

SMS – A Share Memory Service for Web Service Oriented Applications

Jui-Meng Chen, Jia-Wei Lee, Bo-Fu Shen, and Jung-Sing Jwo
Department of Computer Science and Information Engineering
Tunghai University
Taichung, Taiwan
E-mail: jwo@mail.thu.edu.tw

ABSTRACT

Software as services, or web service oriented application can be treated as the next evolution technology for implementing dynamic e-Business solutions. The major advantage of using web services is that they can be considered as loosely coupled dynamically bound components with great interoperability. However, it becomes complicated to implement a web service when its states or data are required either to be passed or to be shared by the other services. In this paper a new service called Share Memory Service (SMS) is proposed. It provides a mechanism for the services in a web service oriented application to pass session context and share their data or objects transparently. Web service oriented applications built upon SMS are called SMS applications. A set of API for developing SMS applications is also provided. In this research, we use Java technology to verify our proposed solution. Furthermore, in order to simplify the coding effort, five new Java pseudo-instructions are introduced. With these new instructions, developers need not use the API directly when developing an SMS application. Also the concept of the capability of using global data and objects in a web service oriented application can simplify the system design process.

Keywords: web services, SOAP, web service oriented applications, share memory service, SMS applications

1. INTRODUCTION

The previous major distributed computing technologies adopted in the Internet e-Business solution, such as CORBA and Java RMI, are considered to be tightly coupled between the cooperating components. The shared context among these components should have strong agreement. The drawback with this is that a change to any component can cause breakage in a variety of dependent applications. This is in part responsible for the limitations of system development and the high cost of system maintenance. Therefore, the current trend in the enterprise application space is moving away from tightly coupled monolithic systems [8-11]. *Web services*, also recognized as *loosely coupled* dynamically bound components, are more likely to dominate the next generation e-Business applications [1, 3, 5, 8-11].

The concept of web service is an application identified by a URI. Its interfaces and binding are defined, described and discovered by XML artifacts. It supports direct interactions with other services using XML based messages via *Simple Object Access Protocol (SOAP)* [4]. The advantage of using web services is that they make an application's functionality available over the Internet in a standardized and programmatic manner. Applications that could not be accessed except by following rigid proprietary protocols are now accessible over the Internet using the same infrastructure. Furthermore, applications can talk to each other regardless of the language that they were developed in, the platform they were developed for or the object models and internal protocols they use.

Software as services, or *Web Service oriented Application (WSOA)* is a set of cooperating web services to perform a particular task. The sequence of the service invocations in a web service oriented application is called a *service flow*. When developing a web service oriented application, unlike traditional application, with the universally agreed specification developers can dynamically integrate the web services they want and deploy the related web services together to form the application. In fact, we can treat WSOA as a conceptual architecture for implementing dynamic e-Business solutions. Undoubtedly, WSOA is the next evolution of the software architecture.

As we have mentioned in the above paragraph, the major advantage of using web services is that they can be considered as loosely coupled dynamically bound

components with great interoperability. However, this advantage could create implementation difficulty when considering the following two issues. The first issue is related to the *state* or *session context* passing across different services in a web service flow. The second issue is the requirement of sharing *global* data and objects among all the services in a web service oriented application. In fact, it becomes complicated to implement a web service when its objects are required either to be passed or to be shared by the other services. Also the more objects are shared, the more parameters in a SOAP message are created. Usually, a large SOAP message is inconvenient to be handled when implementing a service.

In addition to the above two implementation issues, from the design point of view, the concept of the capability of using global data or objects in a web service oriented application indeed can simplify the system design process. Therefore, in this paper we propose a new service called *Share Memory Service (SMS)* such that it provides a mechanism for the services of a web service oriented application to pass session context and share their data or objects transparently. The web service oriented applications built upon SMS are called *SMS applications*. A set of API for developing the web services in a SMS application is also provided. In this research, we use Java technology to verify our proposed solution. Furthermore, in order to simplify the coding effort, five new Java *pseudo-instructions* are introduced. With these new instructions, developers need not use the API directly.

This paper is organized as follows. In Section 2, the concept of web services is discussed. Section 3 introduces the architecture for an SMS application. The implementation of the solution is given Section 4. We explain the usage of our solution by an example in Section 5. Section 6 is the conclusion remark of this research.

2. WEB SERVICES AND SOAP

Web service technology is based on the existing and emerging standards such as *HTTP*, *XML (Extensible Markup Language)*, *SOAP (Simple Object Access Protocol)*, *WSDL (Web Services Description Language)* and *UDDI (Universal Description, Discovery and integration)* [2, 4, 5-7]. For better understanding of the following sections, we briefly describe the concept of web services in the rest of this section.

A web service can be considered as an interface that describes a collection of operations that are network-accessible. The service description of a web service is in standard formal XML notation. It covers the details required for a client or another service to interact with the service. A service description basically includes the message format, transport protocol and the location of the service. The architecture of web services is based on the interactions between three roles – *service provider*, *service registry* and *service requestor*. Figure 2.1 illustrates how web service works. We briefly explain them in the following:

- ✍ Service Provider: A service provider hosts its web service and publishes the service's definition and binding information to the UDDI registry in WSDL.
- ✍ Service Requestor: A service requestor is a client program. It first looks up the service via UDDI and then binds to the service that is hosted by the corresponding service provider.

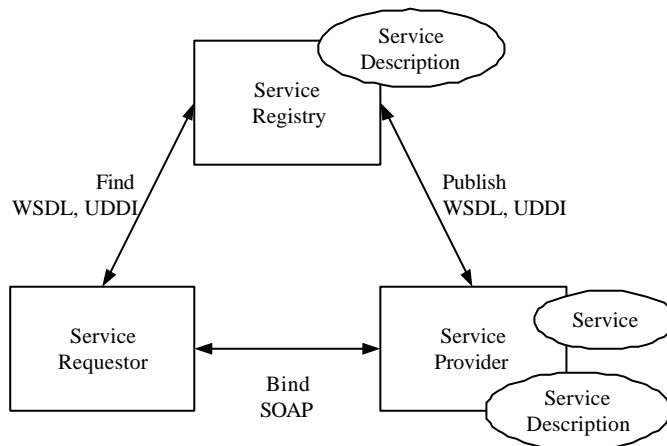


Figure 2.1 Roles and operations in web service architecture.

- ✍ Service Registry: A service registry is responsible for putting all the service definition and binding information available to service requestors. It also requires to manage all the interaction between service providers and service requestors. SOAP is now the standard for communication between service requestors and service registry.
- ✍ Publish: This operation helps service provider publishing its service description to the service registry so that the service requestor can find the

service.

- ✍✍ Find: A service requestor can query the service registry for the type of the required service by the find operation.
- ✍✍ Bind: A service requestor can use bind operation to locate, contact and invoke the requested service.

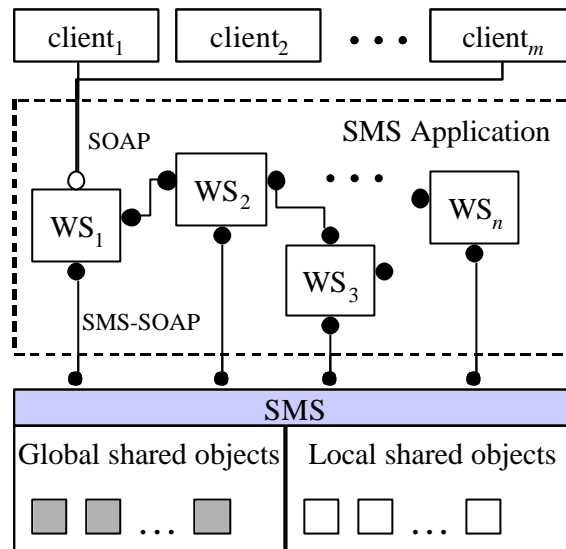
With the above brief introduction of web services, one more important issue related to our work that is required to be explained is SOAP. SOAP is a lightweight protocol for exchanging structured and typed information between peers in a decentralized and distributed environment. It is an XML based protocol that consists of three parts:

- ✍✍ SOAP Envelope: It defines an overall framework for expressing what is in a message, who should deal with it, and whether it is optional or mandatory.
- ✍✍ SOAP Encoding Rule: It defines a serialization mechanism that can be used to exchange instances of application-defined datatypes.
- ✍✍ SOAP PRC Representation: It defines a convention that can be used to represent remote procedure calls and responses.

With the understanding of web services and SOAP, we start to explain our solution from next section.

3. SMS APPLICATION FRAMEWORK

The basic idea behind SMS application framework is that it can let web services define shared objects and let other alliance web services transparently access these objects. SMS application framework consists of three main parts as showed in Figure 3.1. They are SMS, SMS web services and SMS-SOAP. We describe them in the following:



WS_1, WS_2, \dots, WS_n are SMS web services.
Communication links with solid ends are SMS-SOAP.

Figure 3.1. An SMS application framework.

Shared Memory Service

The shared objects of SMS services are actually stored in the SMS memory space and therefore SMS is responsible for handling the requests of accessing these shared objects from SMS web services. In order to maintain the correctness of each request, SMS needs to manage the memory space correctly and effectively. Also to ensure that the SMS web services always access the right states of the shared objects, SMS provides a mechanism to maintain the synchronization of the concurrent operations working upon these shared objects. Since an SMS application can serve different clients simultaneously, the shared object stored in SMS should have its own instance for each client. Shared objects stored in SMS are further divided into two categories. The shared objects that can be access only by the client that invokes the application is called *local shared objects* while the shared objects that can be shared across different clients are called *global shared objects*. We will discuss how SMS is implemented in the coming section.


SMS web services:


SMS web services are the web services that can share their states and objects to the other services with the support from SMS. A shared object is declared first by the web service that owns the object and then it is stored into SMS. The other services that want to access this shared object also need to declare it for reference. In order to simplify the invocation of an SMS web service to another SMS web service such that they interact with each other through shared states or objects, the invocations between these two SMS web services are in SMS-SOAP which will be explained next.


SMS-SOAP:

An SMS-SOAP is a SOAP protocol that enables states and objects shared by the two parties of the communication. Also in SMS framework, when web services invoke each other, the session information is wrapped into the SMS-SOAP request or response message. This information is required to help SMS distinguishing the service that triggers it. Clients of an SMS application can invoke the application simply through standard SOAP message. We will further illustrate the design of SMS-SOAP in the next section.

SMS provides three different modes to manage the life cycle of the shared states or objects. They are:

 NEVER_TIME_OUT: When a shared object is declared as NEVER_TIME_OUT, its space will not be reclaimed until the SMS web service invokes the `release()` method.

 DEFAULT_TIME_OUT: When a shared object is declared as DEFAULT_TIME_OUT, it will not be released automatically until the *garbage collector* – the default memory manager in SMS reclaims the space. Garbage collector will not reclaim the space until the write-time or the lease-time of the shared object are expired.

 LRU_TIME_OUT: When a shared object is declared as LRU_TIME_OUT, it will not be released by the garbage collector until the object is recognized as the one that is least recently used.

4. IMPLEMENTATION

In this section, Java technology is selected to implement the solution. The example and API are all in Java.

Figure 4.1 is the component diagram of SMS. They are *Session Controller* (SC), *Memory Binder* (MB), *Access Controller* (AC), and *Garbage Collector* (GC). We describe their functionality in the following:

✍✍ SC:

SC is the interface to handle the requests from SMS web services. It verifies the legality of web services and checks for the Session ID. When a web service sends a request to SMS, *Web Service Verifier* will first verify if the web service is a legal one to prevent web service from being imitated. The information of the legal web service is stored in *Web Service Registry*. If SMS is invoked by a new service in the first time, *UID Generator* will give the service a unique Session ID. SMS web services can use *SOAPMessageProcessor* API listed in APPENDIX A to retrieve the Session ID. *Request Processor* then cooperates with MB to get the default shared virtual memory. SMS web services can use the unique Session ID and the *EntryInfo* API listed in APPENDIX A to access the shared objects.

✍✍ MB:

MB is responsible for initializing the shared virtual memory for the newly declared shared objects. It also helps binding the shared objects to the services according to the Session ID and *EntryInfo*.

✍✍ AC (Access Controller):

Since shared objects can be accessed by various services at the same time, *Access Synchronizer* should guarantee the concurrency control among shared objects and the services are performed correctly.

✍✍ GC (Garbage Collector):

GC is in charge of the reuse of the shared virtual memory. It releases the space by following the life cycle mode of the shared objects.

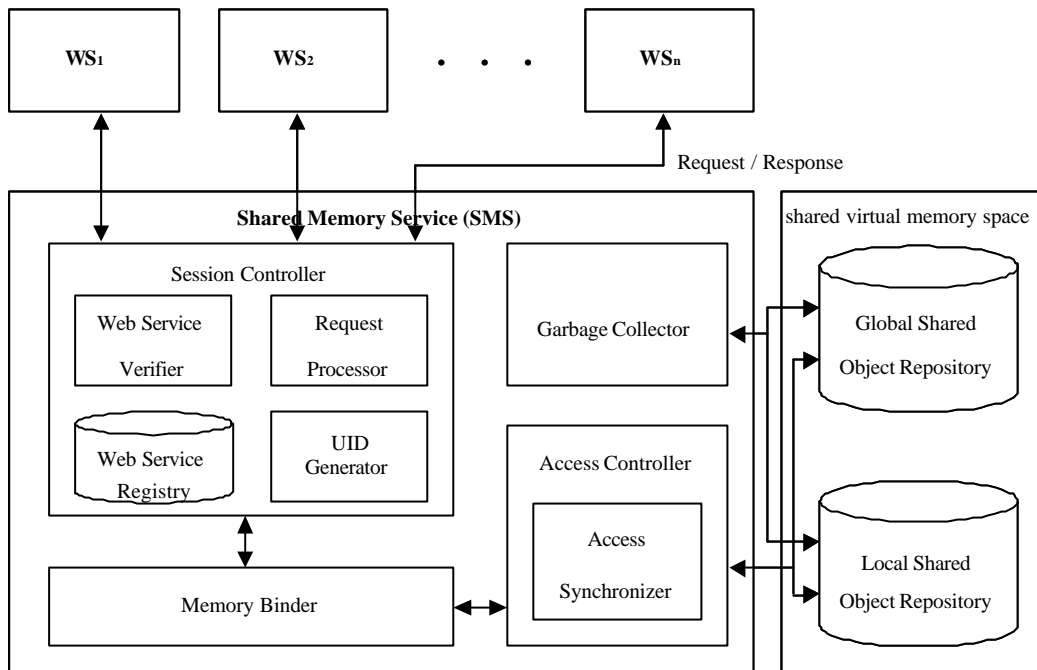


Figure 4.1. SMS component diagram.

When a client invokes an SMS application, the unique session ID which is wrapped into SMS-SOAP message is passed through every web service that the client has invoked. Since each SMS web service can retrieve the session ID from SMS-SOAP message, the session tracking problem among SMS web services can be easily handled. Figure 4.2 shows a fragment of an example SMS-SOAP message. In this example, session tag is used to identify the session and its legal time.

```

...
Content-type: multipart/mixed; boundary="simple boundary"

--simple boundary
Content-type: text/xml; charset="utf-8"
<session>
    <id>WSSMS7810i98</id>
    <created-time>January 1,2002,00:00:00 GMT</created-time>
    <expired-time>January 1,2002,12:00:00 GMT</expired-time>
</session>
--simple boundary--
Content-type: text/xml; charset="utf-8"
//customer XML SOAP message
...

```

Figure 4.2 Fragment of an example SMS-SOAP message.

When developing SMS web services, a set of Java SMS API is provided by our solution. This set of API is given in APPENDIX A. We briefly describe the API as follows. *SOAPSession* is the API to help encapsulate and retrieve session information into and from SMS-SOAP messages respectively. *SOAPMessageProcessor* is responsible for creating and retrieving SMS-SOAP messages. *SMSContextFactory* is used to produce *SMSContext*. SMS web services can use *SMSContext* to lookup the corresponding SMS that is published in UDDI-compliant registry. *EntryInfo* is used to record the description of the shared states and objects. It includes the information about whether the object is a global shared object or a local shared object, the name of the shared object, and the life cycle of the shared object. SMS web services can use *SharedMemoryService* to bind, read and write to the shared objects described in *EntryInfo*. In this API, three exceptions are provided. They are *SMSInternalException* – to indicate SMS runtime exception, *ReadException* – to indicate the shared object is unreadable, and *WriteException* – to indicate the shared object cannot be updated currently.

In order to simply the coding effort, five new Java *pseudo-instructions* are introduced. With these new instructions, developers need not use the API directly. The first instruction is called *GetSOAPSession*. It is used to retrieve the session information from SOAP messages. *SetSOAPSession* instruction is used to embed session information into SOAP messages. *ExternDef* instruction is used to define the global or the local shared objects. For a local shared object, the instruction is followed by the type's name and the variable's name. If a global shared object is defined, another web service name needs to be indicated. When a service wants to read a shared object, it needs to use *ExternRefR* to declare the reference. To refer a local shared object, type name and variable name should be included. When referring global shared object, another service name in a square bracket should be included. When using global shared objects, a sharp-sign (#) is required to be added in front of the global variable name. The final instruction is called *ExternRefW*. It is used to update a shared object. The syntax of this instruction is much similar to *ExternRefR*. However, the implementation of *ExternRefW* needs to handle the possible concurrency control problem.

5. SMS APPLICATION EXAMPLE

In this section, we use a simple online shopping example to illustrate how to use SMS framework to build a web service oriented application. In the application, users can invoke `PaymentService`, `UserRegistryService`, and `UserShoppingService` three services to do online shopping. `UserRegistryService` is responsible for adding, deleting, modifying, and presenting a user's information. `UserShoppingService` is responsible for recording the items, the price, and the quantity a user has purchased. `PaymentService` creates the bill for the user.

Figure 5.1 shows this example as a web service oriented application without using SMS. In this scenario, `PaymentService` sends SOAP message to `UserRegistryService` to request user's information. `PaymentService` also sends SOAP message to `UserShoppingService` to get the items in the user's shopping cart. When receiving the SOAP messages returned from `UserRegistryService` and `UserShoppingService`, `PaymentService` creates the bill for user to confirm.

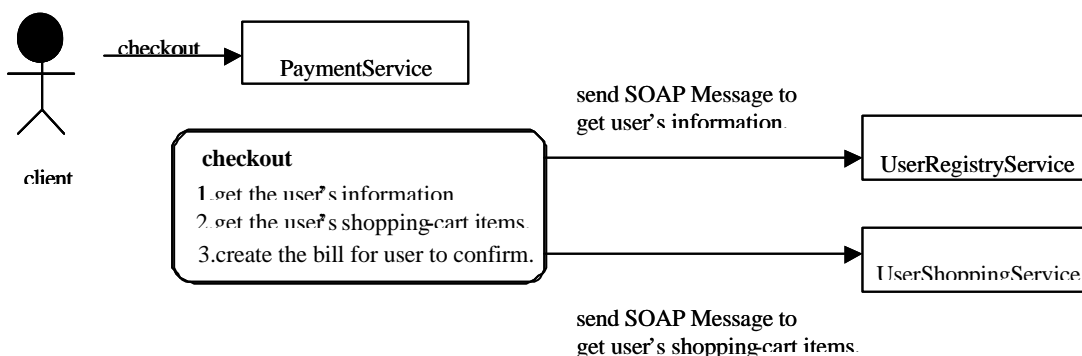


Figure 5.1. Example as a web service oriented application without using SMS.

Figure 5.2 is the same story implemented as an SMS application. `UserRegistryService` and `UserShoppingService` store the information into the shared virtual memory via SMS. Instead of sending SOAP messages to `UserRegistryService` and `UserShoppingService` to request the required information, `PaymentService` can directly access the shared objects defined by `UserRegistryService` and `UserShoppingService`. That is, `PaymentService` can transparently access these objects. It is quite obvious that using SMS application framework to build this example is much more trivial and easier than only using web services since more SOAP messages required to be processed.

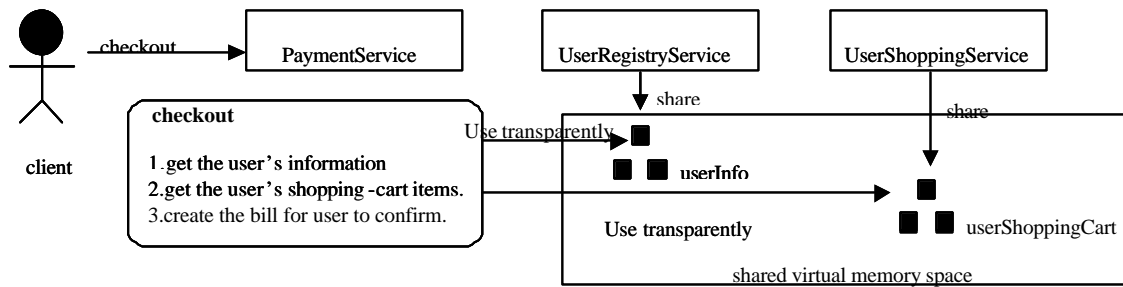


Figure 5.2. Example as an SMS application.

The partial code of the SMS application implementation for Figure 5.2 is listed in Figure 5.3. Figure 5.3 also displays the Java codes such that the pseudo-instructions are translated.

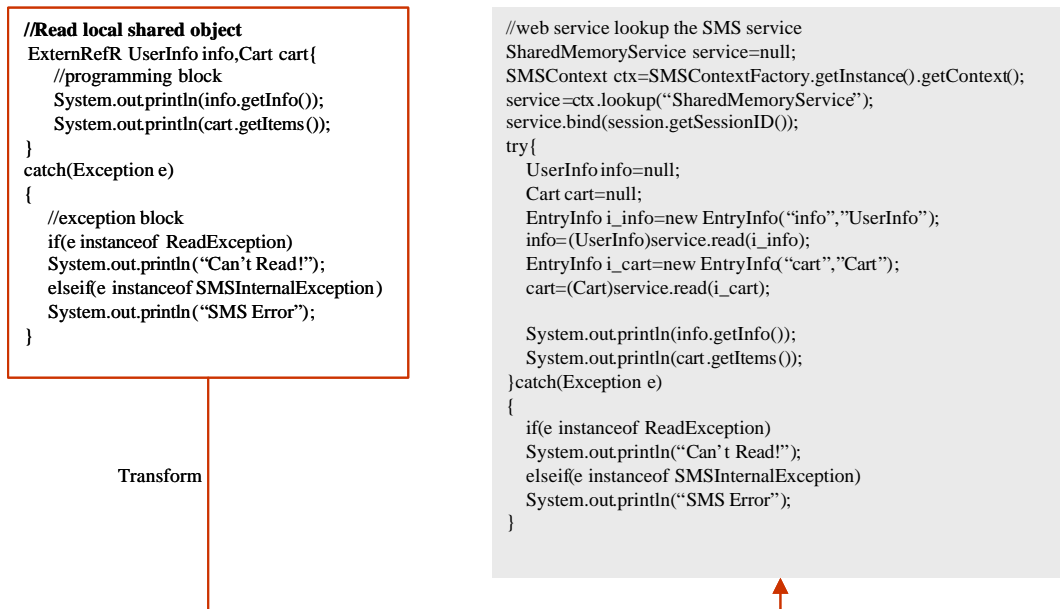


Figure 5.3. Pseudo-instruction and its translation.

6. CONCLUSION

In this paper a new web service application framework has been proposed. With share memory service (SMS), web service oriented applications can be easily designed and implemented. In fact, SMS application framework can dramatically decrease the complexity of sharing the states and objects among web services. SMS also can work with the current web services without too much modification. The performance of our implementation requires further evaluation.

REFERENCES

- [1] C. Adam, "Why Web Services,"
URL:<http://www.webservices.org/index.php/article/articlestatic/75>, 2002.
- [2] Ariba, IBM, Microsoft, "UDDI: White papers,"
URL:<http://www.uddi.org/whitepapers.html>, 2001.
- [3] A. Bosworth, "Developing Web services," *Proceedings of 17th International Conference on Data Engineering*, 2001, pp.477-481.
- [4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", URL:
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508>, 2000.
- [5] E. Castro-Leon, "A perspective on Web Service,"
URL:<http://www.webservices.org/index.php/article/articleprint/113/-1/3/>, 2002.
- [6] E. Christensen, F. Curbera, G Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1,"
URL:<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, 2001.
- [7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE INTERNET COMPUTING*, Vol. 6, Issue 2, 2002, pp.86-93.
- [8] D. Gisolfi, "Web services architect: Part 1 An introduction to dynamic e-business," URL:
<http://www-106.ibm.com/developerworks/webservices/library/ws-arc1/>, 2001.
- [9] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM SYSTEMS JOURNAL*, Vol. 41, NO 2, 2002, pp.170-177.
- [10] IBM Web Services Architecture Team, "Web Service architecture overview The next stage of evolution for e-business,"
URL:<http://www-106.ibm.com/developerworks/webservices/library/w-ovr/>, 2000.
- [11] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0),"
URL:<http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, 2001.

APPENDIX A. SMS API

SOAPSession	SOAPMessageProcessor
+getSessionID():String +getCreatedTime():String +getExpiredTime():String	+getSession(SOAPMessage): SOAPSession +setSession(SOAPMessage,SOAPSession):void
EntryInfo	SharedMemoryService
+EntryInfo(String objName ,String class) +EntryInfo(String objName ,String class, Int memoryManagementType, long leaseTime) +EntryInfo(String objName ,String class, String webServiceName) +EntryInfo(String objName ,String class, String webServiceName, Int memoryManagementType, long leaseTime) +getObjectName():String +getClassName():String +getMemoryManagementType():int +getLeaseTime():long +isGlobal():boolean	+bind(String sessionID):void +read(EntryInfo):Object throws ReadException, SMSInternalException +write(EntryInfo,Object) throws WriteException, SMSInternalException
SMSInternalException	SMSContext
	#init(Hashtable env):void +getEnvironment():Hashtable +lookup(String SMSName): SharedMemoryService +register(String webServiceName):void
ReadException	
WriteException	SMSContextFactory
	+getContext():SMSContext +static getInstance():SMSContextFactory