2SAP: A FLEXIBLE ARCHITECTURE FOR WEB SERVICE ENABLEMENT OF COMMUNICATION SERVICES

Li Li, Wu Chou, Dan Zhuo and Feng Liu
Avaya Labs Research, Avaya Inc., 233 Moun Airy Road, Basking Ridge, USA

Keywords: Two-way Web services, session, meta-services, Web services enablement middleware, stateful transaction, event notification.

Abstract: In this paper, we present a flexible Web service middleware framework based on the approach of 2SAP, two-way Web service application proxy. 2SAP is to address some critical issues in Web service enablement of communication services that require session based stateful transactions, two-way full duplex Web service interaction, asynchronous operations and event notification. We introduce the concept of meta-services and base-services in Web service middleware architecture design, and we investigate three options that can be applied to synthesize the service interface descriptions of these services in current WSDL frameworks. Based on the dependency and interaction pattern analyses, we describe the architecture of 2SAP that modularizes the service implementation and components along two dimensions, i.e. *stateful* vs. *stateless* and *core* vs. *extension*. We show that the architecture of 2SAP can support a variety of collocated and distributed service integration configurations. Our experimental studies indicated that the service platform of the proposed 2SAP architecture can support the needs of Web service and SOA enablement of real-time communication services.

1 INTRODUCTION

Web services have gained tremendous momentum in recent years as an emerging disruptive technology with applications in various market sectors and fields, including telecommunication (Chou et al 2005a, 2005b, 2006a, 2006b, Liu et al 2004, 2006a, 2006b). The fast adoption of Web services reflects the current industrial paradigm shift from Object Oriented Architecture (OOA) to Service-Oriented Architecture (SOA) for business computing. It reinforces the significance and importance of Web service technologies in creating reusable, reliable and scalable business services that are loosely coupled with the physical implementations, and agnostic to hardware devices, operating systems, transport protocols, programming models and languages.

One contributing factor to the success of Web services is the maturity and proliferation of Web service enablement packages and SOAP engines, such as Axis (Axis 2006), and Web service orchestration IDE based on WS-BPEL (WS-BPEL 2006). These Web service packages collectively have made the development, testing and deployment of stateless, synchronous and one-way Web services a relatively straightforward task, when there is a clear boundary and separation between the client and server.

However, as pointed out in several Web service studies (Chou et al 2005b, Li et al 2005a, 2006), the use of Web services to enable communication introduces many new technical challenges. In particular, telecommunication services typically exhibit some distinct characteristics as listed below:

- Stateful transactions: Telecommunication services are stateful in two ways. First, it usually requires the establishment of session association among clients and service providers before any subsequent message exchange can happen. Secondly, the message exchanges in communication often involve many stateful resources.
- Two-way message exchange: In telecommunication services, the role of client and server can be reversed such that a client can act as a server during the interaction.
- Asynchrony: Asynchronous messages are prevalent in communication services because

Li L., Chou W., Zhuo D. and Liu F. (2007).
2SAP: A FLEXIBLE ARCHITECTURE FOR WEB SERVICE ENABLEMENT OF COMMUNICATION SERVICES.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 305-312
DOI: 10.5220/0001291403050312

of the scalability issue and event driven nature of communication services.

- Conversational interaction patterns: The message exchange patterns between the client and server are highly conversational in the sense that the client and server constantly exchange messages back and forth in order to reach a goal.
- Real-time factors: Message interactions must occur in real-time to satisfy the Quality of Service (QoS) requirements for certain communication services.
- Reliability: Message delivery must be reliable. Undelivered, delayed, out-of-order or duplicated messages may put the system in some incorrect state.

To address these issues, one needs to work with the current Web service technologies and protocol standards in order to achieve interoperable solutions between services and platforms. Because Web services are based on XML technologies which support document composition, Web services protocols can be more freely composed instead of strictly layered as in traditional OSI Reference Model for network protocols. This new feature obviously offers architectural design freedom but it posts the following complexities:

- The relationship between services is complex. For example, a communication service may depend on several other Web services, which in turn may depend on each other. Without a more generic framework, the cost of Web service enablement can become prohibitive.
- Protocol standards could be unstable and inconsistent. Standards will evolve at different standard organizations. They can be merged, divided, or rescinded. As a result, the dependencies and references between standards can be out of date or at conflict. It is unrealistic and undesirable to require each Web service implementation to individually deal with these issues where they are common to all services.
- It needs to be able to plug in services on-demand once these services become available. This includes those services implemented by third party vendors as well as the management of those services in service deployments and interactions.
- A Web service middleware framework should be as light-weighted as possible, so that it can run on small devices with limited computing resources, such as a phone. It also must be compatible with current SOAP packages to ensure interoperability.

Motivated by the abovementioned issues and the needs in Web service enablement of communication services, we developed a Web service enablement framework of 2SAP, which stands for 2-way Web Service Application Proxy. The 2SAP framework has been successfully applied to enable several enterprise communication services, including CSTA services (ECMA-366 2004) and Conference services. The communication Web services enabled by 2SAP encapsulate the complexities of telecommunication protocols. It allows programmers and application developers without special training in the field to integrate telecommunication services into various enterprise business applications.

The organization of this paper is as follows. In Section 2, we review the related work and Web service protocols. In Section 3, we describe some critical use cases that motivated this research. In Section 4, we discuss issues of Web service interfaces integration. Based on that, we introduce our design and describe the proposed approach of 2SAP (two-way Web service application proxy) framework. Platform components and service deployment configurations of 2SAP are described in Section 6. Applications and use cases of 2SAP are presented and studied in Section 7. The findings of this paper are summarized in Section 8.

2 RELATED WORK

The runtime architecture of SOAP engines often follows the Chain of Responsibility (Gamma et al 2004) design pattern. The engine provides handler chains that allow customized interceptors (Völter 2005) to be invoked in a prescribed order. Most SOAP engines allow only service level interceptors, but some of them, e.g. Axis2 (Axis2 2006), support message level interceptors as well. Another common feature of SOAP engines is message context, i.e. a "blackboard" that allows interceptors and service object to exchange information asynchronously. Most SOAP engines do not validate messages against XML Schema as it is expensive, but instead perform deserialization of XML message into objects based on XML Schema.

Maheshwari (Maheshwari et al 2004) presented WSMQ, a message-oriented middleware to enhance the reliability of asynchronous Web service interactions. Ostrowski (Ostrowski et al 2006) proposed a hierarchical network architecture in which each node can be a participant in notification

propagations based on the service scopes, sessions and topics.

Hu (Hu et al 2005) described a Web service container architecture for adapting Web services for various backend services. It introduced some useful techniques, such as Web service bean and agent, but it did not address the issues of service interface integration or asynchronous events as the proposed approach of 2SAP does. Weiss (Weiss 2004) proposed a Goal-oriented framework to detect and resolve conflicts between Web services interactions in Web service composition.

Some theoretic analyses and practical solutions to issues in CSTA Web services are discussed in several studies (Chou et al 2005b, Li et al 2005a, 2006) leveraging the Web service standards of WS-Session (ECMA-366 2005), ECMA-348 (ECMA-348 2004), ECMA TR-90 (ECMA TR-90 2005), and WS-Eventing (WS-Eventing 2006). However, the architectural details of 2SAP that enable those services were not presented and addressed in those papers.

3 USE CASE ANALYSES

From message exchange perspective, 2SAP serves as a gateway between Web service clients and backend services that may or may not be Web services based. In general, the backend services can be categorized as either message based system or API based service platform component. Figure 1 illustrates some typical use cases of 2SAP in some practical scenarios.

In use case (a), 2SAP provides both client and middleware to access the backend CSTA server. It provides support for ECMA-323 XML messages over TCP/IP but without any session or event subscription services. 2SAP needs to translate two-way messages between ECMA-348, which is based on Web services (WSDL), and ECMA-323, which is a non-Web service XML protocol. This is in addition to manage the session and event subscription services for the backend server.

In use case (b), a conference server provides a Java SDK for service access. The SDK provides session and listener Java objects to access session and event managements respectively. 2SAP needs to translate between SOAP messages and Java objects and map session and Web services for events into SDK logic and API.

In use case (c), 2SAP acts as a SOAP broker between a BPEL client and a group of backend Web services. Here 2SAP manages the session and event services on behalf of the backend services and perform content-based routing.

In these use cases, because the complexities of managing two-way, stateful, and asynchronous Web service interactions are encapsulated in 2SAP, backend services can focus on their business logic.

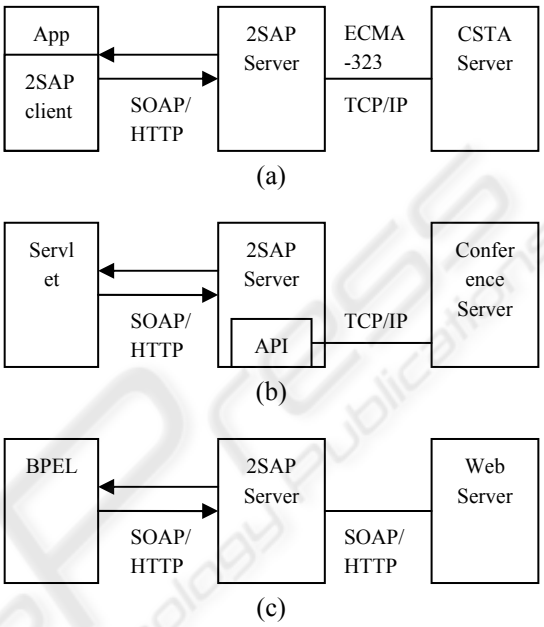

Figure 1: Typical use cases of 2SAP server.

4 SERVICE INTERFACE INTEGRATION

One important issue is how to expose the integrated service descriptions to the Web service clients using standard WSDL definitions in a cohesive and flexible way. The way the services are synthesized determines how they should be accessed by the clients and how they can be implemented and maintained by the providers.

It is clear from the use cases that there are actually two levels of Web services: the *base* services that provide specific functions in a particular domain, and *meta-services* that enforce rules for certain *common aspects*, such as session, event subscription, security, etc., of the base services. Without a base service, a meta-service has no effect on real world. For example, WS-Session services must be combined with some base services to create fully functional services. On the other hand, meta-services are independent of base services and they can be synthesized with different and even multiple base services.

Given a WSDL description of a base service, we consider the following factors in integrating meta-services with the base services:

- Semantic relations between services
- Lifecycle dependency of services
- Modularity of services
- Transport binding control of services

In WSDL 1.1 and WSDL 2.0 frameworks, there are three options to incorporate meta-services into a base service, i.e. *interface*, *port* and *service*.

In interface based approach, the appropriate portType or Interface of the base service is extended to include the meta-service operations. Semantically, the base service *inherits* the meta-service (WSDL 1.1 does not actually support portType inheritance and the operations have to be copied over). This approach enforces that the base and the meta-services have the same lifecycle. However, modularity of services is sacrificed in WSDL 1.1. Multiple inheritance mechanism in WSDL 2.0 promotes modularity but may introduce common problems in inheritance such as operation conflicts. As base services and meta-services converges at interface level, they will be bound to the same transport and accessed on a single endpoint.

In port based approach, the port for the meta-services is added to the ports of the base service. Semantically, the base service *aggregates* the meta-service. This is consistent with WSDL 1.1 view that a service definition groups *related* ports (endpoints). This is also a common practice in the Internet, where related protocols run on different transport ports (for example RTP and RTCP). The base service and meta-service may or may not have the same lifecycle, depending on whether they are bound to the same endpoint. It is possible for several base services to share a meta-service. As this approach does not require any modifications to the service interface of the base service, interface modularity is therefore maintained. Port based approach allows base service and meta-service use different transports, for example JMS for reliable event subscription and UDP for fast event delivery.

In service based approach, the meta-service is defined as a new service in the base service WSDL file. Semantically, the base and meta-services are *coordinated* by the fact that service definitions have the same target namespace. The base service and meta-service may or may not have the same lifecycle, depending on whether they are bound to the same endpoint. It is possible for several base services to share a meta-service. This approach requires minimum changes to the base service WSDL. Clearly, different transport bindings can be used for base services and meta-services.

5 PROTOCOL INTERACTIONS

In the proposed 2SAP approach, there are three types of relationship between services and protocols, i.e. *dependency*, *binding* and *utility*.

Service *X* depends on service *Y* if the specification of *X* reference XML definitions from *Y*. For instance, WS-Eventing references WS-Addressing definitions.

Binding maps an abstract service/protocol to a concrete protocol. For instance, all services bind to SOAP.

In utility, service *X* uses resources maintained by service *Y* at runtime. For example, ECMA-348 services use event subscriptions created by WS-Eventing and sessions created by WS-Session. In addition, ECMA-348 can create its own monitor resources and store them in a session resource managed by WS-Session. WS-Eventing can also use WS-Session when a client subscribes to events in the sessions created by WS-Session. On the other hand, WS-Session uses subscriptions of WS-Eventing to deliver session events. Base services may also use Generic Event Sink service to receive the subscribed CSTA events (ECMA TR-90 2005).

Figure 2 illustrates the relationships among some services that 2SAP supports, where dependency is represented by solid arrows, binding by solid lollipops, and utility by dashed arrows.

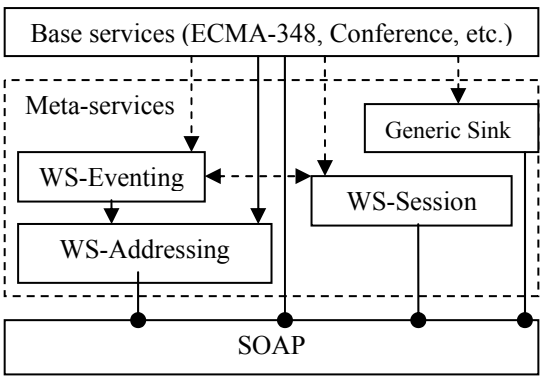

Figure 2: Three types of service/protocol relationships.

To realize these relationships at runtime, we must allow services to interact with each other at two levels: *resource* and *message*.

At resource level, a service needs to access the resources created by another service or create certain relationship between resources across services. In

this case, the service can be modelled as a "factory" that manages the lifecycle and persistence of its resources and expose them through well-defined APIs.

At the message level, a service needs to access or change the messages as well as call flows of another service. For example, the ECMA-348 service must analyze WS-Eventing subscribe message to determine if a session is the target of the subscription. If so, the link between the subscription and session must be established. Otherwise, it must throw a fault message. To permit this kind of interaction, the service must separate its core logic from extensions so that the extension can be manipulated while the core is protected and can be invoked from different context. For extension, we found the Interceptor design pattern is the most flexible choice as it leaves the client, service and the interceptors loosely coupled and fits well with current SOAP framework.

6 2SAP ARCHITECTURE

To modularize the meta-service implementation, we separate each service according to two dimensions: *stateless* vs. *stateful*, and *core* vs. *extension*. The stateless component is involved in the processing of messages in stateful interactions but relies on stateful component for managing stateful resources. In particular, the stateful component provides local and remote access APIs to manage subscriptions and sessions as time-based leases (Völter et al 2005). The core component implements the mandatory logic of the service and leaves the extension part to the integration interceptors. The main components of 2SAP architecture are illustrated in Figure 3 where stateless components are shown as rectangles and stateful ones as disks. An arrowed line indicates the dependency between components and a circle indicates that the particular service provides hooks for interceptors.

To eliminate the dependency between event subscription and diverse event sources, we introduce a topic tree data model to facilitate creation of hierarchical event topics. The topic tree abstracts resources from various services into a uniform topic hierarchy with event propagation rules. Such abstraction enables a client to subscribe to a group of event sources which may not exist yet.

Figure 4 illustrates a typical topic tree created for ECMA-348 services when one subscription is created for a session and the other for a monitor within the session. The dotted arrows indicate the possible paths of event propagation. In this topic tree, if Session1 terminates, Subscription1 will receive the event notification. Events from Monitor1 will also propagate to both Subscription1 and Subscription2 along the topics, if event bubbling is enabled.

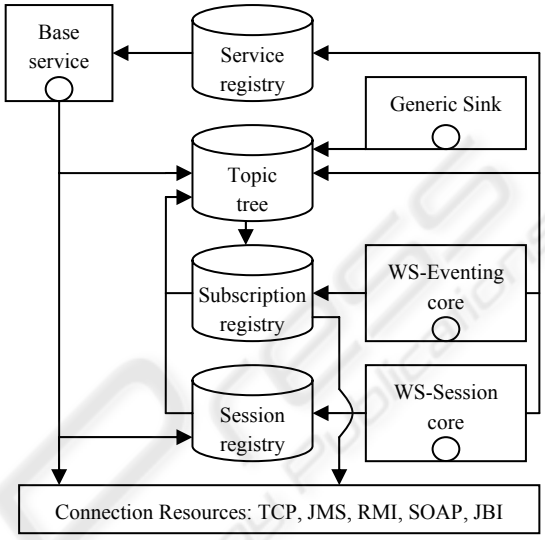

Figure 3: Main components of 2SAP framework.

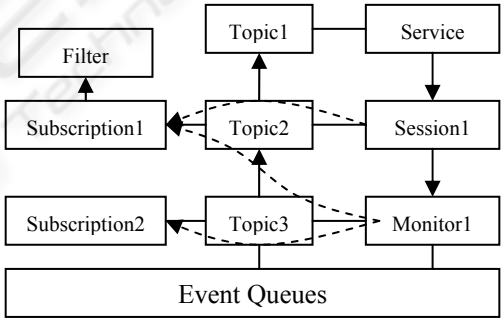

Figure 4: A typical event topic tree created by 2SAP for ECMA-348 Web service.

The core of services provides well-defined interfaces so that it can be invoked either from SOAP engine or directly by client. Combined with interceptors, same logic can be used conveniently to create *virtual* resources that span client and server. Figure 5 illustrates the message flow of WS-Session StartApplicationSession that creates an association consisting of a local session and a remote session using the same core component intercepted by a connector. This association can be shared by many clients and servers, as defined by WS-Session.

To allow changes for the extension while protecting the core logic of a service, the integration

interceptors can read the messages directly and change the messages indirectly using message context.

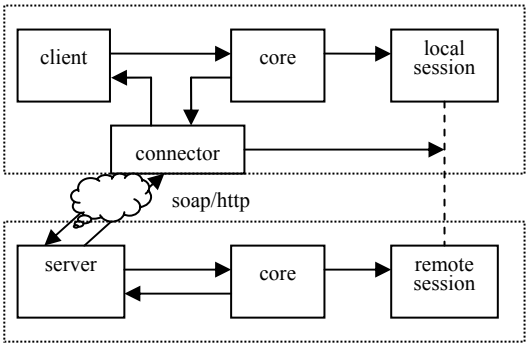

Figure 5: Creation of association between client and server with reusable WS-Session core components.

For example, two integration interceptors, one from WS-Session, the other one from ECMA-348, intercept WS-Eventing event subscription message and create topics for session and monitor services respectively as shown in Figure 4. In case of Generic Event Sink service, interceptors act as event listeners that handle incoming events and dispatch them to proper topics. As a result, both WS-Eventing and Generic Event Sink core components are agnostic to any service specific logic and resources.

Without sharing, it appears that *N* interacting services could result in *N(N-1)* integration interceptors. Fortunately this is not the case because some services can share an interceptor with common behaviour. For example, ECMA-348 and WS-Eventing services share a session interceptor that retrieves a session object of WS-Session from the session registry according to the session ID provided in the SOAP message.

The following subsections illustrate several typical scenarios that 2SAP meta-services are integrated with base services.

6.1 Collocated Services

In collocated case (Figure 6a, 6b), all services are deployed at one endpoint for inheritance based integration. To implement this configuration with SOAP engines that permit only one service object per endpoint, the meta-services (MS) can be implemented as interceptors of the base service (BS). Each meta-service module has to demultiplex incoming messages (Figure 6a). For example, a WS-Session interceptor will only handle its messages and skip the others. While this approach is highly modular and configurable, it has some drawbacks: 1) sequential message dispatching is suboptimal; and 2) meta-services have to cope with unchecked raw SOAP messages instead of well-formed objects deserialized by the SOAP engine.

To overcome these limitations while maintaining the benefits, a dispatcher is employed to associate an endpoint with multiple service objects, using a routing table which is configurable at deployment and runtime. To allow services and interceptors to exchange information, a context is maintained by the dispatcher and shared across message flows (Figure 6b).

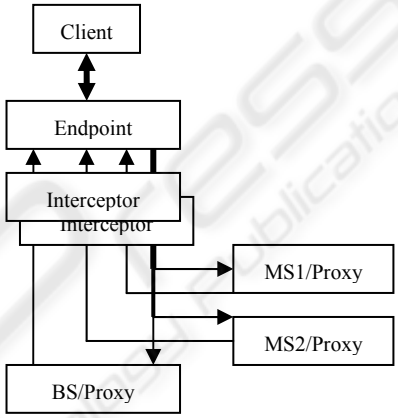

Figure 6a: Collocated services integration with interceptors.

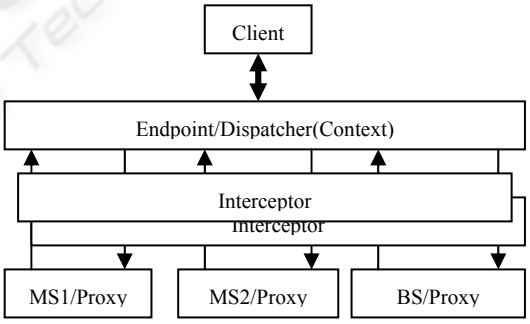

Figure 6b: Collocated services integration with a dispatcher.

In both configurations, the meta-services and base services can be distributed to different hosts by using proxies. The architecture in Figure 6b is particularly suitable for content-based message routing. This architecture supports various distributed event broker architectures outlined in (Li et al 2006) where interceptors are used to transform WS-Eventing SOAP messages.

6.2 Distributed Services

In this setup (Figure 7), services are deployed at different endpoints, possibly on different machines, using port or service based description integration.

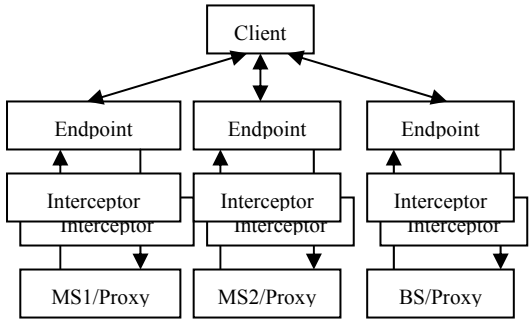

Figure 7: Distributed services integration.

To reduce location coupling in this situation, 2SAP can dynamically provision a registered base service according to the access control and return the base service endpoint to the client when a session is established.

6.3 Benefits of 2SAP

The modularization and abstraction of meta-services and its resources in 2SAP creates the following benefits:

- Each service is self-contained and can be implemented independently even though it may depend on other services.
- The stateless and stateful components of a service can be collocated or distributed to provide flexible architecture.
- The topic tree, factory pattern and integration interceptors support asynchronous event management and allow services to be integrated while loosely coupled.
- Reusable core components facilitate development 2SAP client and server for peer-to-peer Web services.
- The three service description integration options (inheritance, aggregation and coordination) can be supported with collocated and distributed services which provide design freedom in scalable service enablement and integration.

7 SERVICE ENABLEMENT APPLICATIONS

The architecture and techniques developed in 2SAP have been applied to Web service enablement of some backend communication services, including ECMA-348 service (Chou et al 2005a, 2005b, 2006a), real-time multimedia conference services, call center and dialog system services (Li et al 2005b), and integrated with standard based SOA framework of JBI (Java Business Integration) (JBI 2005) service bus. It is also applied to Web service enable communication endpoints (Chou et al 2006b, Liu et al 2004). The performance of 2SAP architecture is satisfactory for communication requirements that end-to-end signal delay is within 300 milliseconds (ms). In recent experiments, we observe that the average processing time per message for meta-services is within 20 ms while the average roundtrip time of meta-service message is about 60 ms, which includes local network transport time and SOAP engine (Axis 2006) processing time. The measurements were obtained on laptop computers with 1.6 GHz CPU and 512 MB memory, running Windows XP professional.

8 SUMMARY

In this paper, we studied several important and practical issues in Web service enablement for communication services that require two-way, stateful and asynchronous interactions. We presented some critical use cases that motivated our 2SAP, two-way Web service application proxy framework. From these use cases, we proposed the concept of base service and meta-service and we analyzed three options, interface, port and service, to synthesize the service interface descriptions of these services in current WSDL frameworks. Following these options, we discussed the relationships between services and protocols, i.e. dependency, binding and utility, as well as resource and message level interactions that realize these relationships. These analyses and discussions led us to the 2SAP architecture which is based on modularization of each service along two dimensions: stateful vs. stateless and core vs. extension. 2SAP is implemented with well-established factory and interceptor design patterns. We demonstrate the versatility of 2SAP with architectural variations that 2SAP can support. Finally, some communication applications enabled by 2SAP and performance of

2SAP were discussed to demonstrate the approach of 2SAP is feasible for Web service and SOA enablement of real-time communication.

Research is on-going to further extend and enhance the 2SAP framework for Web service enablement, including the improvement of the backboard data model, integration with additional meta-services to improve the platform reliability and security, etc. The initial results indicated that the proposed 2SAP framework is quite extensible to support these extensions in a structural way.

REFERENCES

Axis, 2006. Apache Web Services - Axis, *http://ws.apache.org/axis/*.

Axis2, 2006. Apache Axis2/Java, *http://ws.apache.org/axis2/*.

Chou, W., Li, L., Liu, F., 2005a. Web Service for Communication Service Management, *The 17th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), page 584-589, Taipei, Taiwan, July 2005.*

Chou, W., Li, L., Liu, F., 2005b. Web Service Enablement of Communication Services, *Proceedings of ICWS 2005, Volume 2, page 393-400, Orlando, Florida, July 2005.*

Chou, W., Liu, F., Li, L., 2006a. Web Services for Telecommunication, *Proceedings of Advanced International Conference on Telecommunications (AICT 2006), AICT 11: Special Mechanisms, 6 pages, Guadeloupe, French Caribbean, February 2006.*

Chou, W., Li, L., Liu, F., 2006b. Web Service Initiation Protocol for Multimedia and Voice Communication over IP, *Proceedings of IEEE International Conference on Web Services (ICWS 2006), page 515-522, Chicago, September 2006.*

ECMA-348, 2004. Standard ECMA-348 Web Services Description Language (WSDL) for CSTA Phase III 2nd edition (June 2004), *http://www.ecma-international.org/publications/standards/Ecma-348.htm.*

ECMA-366, 2005. Standard ECMA-366 WS-Session - Web Services for Application Session Services (June 2005), *http://www.ecma-international.org/publications/standards/Ecma-366.htm*

ECMA TR-90, 2005. Technical Report TR/90 Session Management, Event Notification, and Computing Function Services - Amendments for ECMA-348, December 2005, *http://www.ecma-international.org/publications/techreports/E-TR-090.htm*

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

Hu, J., Guo, C., Zou, P., 2005. WSCF: A Framework for Web Service-based Application Supporting Environment, *Proceedings of ICWS 2005, Volume 2, page 445-452, Orlando, Florida, July 2005.*

JBI 2005. JSR 208: Java™ Business Integration (JBI), *http://jcp.org/en/jsr/detail?id=208*

Li, L., Chou, W., 2005a. Two-way Web Service: from Interface Design to Interface Verification, *Proceedings of ICWS 2005, Volume 2, page 525-532, Orlando Florida, July 2005.*

Li, L., Chou, W., Liu, F., 2005b. An Extensible Three-tier XML Dialogue System Architecture for Multimodal Interaction and Automated Agent Services, *Proceedings of 9th IASTED Conference on Internet and Multimedia Systems, and Applications, page 13-17, Hawaii, August 2005.*

Li, L., Chou, W., 2006. Semantic Modelling and Design Patterns for Asynchronous Events in Web Service Interaction, *Proceedings of IEEE International Conference on Web Services (ICWS 2006), page 223-230, Chicago, September 2006.*

Liu, F., Chou, W., Li, L., Li J., 2004. WSIP-Web Service SIP Endpoint for Converged Communication over IP, *Proceedings of 2004 IEEE International Conference on Web Services (ICWS2004), San Diego, California, USA, page 690-697, July 6-9, 2004.*

Liu, F., Wang, G., Li, L., Chou, W., 2006a. Web Services for Distributed Communication Systems, *Proceedings of 2006 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI 2006), page 1030-1035, Shanghai, China, June 2006.*

Liu, F., Wang, G., Chou, W., Lookman, F. Li, L., 2006b. Target: Two-way Web Service Router Gateway, *Proceedings of IEEE International Conference on Web Services (ICWS 2006), page 629-636, Chicago, September 2006.*

Maheshwari, P., Tang, H., Liang, R., 2004. Enhancing Web Services with Message-Oriented Middleware, *Proceedings of IEEE International Conference on Web Services (ICWS 2004), page 88-95, San Jose, July 2004*

Ostrowski, K., Birman, K., 2006. Extensible Web Services Architecture for Notification in Large-Scale Systems, *Proceedings of IEEE International Conference on Web Services (ICWS 2006), page 383-392, Chicago, September 2006.*

Völter, M., Kircher, M., Zdun, U., 2004. Remoting Patterns, *John Wiley & Sons, 2004.*

Weiss, M., Esfandiari, B., 2004. On Feature Interactions among Web Services, *Proceedings of IEEE International Conference on Web Services (ICWS 2004), page 88-95, San Jose, July 2004*

WS-BPEL, 2006. Web Services Business Process Execution Language 2.0, *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel*

WS-Eventing, 2006. Web Services Eventing (WS-Eventing), W3C Member Submission 15 March 2006, *http://www.w3.org/Submission/WS-Eventing/*