International Journal of Web Services Research

The Session Concept and Web Services

Hal Hildebrand, Oracle Corporation

Mark Little, Arjuna Technologies Limited

Anish Karmarkar, Oracle Corporation

Greg Pavlik, Oracle Corporation

Abstract

This paper describes the session concept as it relates to middleware systems in general and Web services in particular. Common applications of the session concept are found in distributed object systems, the Web, and messaging middleware systems. In the context of Web services, explicit building blocks for session-oriented protocols and services have been proposed in two specifications, WS-Addressing and WS-Context. The distinguishing characteristic of these two proposals is the degree of coupling they introduce between session participants. We compare the underlying models in these specifications, as it relates to the session concept in Web services, and conclude that WS-Context is a superior model for supporting a viable Web services architecture.

Session Models in Distributed Systems

Agents in distributed systems communicate by sending messages between software components. Middleware software provides plumbing and programming models for applications in order to:

1) Ease the burden of (re)programming network-level message passing on a per application basis.
2) Provide a conceptual framework to organize application logic.
3) Manage system protocols and their relationship to the execution environment.

As an example, distributed object systems like CORBA(Object Management Group, March 2004) or Java RMI (Sun Microsystems, Java Remote Method Invocation Specification, 2003) provide a programming model for networked services based on the idea of distribution transparency. When a client application works with a distributed object reference, the application itself has no knowledge as to whether the object is collocated in the same address space and available as a local object or distributed across the network. In the latter case, the use of the networked service requires the middleware system to respond to object invocations by marshalling the request to a protocol specific wire-level message and then to send the message over the network. Similarly, the networked service is supported by middleware that decodes the network message and in turn invokes a method on a local object implementation. These kinds of systems provide a simple programming model that is already familiar to developers who have used object-oriented programming languages. The effect is to make building distributed systems relatively simple, at least by comparison to hand-rolling custom message-over-sockets code on a per application basis. Support for system protocols like transactions is layered on top of the basic distributed object model.

One of the common features of all middleware systems is support for the session concept. For the purposes of this paper, a session is a mechanism for correlating multiple messages in order to achieve some application-visible semantic. This is typically done on behalf of a client within a service endpoint. In general, middleware systems decouple session association from specific communication channels to improve robustness. To achieve this, the session model is layered on top of a communication channel that links the client to network-visible application services. Many middleware systems advertise the session model explicitly as a mechanism for client applications to manage stateful conversations or communicate with stateful "resources". In other cases, the session concept is maintained less explicitly to support system services that are provided to applications.

In order to illustrate how the session concept is used in middleware systems, we review several examples.

CORBA Sessions

In CORBA systems, the Interoperable Object Reference (IOR) is the primary embodiment of the session concept. The IOR is an address-space distributable reference to an object. The IOR itself is a CDR encoded byte array that contains information of the following form (expressed in CORBA IDL):

```
module IIOP {
   struct Version {
      octet major;
      octet minor;
   };
   struct ProfileBody_1_ {
      Version iiop_version;
       string host;
       unsigned short port;
       sequence<octet> object_key;
```

```
};
struct ProfileBody_1_1 {
   Version iiop_version;
    String host;
    unsigned short port;
    sequence<octet> object_key;
    sequence <IOP::TaggedComponent> components
}
```

It's important to note that the IOR is encoded not only with the information required to establish a communication channel with a networked service, but also with an opaque token called an Object Key. The Object Key is the foundational session mechanism in CORBA. It is used to demultiplex requests to a specific processing component of the networked service endpoint in CORBA systems. In many cases, the IOR is used to associate requests with individual objects instances that may maintain conversational state on behalf of a client. The Object Key can be supplied by application logic within the service or auto-generated by the infrastructure, and is composed of an arbitrary sequence of bytes: in either case, the IOR originates with the network service.

The structure of the IOR itself is based explicitly on the design of the Object Request Broker infrastructure and reflects the intended implementation model for the system. While often used in support of distributed object systems, the IOR is an example of a *service reference session* model, where the network communication channel and the session identifier are combined into a single data structure that must be dereferenced by clients in combination to access a network service.

A typical application implementation pattern that exploits these features relies on a factory to generate IORs and clients to "garbage collect" resources when they are no longer needed; services may also implement autonomous cleanup in response to inactivity as well. These policies are developed on an ad hoc basis because session expiry is not a part of the basic

model. A representative example of this pattern is the stateful session Enterprise JavaBeans (EJB) model (DeMichiel, 2003), where references to stateful session bean instances are used by clients to maintain a stateful conversation with an EJB component. It's worth noting that while EJBs have proven to be highly popular as a mechanism for controlling transactions in J2EE based systems, the stateful session bean model has not been widely deployed. Instead, basic Cookie-based Web sessions have dominated J2EE deployments. Web sessions and stateful EJB sessions are difficult to combine because of their independent session models.

Both EJB and CORBA are best used for tightly coupled systems and assume intimate knowledge of the middleware infrastructure models. Note that in both cases, each objects or components have unique references that must be maintained by clients for continued use of the networked service. Each reference contains an identifier that is used to dispatch requests to implementation artifacts. This is the hallmark characteristic of systems that build on service reference sessions.

For this reason, the maintenance of distributed object references is typically complicated. Because sessions and endpoint information is intertwined, broken references must be reconstituted in order to reconnect to existing sessions. In fact, commercial CORBA systems never fully developed an interoperable mechanism that was adopted in practice to provide fault tolerant systems with recoverable session association.

Distributed object systems often support an additional session model, where information that influences the execution semantic of a service is communicated via context information derived from the client execution environment. Distributed transaction processing facilities like the CORBA Object Transaction Service rely on this model to maintain ACID properties across a number of service invocations that involve state changes to shared resources. The generalized

view of this execution context-derived session is often referred to as an activity model. The activity model is equivalent in computational expressiveness to the session model used in distributed object references. However, activity sessions have several distinguishing characteristics that make them different than reference sessions. Activity sessions are:

1) Dynamically bound to the communication channel.
2) Activity sessions may be initiated by clients, by third parties, or by services themselves.
3) Activity sessions incorporate a lifecycle model that assumes a temporal limit. Typically, APIs to support activity sessions assume a timeout exists and often allow clients to manage or specify timeouts.

HTTP Sessions

Distributed object systems are only one class of middleware frameworks. The ubiquitous HTTP protocol (Fielding et al, 1999) may be extended to provide sessions for managing application state on behalf of a user agent by the Cookie model (Kristol et al, 2000). HTTP itself is defined to be a "generic, stateless" protocol capable of acting as a carrier for many domains. This has been particularly useful for supporting information exchange in a massively scalable system like the Web. However, a purely "stateless" protocol has limited application. The Cookie model was introduced to allow Web-content applications to flexibly manage state. When a user accesses a Web site's origin server, the session is communicated back to the user agent via a Cookie, which contains information necessary to reestablish the session state on a per-request basis. The content of the Cookie is completely dependent on the origin server. In addition, the origin server is responsible for indicating lifecycle semantics for the cookie: for example, it may communicate a max-age value for the Cookie or explicitly timeout the cookie by supplying a

max-age value equal to zero on a response message to the user agent. Subsequent access to the Web resources with which the Cookie is linked require the user agent to piggyback the Cookie on the HTTP request headers based on domain name and path information associated with the HTTP request in order to maintain the session semantic.

When application server middleware is used to provide dynamic Web content/applications, middleware infrastructure usually provides built in HTTP Cookie based state management: application developers may store live data in a session cache and the infrastructure handles request correlation with the session data. In this case, the origin server typically ensures that the sessions are bounded by an inactivity time limit. This model is also familiar to developers that are accustomed to working with the Web programming models, for example, the Java Servlet APIs.

In either case, the HTTP session has no predefined structural relationship with the origin server's Web address or resources. The URL to a web resource represents the resource qua resource and is independent of the Cookie. The server always controls the session and its lifecycle. This model sharply reduces the coupling between user agents and web servers. For example, a Web browser need have no a priori knowledge of Amazon.com in order to purchase a book or CD. In general, the flexibility in the HTTP session model exploits the fact that user agents are most often acting on behalf of human users. Recovery from session invalidation is an ad hoc process. While HTTP sessions have been enormously successful, it is not clear that this model may transfer directly to domains concerned with automated machine-to-machine communications.

Sessions in Message Oriented Middleware

The session model is present in other kinds of middleware systems in less obvious ways. For example, at first blush it appears that message-oriented middleware (MOM) systems only use channels to relay messages to queues or consumers – that any correlation semantic to backend state must be encoded in the message itself by applications. However, MOM systems offer message-grouping facilities can be applied to ordering and delivery assurance semantics. For example, in many MOM systems, a session is created to demarcate and manage the start of an ordered group. To end delivery of ordered messages within a group, the session is closed. As an example, the Web services standard WS-Reliability (Iwasa, 2004) provides explicit protocol instructions to support this paradigm. Sessions are demarcated during application-level message exchange by implicit headers. Message acknowledgements may be communicated on independent communication channels, but messages are correlated with a group identifier that correlates messages with the session. While sessions in MOM systems are not necessarily explicitly accessed by message producers or consumers, the session concept is still very powerful and useful – including in systems that emphasize nominally decoupled message producers and consumers.

Web Services Sessions

So what about Web services? First, when we talk about Web services we are making the narrowing assumption that we are talking about services that are described in WSDL (Christensen et al, 2001) and accessed via SOAP (Box et al, SOAP 1.1, 2001). We can further make the following observations about the intended design paradigm for the Web services model:

1) Web services emphasize loosely coupled systems.

2) Web services do not expose artifacts of the implementation details or component model used to implement the service; Web services emphasize self-encapsulated messages and data transparency.
3) The WSDL definition for a service endpoint is agnostic about session models utilized by the service: specifically, the service element is exposed identically to all clients.

There has been some argument that Web services are in general "stateless". Except for reducibly trivial services (eg, algorithmically calculate some value based on the input data), this is in general false. The distinguishing hallmark of the Web service paradigm is that services minimize the projection of state into the protocol layers: they favor full encapsulation of the information required to correlate messages with resources and state information within the messages that are included in an Message Exchange Pattern.

In many, perhaps the majority, of today's use cases for web services technologies, no explicit protocol session model is required. As a general rule, application messages can maintain sufficient information for service operations to perform the work expressed in the WSDL contract. The emphasis in Web services is on self-contained messages. For example, a Purchase Order message would contain data including a customer id and the purchase order number that are necessary to process the order completely.

However, there are cases in which the session concept is necessary to interject into the execution environment; two examples include the reliable and ordered delivery of a set of messages and the requirement to scope the execution of multiple service operation invocations within a single unit of work. For our purposes, the mechanism for achieving a session in the Web services environment is of fundamental interest. Any use of the session concept increases the

coupling between the message producer and the message consumer, so how sessions are represented in the system will have serious implications for the Web services model itself.

In its own way, a standardized session model is as fundamental to the evolution of the Web Services architecture as the standardized adoption of the Cookie model was to the World Wide Web. When the Web was first developed, all server interactions were stateless; sessions between clients and servers are maintained only long enough to transfer an HTML page and are dropped immediately afterward. This means that costly resources, such as operating system network connections and threads, are not maintained for long durations. This per request session model is essential for supporting popular Web sites, where hundred or thousands of users may be interacting with the pages/services it provides.

However, if a site required a client to provide logon credentials before retrieving Web pages, then they would have to be provided on each visit to the site. Fairly obviously this "memory-less" interaction pattern did not scale beyond a few such sites and the cookie was added to the architecture. As we have noted, a cookie can be used to maintain session-like information between visits to a particular Web site. In fact, session oriented Web services often leverage HTTP Cookies to provide ad hoc session capabilities to support storing application state over multiple message exchanges by exploiting the fact that most Web services are accessed via HTTP today. However, Web services are intended to be accessed via many transport protocols, so this solution is not generally useable.

Currently, there are two proposed mechanisms for modeling the session concept in the Web services community, the EndpointReference in WS-Addressing (Box et al, WS-Addressing, 2004) and the activity concept in WS-Context (Little et al, 2004). As with most aspects of standardization, the value is derived from the potential for its features and functions to be

provided by Web Services vendors, therefore helping application developers solve state management and protocol requirements more easily. Once adopted and implemented, the functionality will not only be available as part of the platform but also it will be available in a standard way across platforms, allowing Web Services from multiple environments to interoperate more easily, efficiently, and effectively than if the developers had to code all of the equivalent features and functionality themselves in a non-standard way. The fact that both WS-Addressing and WS-Context are undergoing standardization is a net positive, but we maintain that the session model in WS-Context is a superior building block for the architecture of Web services. We next consider the session embodiment contained in both specifications.

Web service sessions using WS-Addressing

WS-Addressing is a specification that was jointly developed by Microsoft, IBM and others to provide a mechanism for referencing Web services developed on their platforms. In August 2004 the specification was published as a Member Submission to the W3C (a W3C Member Submission called WS-MessageDelivery (Karmarkar et al 2004) was also published in April 2004, offering similar functionality). A W3C Working Group was formed in October 2004 to standardize a Web services addressing specification and is using the WS-Addressing Member Submission as the starting point for its work.

WS-Addressing defines two extensible constructs: EndpointReference and Message Information Headers. Message Information Headers provide information on the characteristics of a message that are useful for directing messages to network endpoints and for correlating reply messages. EndpointReferences are used to identify and describe service endpoints and instances. For this paper we will discuss only EndpointReferences, as they are most relevant to the session

model in WS-Addressing. EndpointReferences logically extend the WSDL description model. An Endpoint reference contains the following information:

1. A address URI that identifies the endpoint
2. Primary WSDL 1.1 portType of the endpoint
3. WSDL 1.1 service QName/port that contains the definition of the endpoint
4. ReferenceProperties that identify the resource/entity addressed by the EndpointReference. ReferenceProperties are created by the issuer of the EndpointReference and are opaque to the consumer of the EndpointReference.
5. ReferenceParameters that are very similar to ReferenceParameter except that ReferenceParameters are not relevant when comparing two EndpointReference, but ReferenceProperties are. From the point of view of the session model ReferenceParameters do not alter the model and will not be discussed independently in this paper.

The relevant XML schema description for the WS-Addressing EndpointReference looks like:

```
        <xs:complexType name="EndpointReferenceType">
          <xs:sequence>
            <xs:element name="Address" type="wsa:AttributedURI"/>
            <xs:element name="ReferenceProperties"
type="wsa:ReferencePropertiesType" minOccurs="0"/>
            <xs:element name="ReferenceParameters"
type="wsa:ReferenceParametersType" minOccurs="0"/>
            <xs:element name="PortType" type="wsa:AttributedQName"
minOccurs="0"/>
            <xs:element name="ServiceName" type="wsa:ServiceNameType"
minOccurs="0"/>
              <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>
                  If "Policy" elements from namespace
"http://schemas.xmlsoap.org/ws/2002/12/policy#policy" are used, they must appear first
(before any extensibility elements).
                </xs:documentation>
              </xs:annotation>
            </xs:any>
          </xs:sequence>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:complexType>
```

In order to better understand the schema description, we provide an example EndpointReference for discussion:

```
      <wsa:EndpointReference

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
              xmlns:myns="http://example.com/ref-props-params">
          <wsa:Address>http://myurl.com/myservice</wsa:Address>
          <wsa:ReferenceProperties>
              <myns:SessionID>session$id:0123456789</myns:SessionID>
          </wsa:ReferenceProperties>
          <wsa:ReferenceParameters>
              <myns:CustomerID>Acme Inc.</myns:CustomerID>
          </wsa:ReferenceParameters>
          <wsa:ServiceName
PortName="myPort">myService</wsa:ServiceName>
      </wsa:EndpointReference>
```

In the example above, the endpoint address is "http://myurl.com/myservice/", has a ReferenceProperty containing the session identifier, a ReferenceParameter containing the customer identifier and the WSDL 1.1 service QName and port identifying the description of the endpoint.

The EndpointReference itself appears to be very similar to the IOR structure discussed earlier. The EndpointReference contains network information required to reach a service endpoint in the form of a URL. In addition (and like the IOR), the EndpointReference also contains a set of ReferenceProperties/ReferenceParameters, which are opaque tokens embedded in the reference and propagated back to the service during message dispatch.

SOAP message invocations to the service include each ReferenceProperty and ReferenceParameter bound as a separate SOAP header block. A SOAP request sent to the entity identified by the EndpointReference above looks like:

```
      <soap:Envelope

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
              xmlns:myns="http://example.com/ref-props-params">
          <soap:Header>
              ...
              <wsa:To>http://myurl.com/myservice</wsa:To>
              <myns:CustomerID>Acme Inc.</myns:CustomerID>
              <myns:SessionID>session$id:0123456789</myns:SessionID>
              ...
          </soap:Header>
          <soap:Body>
              ...
          </soap:Body>
      </wsa:EndpointReference>
```

WS-Addressing is intended to provide a building block for higher level abstractions. In the specification itself, what exactly an EndpointReference including ReferenceProperties/ReferenceParameters refers to is not defined. It could be an entity modeled by the service, or alternatively, a resource managed by the service. The specification also refers to Web service "instances"; since there is no such entity in the vocabulary of Web services, it's possible the authors of the WS-Addressing specification intend to release another specification that provides an appropriate definition. The WS-Resource Framework (RF) as submitted to OASIS was a retrograde set of specifications that built on WS-Addressing to use the base EndpointReference as a building block to structure a model that resembles distributed object

systems like CORBA. The WS-RF specifications provided an elaborate framework that attempts to cope with the brittleness problems created by the reference session model. The design intention of WS-RF was to provide a general model for stateful Web services (Foster et al, 2004).

Because WS-Addressing is clearly based on the session reference model, a natural way to think about the ReferenceProperties/ReferenceParameters is to compare them to the Object Key in the IOR found in CORBA systems. The ReferenceProperties/ReferenceParameters provide a mechanism for reestablishing the execution context for the request message once it reaches the service network endpoint. Because WS-Addressing is a "building block" specification for protocols and product features, it is positioned to provide a fundamental session model for the Web services architecture moving forward.

Web services sessions using WS-Context

Since December of 2003, the Web services community has been developing a standard called WS-Context in the OASIS consortium to provide an explicit session model for Web services. WS-Context provides a session construct that is decoupled from application-domain service representations. The specification defines a basic context structure, explains how it can be linked explicitly to the activity concept, and provides WSDL bindings for services that manage activity-sessions. The specification is decomposable, so that session models that don't map precisely to the activity concept can use the context structure as well; currently proposed specifications that use the activity session model include WS-Reliability and WS-ReliableMessaging, Ws-Coordination and WS-CoordinationFramework, WS-AtomicTransaction, WS-BusinessActivity, WS-TXM, SOAPConversations and WS-

Enumeration, suggesting that the model is particularly robust for building session based protocols.

The WS-Context context structure looks like:

```
<xsd:complexType name="ContextType">
   <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      <xsd:element name="context-identifier" type="
tns:contextIdentifierType xsd:"/>
      <xsd:element name="context-service" type="ref:ServiceRefType"
minOccurs="0"/>
      <xsd:element name="type" type="xsd:anyURI"/>
      <xsd:element name=”context-manager” type=”ref:ServiceRefType”
minOccurs=”0”/>
      <xsd:element name="parent-context" type=”tns:ContextType”
minOccurs="0">
   </xsd:sequence>
   <xsd:attribute name="timeout" type="xsd:int" use="optional"/>
   <xsd:attribute ref=”wsu:Id” use=”optional”/>
</xsd:complexType>
```

There are several important aspects of the Context structure:

1. The context provides a required type attribute to identify the type of session model that is being represented as the intended interaction semantic. This allows processors to unambiguously understand the “meaning” of the contextualization header.
2. Contexts contain explicit lifecycle semantics. A time to live is normally associated with sessions. This provides a normal framework for understanding the validity and invalidation rules for a session.
3. The context can contain arbitrary data elements. Depending on the rules governing the system and protocols in use, these data elements may be arbitrarily augmented by the clients and service endpoints. This allows for sessions to be

shared across multiple services and for the shared session content to be change over time.

4. Contexts may be passed by-value or by-reference. By-value contexts support familiar use cases like distributed transaction processing. By-reference contexts are particularly useful for business protocols that require large amounts of data to be included with messages, for example, legal restrictions related to a sales process.
5. The Context structure acts as a wrapper for information related to the session protocol. The SOAP binding for WS-Context includes the Context itself as a SOAP header, maintaining clear encapsulation rules for contextualized messages.

These characteristics allow the Context to be used for a variety of protocols, including the direct management of application state. Most importantly, the session concept is loosely coupled with respect to communication channels and service endpoints: the session may be used in conjunction with a service for a short period or even shared across multiple services. Late binding also means that protocols may use WS-Context to support either ephemeral or long-lived sessions in associated with a fixed service endpoint definition as appropriate within an application.

As mentioned, the context is bound to SOAP messages as a header. An example SOAP message including a context header would appear on the wire as:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
  <soap:Header>
    <context xmlns="http://docs.oasis-open.org/wscaf/2004/09/wsctx"
timeout="100"
      xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
      xmlns:soapbind=http://schemas.xmlsoap.org/wsdl/soap/
      soap:mustUnderstand="1">
      <context-identifier>
        http://docs.oasis-open.org/wscaf/2004/09/wsctx/abcdef:012345
      </context-identifier>
      <type>
        http://docs.oasis-open.org/wscaf/2004/09/wsctx/context/type1
      </type>
    </context>
  </soap:Header>
  <soap:Body>
    <!-- Application Payload -->
  </soap:Body>
</soap:Envelope>
```

The WS-Context specification also provides two (optional) service endpoint interfaces for managing the lifecycle and contents of by-reference contexts, the ContextService and the ContextManager. The ContextService provides the ability to begin activities and terminate (complete) activities. It’s important to note that this interface is optional: services could create, use and manage Contexts like HTTP Cookies. The ContextManager allows agents to dereference Contexts for up-to-date data content and is used in conjunction with by-reference Contexts.

The target use cases for WS-Context include support for transaction models in the Web services environment, reliable messaging sessions, structured/global interaction models like choreographies and conversational state management in temporally coupled service interactions. It is intended to be used as a building block by other specifications that require session constructs – in fact, several specifications related to transaction protocols in OASIS are already building on WS-Context -- and to supply execution contexts for application level state correlation. In the latter case, instead of adopting a reference-based session model like CORBA, the application

session model can be used in ways that are largely analogous to HTTP sessions that are represented by a Cookie.

WS-Context provides a more generalized session model than what is present in WS-Addressing and is focused on session support with minimal coupling between services. The WS-Context model is also consistent with the model of Web services provided by WSDL: all clients have an equivalent view of the service, references to which are based on the wsdl:service element.

Observations

Right now, there are two primary models for the session concept that are being defined by companies participating in defining Web services: the WS-Addressing EndpointReference with ReferenceProperties/ReferenceParameters and the WS-Context explicit context structure. The WS-Addressing session model provides coupling between the web service endpoint information and the session data, which is analogous to object references in distributed object systems. WS-Context provides a session model that is an evolution of the session models found in HTTP servers, transaction, and MOM systems.

As we observed, the fundamental, widely-accepted characteristics of Web services are that they can be described in WSDL and are accessed via SOAP. WS-Addressing EndpointReferences plus ReferenceProperties/ReferenceParameters do not map to any concepts described in WSDL. WS-Context allows the session requirements to be described in the WSDL SOAP bindings, by specifying the necessary headers. Capabilities related to sessions can also be described in some kind of policy description for the service.

We also considered three other characteristics that made web services attractive for integrating computing systems. Let's consider how both session models affect these characteristics:

1) Web services emphasize loosely coupled systems.

WS-Addressing EndpointReferences with ReferenceProperties/ReferenceParameters tightly couple the session to the reference. Clients cannot switch or alter the interaction semantic with respect to the service. Clients must maintain a special reference on a per-relationship basis with each service, further coupling the service client and the service itself. This has two important consequences: it creates a brittle relationship between the client and the network service in which the client's understanding of the service is limited to a particular session. Termination of that session invalidates the client's communication channel to the service. Secondly, this results in a scalablility problem. Clients must contain special reference-pointers to services for each relationship that is linked by the session. Often this results in the unnecessary management and storage of redundant data.

On the other hand, WS-Context allows a service client to more naturally bind the relationship to the service dynamically and temporarily. The client's communication channel to the service is not impacted by a specific session relationship.

This has special implications as we consider scaling Web services from intra-domain deployments to general services offered on the Internet. The current interaction pattern for Web Services is based on coarse-grained services or components. The architecture is deliberately not prescriptive about what happens behind service endpoints: Web Services are ultimately only concerned with the transfer of structured data between

parties, plus any meta-level information to safeguard such transfers (e.g., by encrypting or digitally signing messages). This gives flexibility of implementation, allowing systems to adapt to changes in requirements, technology etc. without directly affecting users. It also means that issues such as whether or not a service maintains state on behalf of users or their (temporally bounded) interactions, has been an implementation choice not typically exposed to users.

If a session-like model based on WS-Addressing were to be used when interacting with stateful services, then the tight coupling between state and service would impact on clients. As in other distribution environments where this model is used (e.g., CORBA or J2EE), the remote reference (address) that the client has to the service endpoint *must* be remembered by the client for subsequent invocations. If the client application interacts with multiple services within the same logical session, then it is often the case that the state of a service has relevance to the client only when used in conjunction with the associated states of the other services. This necessarily means that the client must remember each service reference and somehow associate them with a specific interaction; multiple interactions will obviously result in different reference sets that may be combined to represent each sessions.

For example, if there are N services used within the same application session, each maintaining m different states, the client application will have to maintain N*m reference endpoints. It is worth remembering that the initial service endpoint references will often be obtained from some bootstrap process such as UDDI. But in this model, these references are stateless and of no use beyond starting the application interactions.

Subsequent visits to these sites that require access to specific states must use different references in the WS-Addressing model.

This obviously does not scale to an environment the size of the Web. However, an alternative approach is to use WS-Context and continue to embrace the inherently loosely-coupled nature of Web Services. As we have shown, each interaction with a set of services can be modeled as a session, and this in turn can be modeled as a WS-Context activity with an associated context. Whenever a client application interacts with a set of services within the same session, the context is propagated to the services and they map this context to the necessary states that the client interaction requires.

How this mapping occurs is an implementation specific choice that need not be exposed to the client. Furthermore, since each service within a specific session gets the same context, upon later revisiting these services and providing the same context again, the client application can be sure to return to a consistent set of states. So for the N services and m states in our previous example, the client need only maintain N endpoint references and as we mentioned earlier, typically these will be obtained from the bootstrap process anyway. Thus, this model scales much better.

2) Web services do not expose artifacts of the implementation details or component model used to implement the service.

WS-Addressing EndpointReferences with ReferenceProperties/ReferenceParameters naturally encourage modeling components and/or resources behind the service façade by focusing attention on pointers to something other than the service itself.

The utility of this design is the ability to export references to abstractions with memory across invocations, rather than services themselves. This shifts the design center from self-contained messages and toward operations that offer functionality based on the identity encoded in the session reference. Consider a purchase order. A message-centric system might accept a purchase order as a self-contained document. In this case, WSDL operations would require purchase orders as input messages for a few coarse grained functions. Using the session reference model, a purchase order might be exposed to users as an EndpointReference to a purchase order managed by the system. Operations would tend to be fine-grained modifications to the properties of the purchase order identified by the reference that induce state changes on an internal representation of a purchase order abstraction.

WS-Context focuses on the expression of a session semantic only for the duration of the active session. It bears no relationship to the backend resources or implementation strategy employed by the service. Even if used to manage conversational state explicitly, there is no built-in relationship between WS-Context and any domain-specific modeling artifacts: the session mechanism and domain models are considered to be orthogonal.

3) WSDL itself is agnostic about session models

Nothing about the semantic of the session model contained in EndpointReferences plus ReferenceProperties/ReferenceParameters can be expressed in WSDL. At best, a factory pattern can be used to return object-reference-like structures to clients. From this perspective, WS-Addressing is broadly misaligned with the normal models provided for developing Web services.

Requirements for WS-Context headers can be expressed directly in WSDL and map naturally to other proposed mechanisms for expressions about the capabilities and requirements of services like WS-Policy.

Conclusions

Design decisions for service implementers are strongly influenced by how session concepts are expressed in the Web services environment. Of the currently proposed models available to capture the session concept, WS-Context focuses on loosely coupled, message-oriented systems design. We believe this is the appropriate model to ensure that Web services successfully fulfill their role as a technology for integrating heterogeneous computing environments with minimal coupling between systems.

References

Booth, David et al (11 February 2004) *Web Service Architecture*, retrieved from http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

Box, Don et al (8 May 2000) *Simple Object Access Protocol (SOAP) 1.1*, retrieved from http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

Box , Don et al (10 August 2004) *Web Services Addressing (WS-Addressing)*, retrieved from http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/.

Christensen, Erik et al (15 March 2001) *Web Services Description Language (WSDL) 1.1*, retrieved from http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

Czajkowski, Karl, et al (5 March 2004) *WS-Resource Framework*, retrieved from http://www-fp.globus.org/wsrf/specs/ws-wsrf.pdf

DeMichiel, Linda G. (12 November 2003) *Enterprise JavaBeans Specification, Version 2.1*. (See http://java.sun.com/products/ejb/)

Fielding, R. et al (June 1999) *IETF Network Working Group Request for Comments: 2616 "Hypertext Transfer Protocol – HTTP/1.1"*, retrieved from http://www.ietf.org/rfc/rfc2616.txt.

Foster, Ian et al (5 March 2004) "Modeling Stateful Resources with Web Services", retrieved from http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

Gudgin, Martin et al (24 June 2003) *SOAP Version 1.2 Part 1: Messaging Framework,* retrieved from http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

Iwasa, Kazunori (24 August 2004) WS-Reliability 1.1, retrieved from http://www.oasis-open.org/committees/download.php/9330/.

Karmarkar, Anish et al (26 April 2004) *WS-MessageDelivery Version 1.0*, retrieved from http://www.w3.org/Submission/2004/SUBM-ws-messagedelivery-20040426/.

Kristol, D. et al (October 2000) *IETF Network Working Group Request for Comments: 2965 "HTTP State Management Mechanism"*, retrieved from ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt.

Little, Mark et al (8 October 2004) *WS-Context*, retrieved from http://www.oasis-open.org/committees/download.php/9806/.

OASIS Web Services Composite Application Framework (WS-CAF) TC (2003), retrieved from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf.

OASIS Web Service Resource Framework (WSRF) TC (2004) retrieved from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

Object Management Group (March 2004) *Common Object Request Broker Architecture: Core Specification*. (See http://www.omg.org/docs/formal/04-03-01.pdf)

Sun Microsystems (2003) *Java Remote Method Invocation Specification*, retrieved from http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html.

W3C Web Services Addressing Working Group (7 November 2004) retrieved from http://www.w3.org/2002/ws/addr/