations to new services. Open, web based dynamic service providers require much faster, easier to change authorization systems. Real time manageability of security databases is a significant impediment to the dynamic advertising, discovery, and invocation of e-services.

Solution Requirements

Overall HP’s position is that XML documents must be able to include security context information that is discoverable, invocable, and interoperable. The solution needs to include:

- *Authorization.* Computers do not need identities to perform requested services. Computers need authorizations. Authorizations should be embeddable or attachable to XML service requests. Authorizations range from simple payment (credit card number, digital cash...) to a certified, tamper resistant authorization message originating from an authorization authority the service recognizes.

- *Accountability.* While computers only need authorizations, people need methods of determining accountability. Usually this is achieved through an access control list, which maps authorizations to an identity and an audit trail of activity. Other forms of accountability information are possible and should be explored.

- *Manageability.* Multiple methods of authorization management must be supported. Three primary ones are:

  Mapping names (identities) to authorizations. This is one of the most common forms of managing Service access to named individuals. Also knows as an Access Control List, mapping names to authorizations works well in small communities where identities are commonly known and used. This method does not scale well for large communities where name collisions can occur. An example is below:

Authorization Access Control list example for three identities.

Name	Service 1	Service 2	Service 3	Service 4 …	Service n
Mike	Yes	Yes	No	Yes	No
Terry	No	No	Yes	No	No
Eliot	Yes	Yes	Yes	Yes	Yes

Mapping names to roles, mapping roles to authorizations. An extension of the access control list is the creation of roles and privileges. Individual identities are mapped to roles, which in turn are mapped to specific authorizations. This scheme is somewhat more manageable for services that have a large client base (large number of names) or a distributed client base, where individual authorization is delegated. An example is below.

Authorization Access Control list example for three roles.

Role	Service	Service 2 Add User	Service 3 Bill User	Service 4 Credit	Service n
Subscriber	Yes	No	No	No	No
Admin	No	Yes	No	No	No
Back Office	No	No	Yes	Yes	No

Identity to Role Map

Identity	Subscriber	Admin	Back Office
Ted	Yes	No	No
Mike	No	Yes	No
Terry	No	No	Yes

Direct Authorization from the service provider. There are times when a determining authorization does not require checking a map of identity to authorization, as in the two examples described above. A service can simply grant access to an entity. This approach is best documented in the Simple Public Key Infrastructure (SPKI). In this system a service requestor is given an authorization to use the service by the service. When a service is invoked, the requestor presents a tamper resistant message authorizing the access to the service provider. No external authorization database check is required, since the authorization verifier has all the information required in the service invocation request. Direct authorizations may be delegable.

- *End to End Security.* Service invocations and responses should be able to be confidential (encrypted) between the requestor and the service provider, without relying upon any intermediary third party such as a web server, portal, etc. Applications do exist where intermediaries may need to examine the traffic flow between service provider and consumer, and the security architecture should be able to support this.

- *Support of existing security mechanisms, protection of IT investment already in place*. Security contexts need to have the flexibility of using legacy security infrastructure components such as X.509 identity certificates, S2ML, SSL, and integration with web browser security.

- *Discoverability*. XML conversations need to support the exchange of security context information, and the requirements to access a service must be discoverable by the service requestor. A conversation infrastructure must be developed that includes security.

- *Scalability*. The solution must support the service issuing its own authorization to a client without the need of an additional trusted party (such as a CA). Certificate Authority or Authorization Authority oriented solutions such as S2ML, SSL should be supportable as well for those service providers who want/need to leverage them.

- *Privacy*. A method for interoperable exchange of privacy intentions is strongly desired. (Leverage / extend P3P?)

- *Tamper Resistance*. The security architecture must provide for tamper resistance (most likely through digital signatures, XML DSIG?).

- *Non-Repudiation*. Tamper resistance, which is technically feasible, must not be confused with non-repudiation, which is not. Digital signature technology can present some evidence of the signer's intent, but non repudiation and the legal trustworthyness of exchanged documents must be governed not only by available technology, but also by law and contractual agreements.

Relationship to Other Standards

To address the requirements above, many already proven security standards must be considered and supported in the proposed new standard where they add value. While not intended to be an exhaustive list of standards to consider, the following serves as a starting point. It is important to note that the W3C already has experience in developing and recommending security solutions, and several of the standards have origins in W3C activities.

SSL secures a web browser to a web server. In the case of a portal of many services, confidentiality offered by SSL ends, with the client relying on the web server to end service security outside of the client's control. SSL is not end to end in a distributed service world.

S2ML provides an authorization authority model that must be queried every time an authorization is required. S2ML is a standard in process at the OASIS standards organization.

XML Encryption provides confidentiality of an XML document, but it does not provide authorization information in a standard, interoperable way.

X.509 PKI – OK if the service provider can use name to authorization maps.

SPKI – OK if the service provider can't use name to authorization maps.

Session Layer Security (SLS) – While not strictly a standard, HP has invested uniquely in a distributed authorization system to secure end to end Java applications. The security mechanism of HP's E-Speak, SLS could be extended to XML. HP awaits the opportunity to work with the W3C to propose and extend this technology.

P3P – A description of a site's privacy policy that can be downloaded and automatically compared to the user's preferences.

XML DSIG – Provides digital signatures to XML documents. This functionality should be considered for inclusion as part of the XML security standard to support tamper resistance.

A Framework for Business Composition

Andy Seaborne, Eric Stammers, Fabio Casati, Giacomo Piccinelli, Ming-Chien Shan

Today, the Internet is not only being used to provide information and perform e-commerce transactions, but also as the platform through which services are delivered to businesses and customers. More and more companies are rushing to provide all sorts of services on the Web, ranging from more "traditional" on-line travel reservations and directory services to real-time traffic reports and even outsourcing of entire business functions of an organization, such as IT or human resources departments.

Business composition is the ability for one business to compose e-services (possibly offered by different companies) to provide value-added services to its customers. Composition of e-services has many similarities with business process (workflow) automation. An e-service virtualises the customer interaction aspects of the business processes implementing a service. In order to specify how services should be composed, we must define the interaction process associated with a service as well as the flow of service and inter-service invocations. In this paper we refer to a business process that composes e-services as *e-process*.

We first outline the trend to business composition and e-processes, and then set out some requirements on standards to be developed to support e-processes.

E-Processes and E-Services

E-processes are typically designed, developed, and deployed by enterprises that want to compose internal capabilities with third-party capabilities, either for internal use or to expose them as (complex, value-added) e-services to customers. Note that, while the e-services involved in an e-process may belong to several companies, the e-process is company-specific. Its definition is typically known to and controlled by a single company. For both e-processes and e-services, we do not envision the need for companies to expose internal details of how they run their business. Still, companies should be able to express in a standard format the interaction aspects for their service offer as well as for their service needs. A common language is crucial for the assessment of the compatibility between the interaction processes offered by an e-service provider and those expected by the designer of an e-process.

The effectiveness and efficiency of business processes impact directly the profitability of a company. It is in the best interest of e-service providers as well as e-service consumers to understand the operational requirements for their cooperation. An e-process defines the interactions between the company owning the process and the e-service providers involved in its implementation. In particular, an e-process defines the orchestration activity needed to enable the cooperation between the e-service providers. As traditional processes were designed around the operational model of customised business applications, e-processes should be designed around e-services. A clear understanding of the business interaction model of an e-service is paramount.

The Evolution of e-Processes

Phase 1 : integration of existing internal assets

Today, enterprises are automating their processes to reduce costs and improve process execution quality and speed. Process automation and management technologies enable the separation between business logic, resource logic, and application logic. Process can be controlled, managed, and evolved separately from the applications. Still, in this phase, resources are internal to the enterprise.

Phase 2 : static integration with partner processes on a case-by-case basis

By incorporating e-services provided by business partners into an e-process, an enterprise can create processes that utilize external resources. However, in this phase, service selection and invocation is still performed in an ad-hoc way, and require preliminary agreements (from a business, legal, and technical perspective) between the cooperating companies.

Phase 3 : dynamic integration with negotiation, with companies

Beyond such static use of external services, fully dynamic e-processes make decisions each time they are executed in order to invoke the best available service that can fulfill the customer's needs. The tradition design-deploy cycle of phases 1 and 2 has changed to a per-instance set of decisions.

In the remainder of the paper we focus on dynamic e-processes, since these are the ones that can provide the best value and are in line with the philosophy of the e-service environment. In particular, which aspects of e-services should be standardized in order for users to define e-processes that compose e-services and execute them in a fully dynamic and efficient way.

Enabling Standards

We now focus on dynamic e-processes, since these are the ones that can provide the best value and are in line with the philosophy of the e-service environment. In particular, which aspects of e-services should be standardized in order for users to define e-processes that compose e-services and execute them in a fully dynamic and efficient way. Some of these standards are under development by industry consortia or standardization bodies, while others still need to be developed.

Service Metadata

In order to stay competitive, service providers should offer the best available service in every given moment to every specific customer. Similarly, e-processes should be able to compose the best services available when the process is executed. Hence, when defining an e-process, we need to be able to describe the type of service we want, and let the system discover which specific e-service can satisfy the e-process needs. In order to be able to select e-services and understand their characteristics, a standard for service description is needed. This standard should allow the definition of the e-service properties, so that the e-process can determine its suitability for satisfying the business needs.

We also note that we do not envision the definition of a common, global ontology that can fit every e-service description. In fact, while businesses will certainly want to use standard ontologies when available, they will often need to define attributes faster than allowed by standardization bodies. Hence, the standardization effort here should allow the definition as well as the discovery of ontologies.

Service Interface

The description of an e-service should include the specification of the e-service interface. In order to support e-processes, a standard way of defining this interface is required, so that the e-process execution engine can determine which operations are available on the selected e-services and how to invoke them. It is also desirable to have a description of the semantics of each operation, to understand, for instance, whether the invocation of an operation commits the invoker to a payment or to send a registration form.

Note that services are typically complex entities, offering several methods (operations) to be invoked as part of their interface. For instance, an e-music service may allow users to browse or search the catalog, to listen to songs, or to buy songs. The service provider may want to impose constraints on the order in which these operations are invoked. Hence, a simple, IDL-like definition of the interface may not be sufficient to specify how the interaction with a dynamically discovered service should be conducted. We refer to the set of interactions (i.e., invocation of operations) with an e-service as *conversation*. In order to be able to correctly interact with a selected service, a standard description of the supported conversations is also required.

Transactional Processes and Services

A desirable feature in workflows as well as in service composition is the ability to execute parts of the process in an atomic fashion, i.e., so that all of it is executed or the effect of partial executions can be (semantically) "undone". A typical example is the reservation of a trip, where the customer would like to "atomically" book the flight and the hotel. If, for instance, the flight could be booked but no hotel room is available, then the flight reservation needs to be "cancelled" or "undone". A standard description of the transactional properties of an e-services would give the e-process engine the ability to perform such atomic executions and to identify how to interact with services when performing "commits" and "rollbacks".

Negotiation

Advanced service composition systems may provide facilities for negotiating with the discovered e-services before accessing them, and for reverting to other services if the negotiation fails. A standard for negotiation is required so that service composition tools may have embedded negotiation capabilities. In this way, the benefits of negotiations could be made easily available to e-process designers and users.

Security

Many services may require security credentials before allowing interactions. A standard need to be defined so that e-process definition languages allow the definition of the appropriate security certificates to be used, and e-process engines know how to authenticate and get authorizations for service executions.

Contract

A contract forms the basis of an e-process between businesses. The contract identifies the roles and responsibilities of the parties, the obligations and deliverables. With dynamic integration, it become important to explicitly represent services. In earlier phases, when there was an explicit integration phase, the issues in a contract would have been dealt with. With contract choices being made during the execution of an e-process, the agreements entered into will need to be recorded.

Service Discovery

The previous subsections have discussed mechanisms to enable interaction with an e-service once it has been discovered. However, mechanisms to discover e-services are required, so that the e-process can dynamically look for e-services during each execution. Hence, standards for identifying and querying service directories in order to retrieve information on e-services and their properties are needed.

Non-Requirement

In the previous section we have stated some requirements for dynamic e-processes. In this section we want to state a non-requirement because we believe it to be unnecessary.

There is no need to provide an explicit representation of the business network dynamically generated to deliver the value of a service. In other words, the services and processes used by one service are not exposed to the service clients. The dynamic nature of e-services means that the exact choices of which other services to use may not have been made at the time a contract is entered into.

The service client neither knows nor is concerned about the internal details about how a service meets its obligations. Each service has a partial view of the business network, no single point having a global view of the entire network.

The service provider is free to search and invoke other services that best meets its needs. Companies do not expose their internal operations or be constrained to do business as prescribed by some "global" process definition. Instead, companies want to expose services, but be free to implement such services the way they want.

Transactional Conversations

Svend Frolund and Kannan Govindarajan
Hewlett-Packard Company
email: svend_frolund@hp.com, kannan_govindarajan@hp.com

1 Introduction

We believe that web services should make their transactional properties available to other web services in an explicit and standard way. Transactional properties should be part of a service's interface rather than a hidden aspect of its backend. The transactional behavior of a service can then be exploited by other services to simplify their error-handling logic and to make entire business-to-business interactions transactional. However, such business-to-business transactions are challenging to implement because they span multiple companies and because the underlying transaction protocols execute over wide-area networks.

It is fundamental for web services to communicate through conversations. A conversation is a potentially long-running sequence of interactions (document exchanges) between multiple web services. For example, a manufacturer may engage in a conversation with a supplier and a conversation with a shipper to carry out the activity of purchasing parts. In many situations, the backend logic triggered as part of these conversations may be transactional. For example, it may be possible to arrange for parts to be shipped, and then later cancel the shipment (if the parts have not actually been sent yet). Cancelling the shipment is an example of a compensating transaction, it compensates for the initial transaction that arranged the shipment. Since the notion of conversation is fundamental to web services, the exportation of transactional properties should fit with conversations, giving rise to transactional conversations.

In the Internet setting, atomicity is the most important aspect of transactions. Atomicity means that the effect of a transaction either happens entirely or not at all. We also refer to this as all-or-nothing semantics. If a service A knows that a conversation with another service B is atomic, then A can cancel the conversation and know that B will cleanly revert back to a consistent state. Furthermore, A can rely on the B's transactional behavior to ensure state consistency in the presence of failures, such as logical error conditions (e.g., shipping is impossible) or system-level failures (e.g., the crash of a process or machine).

Services should expose their transactional behavior in a manner that facilitates composition of transactions from different services. For example, it should be possible for the manufacturer to compose a transactional conversation with the supplier and a transactional conversation with a shipper into a transactional activity that includes both conversations. The advantage of creating these multi-conversation transactions is that the manufacturer gets all-or-nothing semantics for the entire end-to-end purchasing activity: if shipping is not available, the order placement is cancelled. This is a very powerful notion, that we believe will significantly reduce the complexity of programming business-to-business activities between multiple web services. Composite transactions provide a notion of "clean abort" for entire business-to-business activities. Moreover, having a standard notion of transaction allows us to build generic software components that perform the transaction composition.

2 Atomicity

We discuss different ways for transactions to be atomic. As terminology, we introduce the notion of a transaction outcome, which is either commit or abort. The outcome is abort if the effect of the transaction is "nothing." The outcome is commit if the effect is "all."

2.1 Two-Phase Commit and Compensation

If we execute two atomic transactions, their combined effect is not necessarily atomic: one transaction may abort and the other may commit, which means that the combined effect is neither all nor nothing. If we create a composite transaction from two constituent transactions, we need to ensure that either both constituent transactions commit or that both constituent transactions abort. There are two traditional ways to ensure this. One way, called two-phase commit, is based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. Another way, called compensation, is based on the idea that a constituent transaction is always allowed to commit, but its effect can be cancelled after it has committed.

With two-phase commit, transactions provide a control interface that allows a transaction coordinator to ensure atomicity. One incarnation of the control interface is the XA specification [3]. Essentially, the control interface provides a prepare method, an abort method, and a commit method. The coordinator calls the prepare method to determine if a transaction is able to commit. If the transaction answers "yes," then the transaction must be able to commit even if failures occur. That is, the transaction is not allowed to later change its mind. If all transactions answer "yes," the coordinator calls their commit method, otherwise the coordinator calls their abort method.

With compensation, there is no additional control interface. Instead each "real" transaction has an associated compensating transaction. With compensation, a coordinator can ensure atomicity for a number of constituent transactions by executing the transactions, and if any transaction aborts, the coordinator executes the compensating transaction for all the transactions that have committed.

2.2 Discussion

Although both two-phase commit and compensation can provide atomicity for composite transactions there are trade-offs between the two methods. Compensation is optimistic in the sense that the coordinator only enters the picture if something bad (e.g, abort) happens. With two-phase commit, the coordinator enters the picture even if all transactions commit. The coordinator always calls prepare and either commit or abort for any transaction. On the other hand, two-phase commit always provides a point after which a service can forget about a transaction. Once the transaction is instructed to commit, the service can forget about the transaction. With compensation the service has to be able to compensate forever. The ability to compensate may or may not require the service to maintain persistent state. Of course, there are hybrid models where compensation is bounded by time or the occurrence of events (such as receiving a notification).

In practice, few systems use two-phase commit in the Internet context. One reason is that, with two-phase commit, a service exposes transaction control to other services. If a service answers “yes” in response to a prepare request, the service has to be able to commit the transaction until instructed otherwise by the coordinator (which may be another service). Few services are willing to export such transaction control in a loosely-coupled system. Another reason for the limited use of two-phase commit is that composite transactions may be long running. If we want transactions to span entire business-to-business activities, we have to accept the possibility that transactions may run for a long time. With two-phase commit, a constituent transaction cannot commit until the composite transaction can commit. Thus, a fast service may be forced to wait for a slow service.

If we want to support two-phase commit, we need a protocol that allows flexible designation of the coordinator role. For example, a given web service may be willing to play the role of participant in certain situations, but may insist on playing the role of coordinator in other situations. If we have a conversation definition language, such as CDL [1], we can capture this distinction through different conversations. A service can export a coordinator version and a participant version of the same logical conversation.

We believe compensation is a fundamental notion of atomicity for web service, and in the remainder of this paper, we shall only consider compensation. This does not reflect a position against two-phase commit, but is merely to simplify the discussion.

3 Isolation, Durability, and Consistency

Traditional database transactions satisfy the ACID properties (atomicity, consistency, isolation, and durability) [2]. Unlike traditional database transactions, we do not believe that transactions, in the context of conversations, should necessarily provide isolation. Isolation is concerned with controlling the concurrent execution of transactions to provide the illusion that concurrent transactions execute one after the other in some (indeterminate) serial order. Isolation is unnecessarily strict for Internet transactions. This is evident from many Internet sites that provide transactions without isolation. For example, sites, such as Amazon.com, provide transactional semantics in the form of compensation (cancelling an order within a given time limit) and do not provide isolation. Besides being unnecessarily strict in many cases, isolation is also costly because transactions may be long running, and providing isolation for long-running transactions hampers the overall performance.

To continue the comparison with database transactions, we would expect the "primitive," constituent transactions to provide durability and consistency. Durability means that transactional updates are made persistent if the transaction completes successfully. Consistency means that a transaction takes the system from one consistent state to another. The durability and consistency of constituent transactions follows from the transactional properties of the backend logic in web services. We do not believe that "composite," multi-conversation transactions should provide any global notions of durability or consistency beyond what the constituent primitive transactions provide. In other words, we do not rely on any particular notion of durability or consistency when we compose primitive transactions into composite transactions.

4 Describing Transactional Properties

We outline briefly what it may take to describe, and thus export, transactional properties of web services. The starting point for our discussion is the assumption that services communicate through explicit conversations. If a service exports a description of its conversations—the conversations it is willing to engage in—the question is how the service can specify the transactional properties of those conversations. The specification makes explicit to other services how the service is transactional. The specification should communicate the following aspects of transactions:

- **Demarcation**. We need to describe which parts of a conversation are transactional. If we consider a conversation as a sequence of interactions, we need to identify the transactional sub-sequences of those interactions. At one extreme, the entire conversation may be transactional. But other possibilities may exist as well. For example, only a single interaction may be transactional, or an identified sub-sequence may be transactional. In general, a single conversation may have multiple transactional parts.

- **Outcome**. To exploit the transactional behavior of a service, we need to know the outcome (commit or abort) of its transactions. One way to communicate the outcome of a transactional conversation to other services is to associate a particular outcome with a particular point in the conversation. For example, a specific interaction may denote abort and another interaction may denote commit. If the conversation reaches an interaction that indicates abort, the parties of the conversation know that the outcome is abort. We need to describe which interactions indicate abort and which interactions indicate commit. Notice that we can also use document types instead of interactions to indicate the outcome of transactions.

- **Compensation**. We need to describe how transactions can be cancelled or compensated for. For example, a conversation may have a particular document that triggers compensation, or different documents may trigger compensation at different points in the conversation. To initiate compensation at a given point in a conversation, sending a compensation document must be a legal interaction at that point in the conversation, and we must be able to generate the appropriate compensation document. Notice that compensation may not be possible at any point during a transactional conversation, so we need to know both how and when compensation is possible.

If a service exports a description of its conversations in the form of an XML document, we can think of the description of the transactional properties as a companion document.

5 Requirements

To conclude, we outline basic requirements for web service transactions.

We want our notion of web service transaction to fit with conversations. Conversations provide the context for transactions: transactions take place within conversations, and we talk about transactional conversations. The integration of conversations and transactions have consequences for the transaction model. Because conversations can be long-running, so can transactions. The transaction protocols, such as two-phase commit and compensation, involve communication between web services. These communications should be first-class members of the conversations between web services. For example, if we have a conversation definition language to describe conversations, we should use that language to describe the transaction protocols as well.

We want to support compensation as part of the transaction model. With two-phase commit, transactional web services rely on an external entity, a transaction coordinator, to communicate the transaction outcome to them. Such reliance on external entities may not always be appropriate in loosely-coupled systems. Compensation does not introduce the same level of reliance on external entities. Our position is not against two-phase commit, but rather in favor of compensation: two-phase commit protocols may be appropriate in certain situations. If we have a transaction model that supports both two-phase commit and compensation, we have to address the issue of "mixed-mode" transactions—transactions whose constituent transactions are based partly on two-phase commit and partly on compensation.

In general, regardless of the choice of transaction model, we want to support a decentralized, peer-to-peer model for transactions. For example, we do not want to assume the existence of a centralized transaction coordinator. We do not want to prevent a centralized notion of coordinator, we simply do not want to rely on one. Notice that the notion of a transaction coordinator may be relevant for both two-phase commit and compensation. A central coordinator might make sense in conjunction with compensation. This coordinator would then gather the outcomes of the various constituent transactions and execute compensation transactions as necessary.

We need to address the issue of trust between the web services that participate in a transaction. Both two-phase commit and compensation assumes that the various parties are well-behaved (or trusted). For example, two-phase commit assumes that participants vote “honestly” and that they do as instructed (commit or abort). Furthermore, the notion of compensation also assumes that a participant actually executes a compensating action if instructed to do so. With two-phase commit, each participant also trusts the coordinator to be in control of the protocol—the protocol is inherently asymmetric because the coordinator knows the outcome before any of the participants. Since trust is a general issue for web services, we assume that some other mechanisms are put in place to deal with trust in a general sense. In terms of transactions, we need to integrate with those general mechanisms to handle trust. It is unlikely that we can treat trust as a completely orthogonal issue to transactions.

References

[1] K. Govindarajan, A. Karp, H. Kuno, D. Beringer, and A. Banerji, “Conversation Definitions: defining interfaces of web services,” submitted to the 2001 W3C workshop on web services.

[2] J. Gray and A. Reuter, “Transaction Processing: concepts and techniques,” Morgan Kaufman Publishers, 1993.

[3] Distributed Transaction Processing: The XA Specification, X/Open Snapshot, 1991.

A Peer-to-Peer Service Interface For Manageability

Vijay Machiraju, Akhil Sahai
E-Service Management Project
E-Services Software Research Department
HP Laboratories, 1501 Page Mill Road, Palo-Alto, CA 94034
{vijaym, asahai}@hpl.hp.com

1. Introduction

Various interfaces and protocols are being defined for seamless composition and inter-operation of web services. Some of these are used by web services to discover each other (e.g., UDDI [1]), some of these are used by services to express their functionality to each other (e.g., WSDL [2]), and some of these are used to invoke operations on each other (e.g., SOAP [3]). While all of these are essential for easy composition, there are still many aspects that services need to agree upon in order to interoperate. One prominent example of such an aspect is manageability.

There are two interpretations of manageability for services. *Manageability from a management system's perspective* refers to whether a service provides sufficient information (events, measurements, and state) and control points (lifecycle control, configuration control, etc) to a management system so that it can be effectively monitored and controlled. There have been many efforts – Common Information Model from Distributed Management Task Force [4], Manageability Service Broker from Open Group [5], and Java Management Extensions from SUN [6] to name a few – for standardizing the interface and protocol between managed services and management systems. So far, management systems and these standards have been focused on managing enterprise applications. As web services become more prevalent, cross-enterprise management of federated services will become increasingly important.

There is a second notion of manageability – *manageability from a peer service's perspective.* Figure 1 shows two peer-to-peer services interacting with each other. An interaction is any form of communication between two services. This could mean a single-step transaction (e.g., a login to a book-selling service), a sequence of related transactions (e.g., logging in, adding books to shopping cart, and checking out), or even business-level processes perceived by the client that may involve manual steps (e.g., the process from ordering books to final delivery of the books). For every step in an interaction, one of the two services initiates the request and the other executes the request. We refer to the service that initiates the request as the *consumer service* and the one that executes the request as the *provider service*. The role of a service could change over the course of an interaction.

From a consumer's perspective, *a provider service is manageable* if the latter provides sufficient visibility and control over itself and over the interactions it executes. For example, a provider that provides information about the progress of a consumer's ongoing interactions or an ability to escalate their speed is more manageable than a provider that does not. From a provider's perspective, *a consumer service is manageable* if it can provide enough information about its service usage back to the provider. For instance, a consumer that can be queried about its perception or quality of experience is more manageable than a consumer that cannot be. Manageability interfaces capture the functionality that should be offered by providers and consumers to each other in order to be manageable.

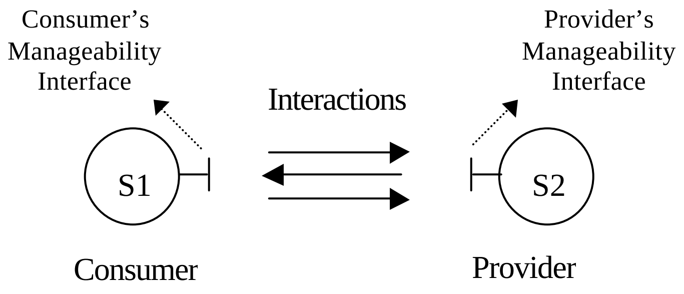
Consumer's Manageability Interface
Provider's Manageability Interface
Interactions
S1
S2
Consumer
Provider

Figure 1: Peer-to-peer services, their roles, and manageability interfaces

There have been no efforts to standardize interfaces and protocols of this nature between web services. In this paper, we first motivate the need for such interfaces by explaining their benefits. We then list a set of elements that should be part of these interfaces. Finally, we conclude with some suggestions for how to approach the standardization.

2. *Need for Peer-to-Peer Service Manageability*

We envision a world where services will be capable of advertising, discovering, composing, and using each other to execute their functionality. Providing interfaces for manageability between services would help in realizing part of this vision.

In the discovery-phase or pre-composition phase, a consumer can use manageability interfaces of providers to obtain valuable information about their quality of service - performance, availability, reliability, and service-level guarantees. Brokers and other discovery services help clients select services. However, the selection criteria are currently limited to price and functionality. Standardizing manageability interfaces will facilitate negotiation and selection based on a whole new set of features. With this additional information, consumer services will be able to assess if they can guarantee end-to-end quality to their own clients taking into account the quality levels guaranteed by their provider services. When services are composed, it is not only the functionality that gets composed; the service-levels, performance, and availability get composed as well. Hence it is only natural for services to expose these details so that a consumer can make informed decisions before composition.

In the execution phase or post-composition phase, a consumer that uses a manageable provider has better visibility and control over the latter; and hence will be able to make better decisions. For example, a consumer that needs faster service can escalate its interactions to the next tier of service temporarily. Similarly, it can request long-lasting transactions to be shutdown so that they can be executed on a different service. Consumers will be able to compare different services for their quality and performance.

Third party rating services can be formed that just collect and sell ratings of various services for their quality. A need for third party mediators would also arise for mediating and monitoring compliance of service level agreements between other services. A real life analogous example would be the credit rating companies and the manner in which they settle disputes. However, the dynamics in case of web services would be different and so the third party mediators have to monitor the compliance and take corresponding actions in real time.

Being manageable is advantageous to the provider too. Manageability is a new differentiator for service providers. Higher levels of visibility and control can be provided to valued customers or at higher prices. We have seen various examples of adhoc manageability offered by various services to be more successful. For example, Fedex offers tracking of their shipping transactions (example of visibility). Most of the on-line e-commerce sites provide some form of a cancellation feature so that a service or product can be cancelled after being purchased. Web services have already realized the importance of accountability and guaranteeing service levels. A standardized manageability interface helps in projecting all these features of a web service.

One of the major drawbacks of current instrumentation and management technologies is the lack of an end-to-end solution. Once a service crosses a service provider (into a client or a supplier), he or she has no visibility or control on what happens with the interactions. Management systems internal to a service provider find it difficult to analyze if a failure or service-level violation is attributed to internal business logic or to a supplier. Having manageability interfaces on all services helps in solving this problem and in facilitating cross-enterprise management systems.

3. Elements of Manageability

Manageability is measured in terms of two factors – *visibility* or ability to observe and monitor, and *controllability* or ability to influence and change. One way to classify the functionality offered through a manageability interface is to differentiate elements of visibility from elements of control. Another dimension of classification is *service-level* versus *interaction-level*. Service-level interfaces are used to obtain information about or execute actions on the overall service. Interaction-level interfaces are used to obtain information about or control specific interactions between services.

As we have already explained, a service could act as a provider, or as a consumer, or both. The type of manageability depends on the role of the service. Table 1 shows examples of functionality that could be offered by a provider to be manageable. Table 2 shows some elements of consumer-side manageability interface.

Table 1: Elements of a server-side manageability interface

Service-level Visibility	**Interaction-level Visibility**
Nature of visibility and control supported.	Expected time an interaction will take to execute before submission.
Quality levels – performance, availability, and reliability – that can be guaranteed by the provider.	Expected time an interaction will take while it is in progress. Track an interaction in progress.
Quality levels currently being guaranteed to the consumer.	Amount of time an interaction took to execute.
History of quality levels with the consumer.	History and statistics about past interactions.
History of quality levels with all the consumers.	Quality levels being guaranteed on certain types of interactions to the consumer.
Generic servic e metrics such as number of current consumers, average turn-around time etc.	
Service specific metrics – e.g., books sold per second for a book-selling service.	
Service-level Control	**Interaction-level Control**

Change current quality levels being guaranteed to the consumer. For e.g., change the consumer from bronze to gold customer.	Abort an interaction. Suspend an interaction. Resume a suspended interaction.
Report a consumer-perceived service-level violation to the provider. Ask for an explanation. May result in compensation according to the contract.	Change the desired quality level for an interaction.

Table 2: Elements of a consumer-side manageability interface

Service-level Visibility	**Service-level Control**
Nature of visibility and control supported.	Slow-down the request rate.
Quality levels perceived by the consumer.	Fail-over to a different service.
Quality levels currently being guaranteed to the consumer.	
Interaction-level Visibility	**Interaction-level Control**
Perceived quality levels for a particular interaction or a class of interactions.	Re-issue or restart an interaction. Suspend an interaction.

4. Realizing the Vision

Realizing service manageability would require involvement and standardization on two fronts:

1. Define a set of standard terms, conversation definitions, and communication messages to support manageability of web services. Services first need to agree upon terms such as quality of service, contracts, and transactions. The next step would be to define conversations that should be supported by services to be certified manageable. For example, conversations for negotiating quality of service and for canceling transactions should be standardized. These conversations should cover all the types of visibility and control that can be offered by web services in a generic manner. Technologies such as XML, WSDL, and current management standards such as CIM would be helpful in defining these vocabularies and conversations.

2. Standardize extensions to current communication protocols such as SOAP to include manageability information. Services could communicate valuable

information about interactions to each other by exploiting the messages that they exchange. For example a message from a service can include expected time of completion as part of its response. Headers that capture information such as quality level expected and conversation context can be standardized for every request. Similarly, quality level delivered, response time information, and conversation context can be part of every response header.

Coordinated efforts are also needed in the area of security for web services. A good foundation of security is a pre-requisite for services to be able to offer visibility and control over their functionality.

5. References

[1] Universal Description, Discovery, and Integration, http://www.uddi.org

[2] Web Services Description Language, http://www-106.ibm.com/developerworks/library/w-wsdl.html

[3] Simple Object Access Protocol, http://www.w3.org/TR/SOAP/

[4] Common Information Model, http://www.dmtf.org/spec/cims.html

[5] Manageability Services Broker, http://www.opengroup.org/management/msb.htm

[6] Java Management Extensions, http://java.sun.com/products/JavaManagement/