2004

Design patterns for web service

Kai Qian

Orlando Karam

Jorge Diaz-Herrera

Jigang Liu

Proceedings of the ACIS 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2004)

June 30 – July 2, 2004
People's Palace Hotel, Beijing, China

Editors

Gongzhu Hu, Central Michigan University, USA
Tao Huang, Institute of Software Chinese Academy of Science, China
Xizhen Ni, Institute of Software Chinese Academy of Science, China
Aoying Zhou, Fudan University, China

ISBN: 0–9700776-8-8

A Publication of the International Association for Computer and Information Science (ACIS)
URL: http://acis.lsfk.org E-mail: acis@lsfk.org

Design Patterns for Web Service

Kai Qian
Dept. of Software Engineering
Southern Polytechnic State University, U.S.A

Orlando Karam
Dept. of Computer Science
Southern Polytechnic State University, U.S.A

Jorge Diaz
College of Computing and Information Science
Rochester Institute of Technology, U.S.A

Jigang Liu
Dept. of Computer Science
Metropolitan State University, U.S.A

Abstract

Web service technology is a component-oriented SOAP based interoperable technology widely used in enterprise Application to Application (A2A), Business to Business (B2B), and EAI. Web Service component composite and connection models are the key issues to make Web service successful in the future. This paper presents a number of Web service compositions and connection patterns which will facilitate the Web service design.

Key words: component, design pattern, Web service, proxy, composite, adapter

1. Introduction

Web service is a new paradigm to deliver application services on Web and enables a programmable Web not just an interactive Web. Web service is the third generation in Web evolution after static HTML and interactive Web development such as PERL, ASP, JSP, and others. Web services are a typical black box reusable building block components in distributed computing. There are many other distributed component frameworks available in industry such as CORBA, MS DCOM, .NET, EJB but only Web service provides a real cross-platform, cross programming languages, cross proprietary restriction, and internet firewall friendly solution to interoperable distributed computing. We can simply say that Web services is a self-descriptive on-line distributed component which exposes its services and functionality via its interface on-line and can be published, located and invoked programmatically over Internet. A key point of the importance of Web services is its ability to support programmatic end points for any applications to get services provided by any Web services on line [1-3].

One philosophy behind Web services is to shift distributed software developments from programming to compositions, from coding ground up to building from existing components either by purchasing or by reusing existing components. This is exactly the advantage of Component-Oriented Programming (COP) and Component Based Software Engineering (CBSD). So the main goal of Web services is not only to provide services on Web but also to provide a mechanism to share its service as a building block to be part of other Web services or application programs via its programmatic Web services end points. Web service has shown the potentials to change the ways of enterprise business modeling in distributed system interactions. A large enterprise system can be divided into many relatively independent Web services components, which in turn get services from other Web services components, deployed on Internet.

The key issues to make Web service be a next generation of distributed computing successfully are the integration, interaction, connection, and composition models of the Web services. Many GoF design patterns can be applied to Web service application design. [10] The next sections discuss the composition patterns, proxy patterns, and the other applicable patterns.

2. Design patterns

Design patterns are a set of rules of how to structure code in order to solve a particular class of problems. Many complex software systems needs a number of design patterns working together A design pattern is also a solution to a common software problem which can be applied in many similar cases. A design pattern has its context, and guideline. Comparing to architecture design the detail pattern oriented design is kind of micro-architecture design and architecture design is a macro design which is conducted before detail design. Pattern oriented design is a OO design methodology which reuses the patterns they have been used many times before instead of design from first principles. Since pattern oriented design is the blueprint of software code development so it is a high level programming idiom.

The design patterns can classified into two diminution categories: scopes and usage. The scope of design

patterns can be at class level pattern, object level pattern, and component level pattern. In term of usage concern, there are creatural pattern concerning construction such as factory and singleton, structural patterns concerning composition such as composite and proxy, and behavioral pattern concerning the interaction such as iterator and observer.

2. Composite Patterns for Web Service

2.1 Composite patterns

The composite pattern is one of the most popular patterns used to construct a compound component. In Java API a container component class is a sub class of AWT component class. There are many similar AWT component classes in this part-whole hierarchy. A composite itself may be a component. A component may be a composite component or primitive component. The container component object is a compound component, which can add another component that in tern can be another container or a primitive component. This concept can be applied to component design beyond class design and object design. There are two alternatives in composite patterns. A component in a composite component may know which component it belongs to. This kind of component can be accessed directly from the client of the composite where the component resides. It is called a containment composition. The other type of component may not know itself which composite it belongs to. It can not be directly accessed from outside. It is called aggregation composition. Figure 1 below shows a general UML of a composite pattern.

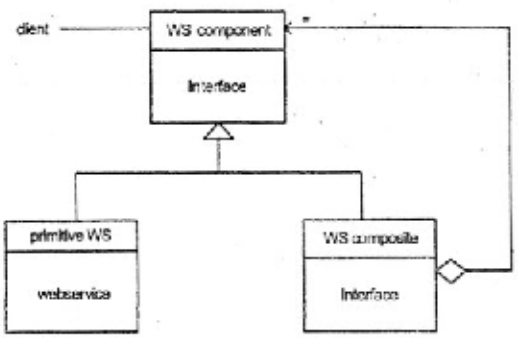


Figure 1 Composite pattern

2.2 Composite patterns in Web service

We have seen Web service applications that a client simply gets a service from a primitive Web service. Actually we can also compose a new Web service from many existing Web services. In order to make Web service be truly useful in A2A, B2B and EAI, there must be a way to combine and coordinate collections of Web services, to glue them together, so that they can be integrated to support a large scale real world business enterprise application. A new compound Web service, which has its own identification, is called composite Web service and is different from primitive Web service. One Web service can be composed into many other Web services and one Web service may consist of many Web services.

There is a many-to-many relationship between a Web service and Web service composition. There are two composed Web services in Figure 2. Both of them use ws3 as a part of their Web services. We can see that a composite Web service is a Web service which combines or glues multiple Web services into a new large Web service.

Web Service Flow Language (WSFL) [1] [2] may be used to determine the sequence of sub process of Web services that form a business process and flow of information.

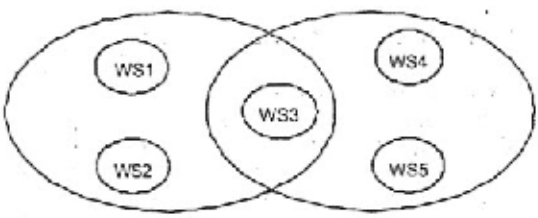


Figure 2 Web service composite

A Web service composite can be implemented by a containment construction, in which the outer Web service holds a reference to one or many inner Web services and outer Web service forwards a Web service request to a inner Web service. The end point of inner Web service is behind the scene. An example of containment construction is the Web service end point for stateless session EJB shown in Figure 3.

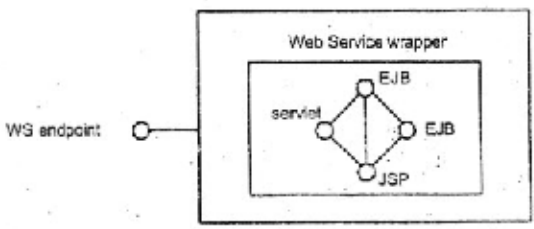


Figure 3 Web service composite (containment) for a J2EE enterprise application

A Web service composite can also be implemented by aggregation construction, in which an inner Web service end point reference is handed out directly to the outer Web service client. Figure 4 shows both of these compositions are hardwired at the design time.

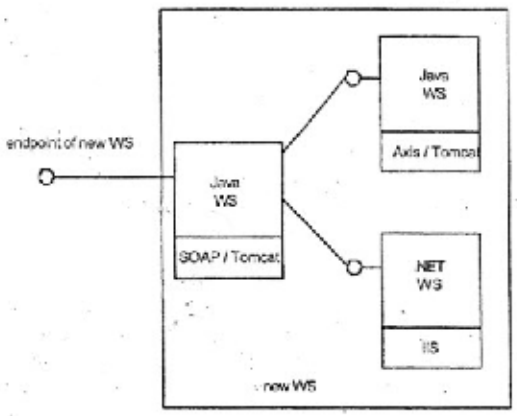


Figure 4 Web service composite (aggregation) of Web service composition

3. Web service interaction models

Web service invocation is done in either synchronous or asynchronous mode. In the synchronous interaction, a client sends a Web service request and halts its operation while waiting for a response. If the service takes a tremendous computation time to complete or the service is not needed until some events are triggered an asynchronous interaction must be used instead. The synchronous interaction is the default interaction in Web service operations. Another reason is in some cases there is no backward channel available for the responses to come back synchronously such as email response to a HTTP request.

Asynchronous message-based operation is a single message passing operation, which can be one-way incoming notification and one-way outgoing notification. One-way message passing operation never expects any response right away. The message may be a signal, a notification, or data to be processed by the target of the message. The two-way out/in request/response operation is a RPC-based operation with combination of incoming and outgoing SOAP message. The source of the message sends out an outgoing SOAP message and expects to get an incoming response SOAP message back right away. In the in/out (solicit/response) mode the target of message gets incoming SOAP message and responds an outgoing SOAP message. Obviously, they are synchronous operations. Figure 5 shows these four types of interactions between any two Web service components or between Web service clients (may be a web service itself) and a Web service component.

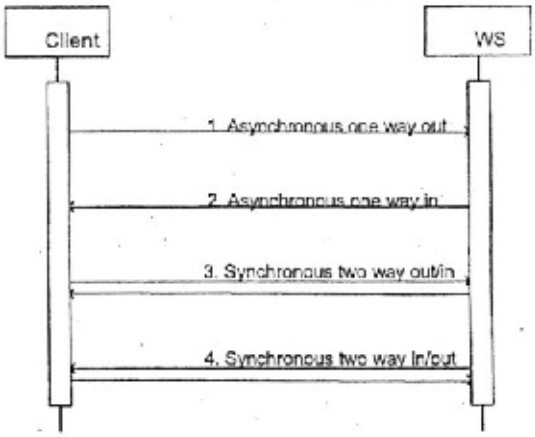
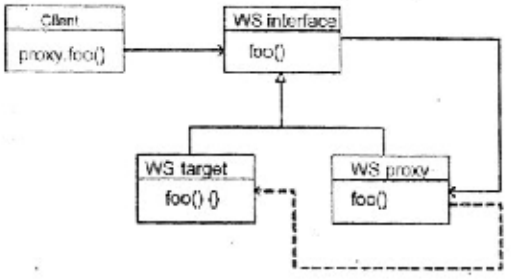


Figure 5 WSDL supported interaction pattern

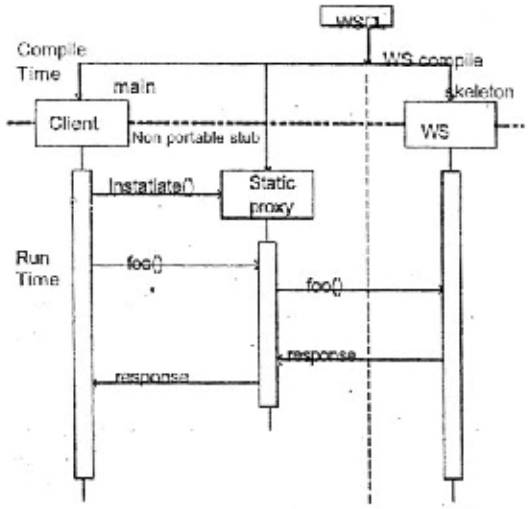
4. Proxy patterns in Web Services

4.1. Proxy pattern

The proxy pattern is often used to represent a remote object, or component that is complex, or time consuming to create, or to enforce access control to that object. It passes on method calls from clients to an actual target. It is a surrogate placeholder of real object or component. A proxy acts on the behalf of another object or component. A proxy does not provide service directly but calls methods of the actual object or component, which provides the needed services. In a distributed computing environment a proxy marshals the method call invocation to serialize it to a remote service and unmarshals the response back from the remote server. The following Figure 6 (a) shows the concepts of a proxy. We focus on the proxy class on client side at this time. The proxy may reside at server side too. There may also be many types of proxies such as remote proxy, virtual proxy, protection proxy, dispatcher proxy, and each proxy can associate with some other design patterns such as adapter, bridge, dispatcher, and factory. In the following section we focus on the proxy patterns used in Web services. The Part (b) of the figure shows a Web service static proxy pattern, which will be discussed later.



a) Proxy pattern



(b) Web service static proxy

Figure 6 Proxy patterns

4.2 Web service static stub proxy

A static stub proxy is created at the compilation time and all related information about Web service including URL, WSDL, service name, port, and even the implementation of the Web service component are known. The static proxy is a representative of the remote Web service component and knows how to handles the request to the server component and response back from the server component. It is very efficient but it not portable at all because it is implementation specific. The figure 6 (b) depicts the concept of a Web service static stub proxy class. The part above dash line indicates the compilation time and the part below indicates the run time. The part on the left side of dash line describe the client and the right side describes the Web service .

4.3 Web service dynamic proxy

In some cases the WSDL of a Web service may not available at the compilation time or it got changed afterward. A dynamic proxy is generated at run time instead of compilation time to avoid such problems. After collecting all necessary information about WSDL and looking for WSDL to get detail information to generate a dynamic proxy which is more portable and less efficient in term of performance. In this case the client still needs to know all details of signatures of the remote method. It is a generic service proxy which supports multiple implementation options. In some scenario, one Web service interface may have multiple implementations or bindings. The client may also want to use different implementations in term of protocols for same interface specification. Figure 7 shows the Web service dynamic proxy pattern where left part of dash line is client and right side of dash line is Web service, and the part above the dash line indicates the compilation time and the part below the dash line indicates the run time.

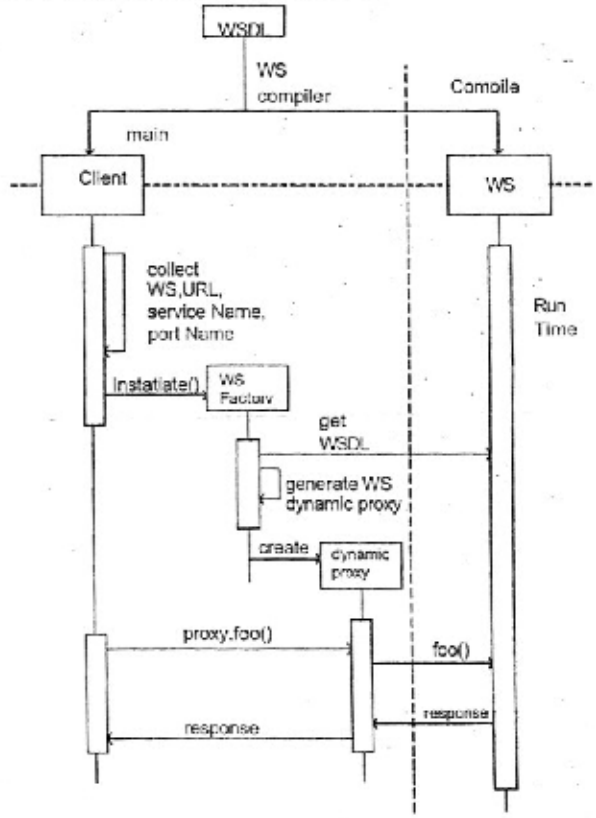


Figure 7 Web service dynamic proxy

4.4 Web service Dynamic Invocation Interface (DII) proxy

The DII proxy is even more portable than dynamic proxy where the client can call remote methods of a Web service even if the signature of the remote method or name of the service are unknown until run time.

At the time Call object proxy is created it is not yet associated with any operation. Next, all operation properties including the Web service end point address, method name, arguments, operation style such as "rpc" can be set or added to the Call object. For example a Web service client application can provide all needed information via command line arguments without knowing anything about target Web service in advance. The trade-off of it is the performance again. This DII model is very similar to DII in CORBA.[12]

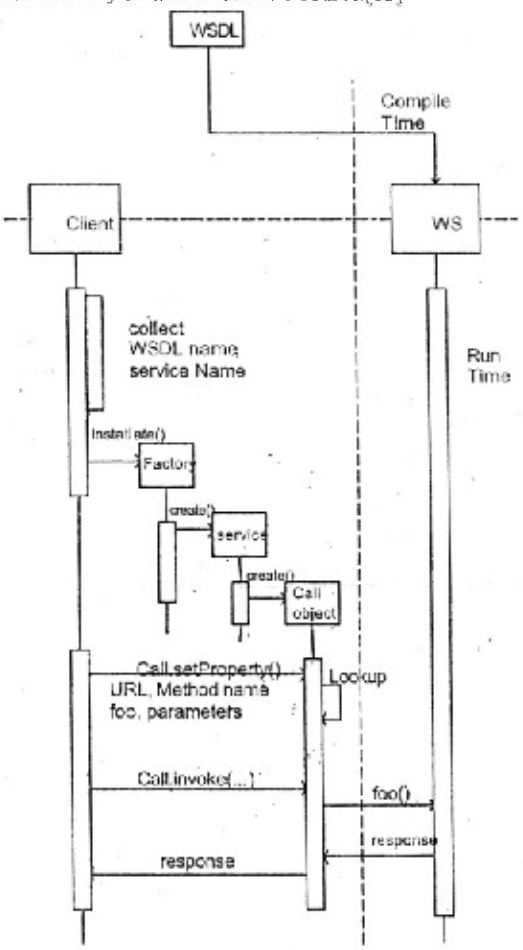


Figure 8 Dynamic Invocation Interface (DII)

4.5 Web service two way asynchronous virtual proxy

An asynchronous communication does not expect to get response immediately instead it can continue its work without being idle in waiting for a notification from the server. There are many way to implement such as callback method which passes in a callback method as an argument of remote method invocation to a server component and lets server component calls back this callback method when it is ready. Other option can be implemented in a half asynchronous mode. The client still needs to poll expected results from a queue where the server puts the response. Figure 8 shows the asynchronous invocation pattern for a Web service client to get a service provided by a Web service at a server site.

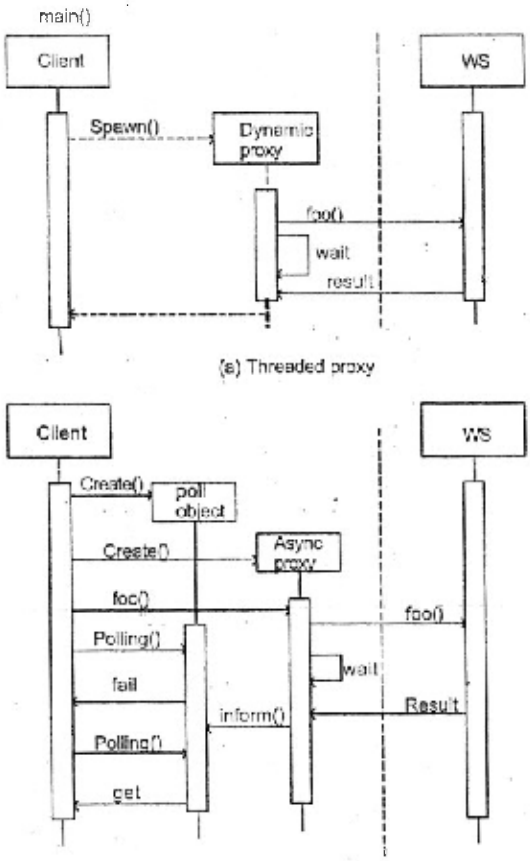


Figure 9 Asynchronous two way poll proxy

The part (a) of Figure 9 depicts a threaded proxy pattern where client pawns a thread to take care of invocation to

remote methods and notifications back from Web service. The part (b) of the figure depicts a half asynchronous polling pattern where client and server are decoupled by a message queue. The client places a request onto the queue and server picks up the request from the queue and then server enter the results onto the queue and lets client to poll the result asynchronously.

We assume that the client is not a Web service component in this section. Figure 10 shows a scenario that both of client and server are Web services.

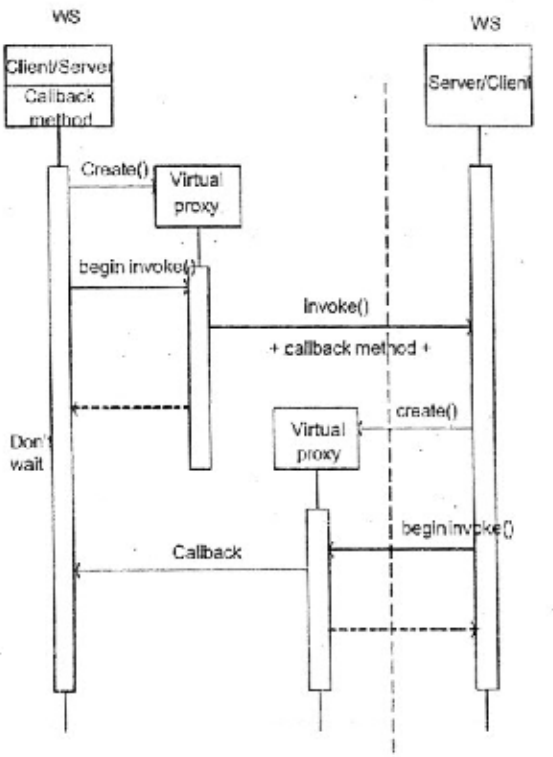


Figure 10 Web service callback proxies

5. Event-based patterns in Web service

The event-based asynchronous communication between event source component and even target component is very popular in event-based applications and also widely used in distributed computing such as CORBA Component Model (CCM). Figure 11 shows the conceptual event-based pattern where the event target must register with event source and specify the event handler which takes an event type object. Whenever such event is triggered the event source will scan the vector where all interested targets are registered and the event handlers of corresponding targets will be invoked. There may be many listeners register with a single event source. The trigger of the event may be a third party.

The WSDL1.0 spec does not support event interaction model. Some Web service vendors are implementing event-based notification which is similar to a publish/subscribe pattern. The solutions are either WSDL is enhanced to support event type or two way asynchronous modes, or Web service implementation API supports the event handling mechanisms.

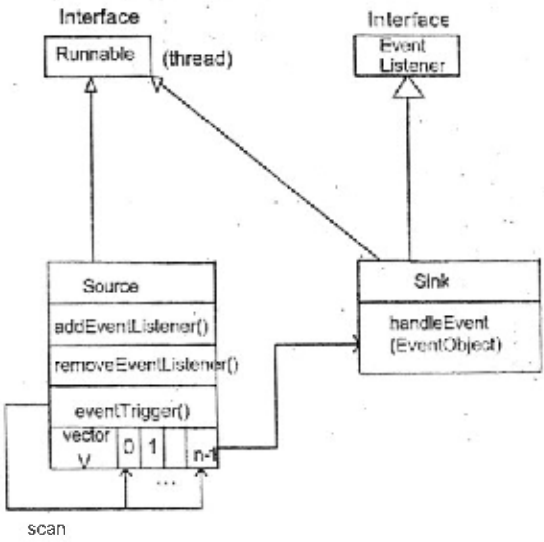


Figure 11 Event-based patterns

6. Adapter pattern in Web service

The adapter pattern is used to convert one interface to another interface in different infrastructure. The idea is to generate a class, object, or component that has the desired interface and make it communicate with the class, object, or component that has a different interface. Figure 12 shows the concept of the adapter pattern at class levels. We can derive a new class from an adaptee class and add new methods to make the new derived class match the target one.

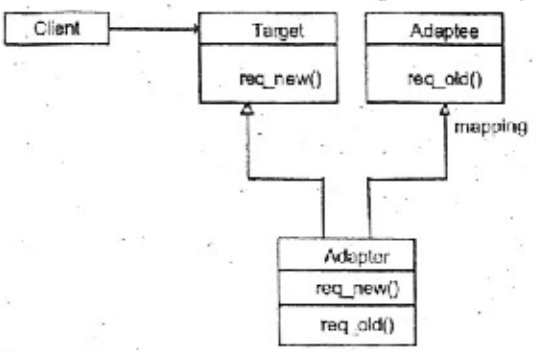


Figure 12 Gateway/Adapter

Of course we can use create a new class which includes the adaptee class and create a method to translate the call within the new one. This pattern can be used for object adapter pattern. A Web service needs to work with other existing application in different technology, protocol, and infrastructure. There is a need to generate an adapter either at client side or server side just like the figure 13 shows. Even there is a need to have an adapter between two Web service components because of different implementations from different Web service vendors.

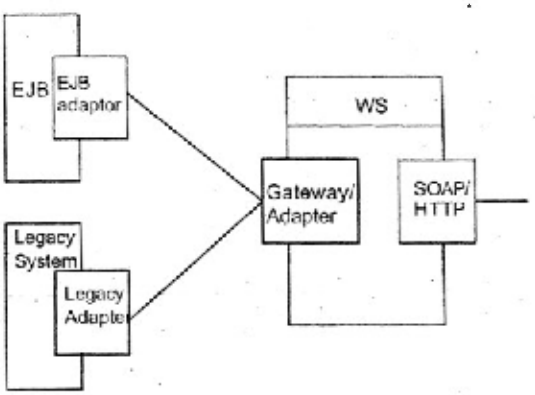


Figure 13 Gateway/Adapter

7. Conclusion

In this paper we present a number of applicable design patterns for Web service constructions many other design patterns can also be applied in Web service design together such as factory, thread-safe, dispatcher, and proxy patterns. We also propose a CCM type event-based design pattern which will enhance the Web service applications. The design pattern is one of the very effective methodologies applied to the detail design phase of Web service engineering.

8. References

[1] F. Curbera, I Silva-Lepe, and S Weerawarana, "On the Integration of Heterogeneous Web Service Partners", 2001
[2] B Srivastava, and J. Koehler, "Web Service Composition – Current Solutions and Open Problems", 2002
[3] F. Curbera, N Mukni, and S Weerawarana, "On the Emergence of a Web Services Component Model", 2001
[4] H. Deitel, Java Web Services for Experienced Programmers, Prentice Hall, 2003
[5] H. Deitel, Web Services, a technical Introduction, Prentice Hall, 2003
[6] M. Clark, P. Fletcher, J. J. Hanson, R. Irani, M. Waterhouse, and J. Thelin, "Web Services Business Strategies and Architectures", 2002
[7] http://msdn.microsoft.com/library/en-us/cptutorials/html
[8] http://www.componentsource.com
[9] http://www.codeproject.com
[10] Gamma, Helm, Johnson, Vissides, "Design patterns", Addison-Wesley, 1998
[11] Mark, Grand, "Patterns in Java", John Wiley & Sons, 1996
[12] T. Mowbray, and R. Malveau, "CORBA Design Patterns", John Wiley & Sons, 1997