Stateful Web Services Using WSE

MARTIN VITEK, STANISLAV UCHYTIL, IVO HERMAN
Department of Telecommunications
Brno University of Technology
Purkynova Street 118, 602 00 Brno
CZECH REPUBLIC

Abstract: Web services represent a technology providing data exchange in the Internet distributed environment through remote procedure calls (RPC). Web services are characterized by their easy and straightforward definition, to which the type of web service communication is related. It is commonly stateless. When a stateful communication is needed, it must be provided by the hosting web service environment. This article deals with some of available techniques used to ensure this type of communication, together with the design of solution for a stateful web service placed on an IIS server using the WSE library.

Key-Words: web service, IIS, stateful, WSE, SOAP.

1 Introduction

A web service is defined as an object that is able to generate, receive and process messages [1]. In practice the SOAP [2] (Simple Object Access Protocol) protocol is used for communication with the web service. The web service is usually placed on a web server that commonly communicates via the http protocol. In this case the SOAP messages are encapsulated inside the http messages (see fig. 1).

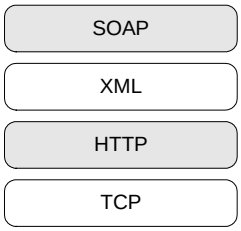

Fig. 1 Web service protocol set.

The usage of http protocol also determines the web service communication method. It is of the request-response type and it has a stateless character because the web service and SOAP specifications themselves do not solve this issue. If there is a requirement for a stateful communication with the web service, a suitable solution must be found in hosting web service environment, usually by a web or an application server on which the service is located, such as in [3] and [4].

As web services belong to one of the rapidly expanding internet technology, it can be found a few methods how to ensure holding web service state data between individual client requests, such as [8] or [14]. But among the most familiar techniques for stateful web service a communication using cookies belongs that is described in the following section. But first we take a short look at the standard stateless web service communication.

In section 3 a new approach how to achieve stateful communication with a web service is presented. This technique is based on the WSE library and uses new designed attributes to simplify and automate the development of stateful web services as in the case of stateless services.

2 Stateless web service communication

The web service is usually implemented in the form of a standard class known from object-oriented programming (OOP). During the stateless communication with the web service each client request is processed by a newly created instance of the web service class (see fig. 2).

When a new instance is created all its local data variables are set to their default values. If the variables contain information from a hard-to-access data source, its repeated obtaining (initialization) with every client request means to lengthen the whole request process time. The time to process only the request itself is usually significantly lower than the time needed for the local variables initialization; the web service communication becomes inefficient.

The hard-to-access data source can be, for example, data from a database or a reference to an object situated in a different process.

The above disadvantages can be solved by stateful communication with a web service that enables holding inner web service local (stateful) data between individual client calls. Initialization of

stateful data takes place only during the first client request for the web service.

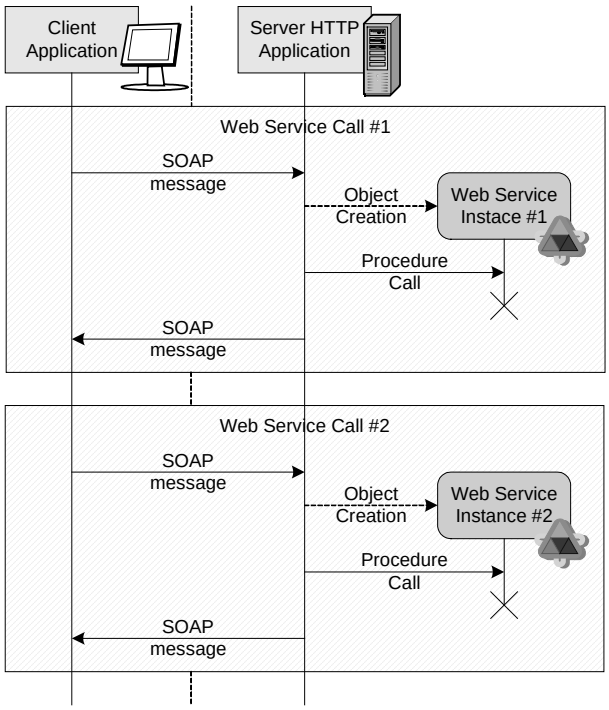

Fig. 2 Individual request processing by new web service instance.

2.1 Stateful communication using cookies

One of the available methods how the stateful communication with the web service can be realized is the usage of cookies. Cookies are small data files that are sent together with the http message (see fig. 3).

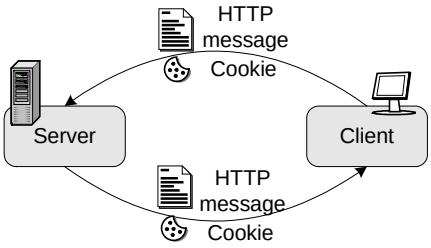

Fig.3 Communication with the help of cookies.

There are two basic techniques of stateful communication with cookies:

1) The state variables are stored directly inside the cookies. For example, this method can be used with web service implementation on the IIS (Internet Information Server) server, where the cookies are accessible inside the web service through the Sessions object [5]. This is suitable especially for smaller data elements because stateful data increases amount data transmitted between the client and the server. If it is sensitive information, it must be protected in an appropriate way [6].

2) Only the http session identifier is held in the cookies while the state variables are stored somewhere on the server. The identifier is for unique determination whose request belongs to the given http session. This type of stateful communication can be found on the IBM Websphere server [7], for example.

The advantage of using cookies is that it is well-known communication method on the Internet. The cookies also solve the problem of stateful data lifetime because it can be related to the cookies lifetime. A disadvantage is that the communication with cookies is not always permitted on the client side (because of security reasons, for example). Then this method of stateful data transmission cannot be used.

2.2 Web Services Extensions

The basic web service and the SOAP protocol specification would be certainly not enough for real practical application of this technology. One of the web service properties is its extensibility. Web service protocol extensions are being designed for more complicated communication techniques, for example WS-Security solves the question of secure communication, WS-Reliable Messaging is for reliable communication, and WS-Transaction is used for transaction management, etc.

One of web service extensions is the WS-Resources specification for managing the access to stateful data resources through web services [8]. With the help of this specification, storing of state in the web service can be realized. But if there is only a requirement for simple holding data between individual client requests, the protocol extensions are unnecessarily robust, i.e. they require extra knowledge and implementation that a web services creator must know.

3 Stateful communication through WSE

The disadvantage of transferring additional cookies together with the http message can be removed by inserting the information form cookies directly in the SOAP message that is encapsulated in the http message. From two above solutions of stateful communication using cookies it is more advantageous to transfer only the session identifier. This solution requires:

1) putting extra information in the SOAP message.
2) processing this information on the server side and restoring the respective stateful data.

If the web service is placed on the IIS Server then these steps can be easily realized through the WSE library. This library is also used in proposed solution for stateful communication with the web service using the WSE library.

3.1 The WSE library

Web Service Enhancement (WSE) is a new library of classes that enables building web services using the newest protocols, such as WS-Security, WS-Routing, DIME, WS-Attachments, etc. WSE is completely integrated into the ASP.NET and offers a possibility how to easily extend web service functionality on the IIS (Internet Information Server) server [9].

The WSE principle is based on SOAP message preprocessing just before the message is processed in a standard way. The preprocessing is executed in filters [10] (see fig. 4).

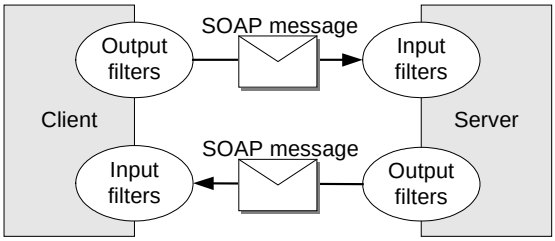

Fig. 4 Input and output WSE filters.

If the WSE library is used, then all outgoing messages on the client side go through a set of output filters, while on the server side all incoming messages pass through a set of input filters. In the server-client direction the message goes successively through the server output filters and the client input filters on the client side.

A great contribution of the WSE library can be seen in the ability to create own filters [11] by means of which message preprocessing can be realized on both (client and server) sides. On the server side the message is preprocessed in the input filters just before the web service method is called, while output filters enable modifying the message just before it is sent to the network. As a matter of course there is the possibility of data exchange between a filter and a web service object through the SOAP context of the message.

The lifetime of filter is interesting. Filters are not created and destroyed during every request for the web service. They are created only when an http application is started and can be destroyed when a web service has not been used for a long time or there is lack of memory on the web server. So one filter processes several requests from one or more clients. The filter keeps its state between individual calls. This property can be used to realize stateful communication with a web service.

3.2 Stateful web service using WSE library

Stateful communication based on WSE can run according to the conception depicted in fig. 2.

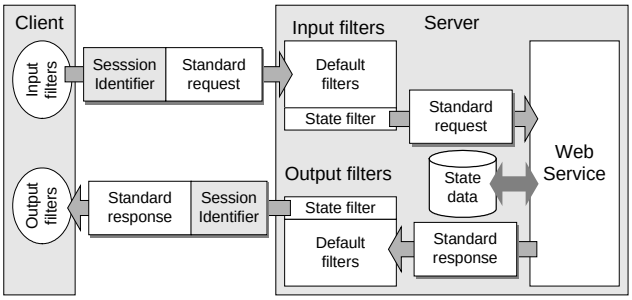

Fig. 5 Stateful communication using the WSE filters.

This conception is based on SOAP messages being preprocessed by a customized WSE filter (state filter) and an additive identifier (state session identifier) in the SOAP message header.

Since the WSE filter holds its state even between incoming requests, it can be used to store client-specific information (stateful data) that needs to be kept between individual web service calls. Each client request contains the unique state session identifier, so the WSE state filter can find out which request belongs to which client-web service connection (session). The filter then contains a list of value pairs including the state identifier and stateful data.

During an incoming request the input state filter finds out the state identifier from the message header, and based on this identifier the filter looks for relevant stateful data in its list. The filter then sets this data on a newly created web service object using the SOAP context. In this way it is guaranteed that at the time of web service method calling, the client-specific data is set as in the previous request. The web service object then appears to keep its state between individual message calls.

When the web service method call finishes, it is necessary to ensure the storing of stateful data before the web service object is destroyed. This activity is provided by the output state filter, which updates the respective record in its stateful data list

and stores the state identifier in the outgoing SOAP message (see fig. 6).

From the client viewpoint the web service appears to store its state between individual calls. For the whole system to operate correctly, it is necessary to place an input and an output state filter also on the client side. The client input filter serves to parse the state identifier from the incoming message, while the output filter inserts the identifier in the message conversely.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap= ... >
  <soap:Header>
    ...
    <stateSession xmlns=...>
       <Id>2004-11-29T21:12:10N0</Id>
       <ExpireTimeout>
          00:02:00
       </ExpireTimeout>
       ...
    </stateSession>
    ...
  </soap:Header>

  <soap:Body>
    <SetValue xmlns=...>
      <theValue>7</theValue>
    </SetValue>
  </soap:Body>

</soap:Envelope>
```

Fig. 6 Reduced content of the SOAP message containing the state session header.

The flow of SOAP requests and responses through the state session filters on the client and server side is depicted in figure 7.

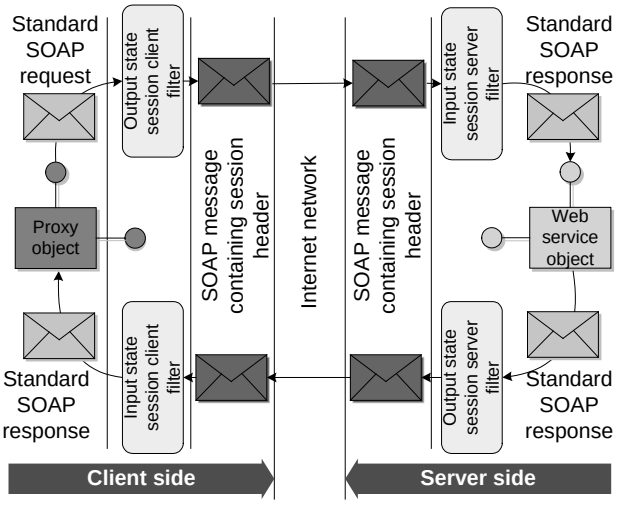

Fig. 7 Block scheme of message flow through the WSE state session filters on the client and server side.

3.3 Session attributes

On the .NET Framework platform, attributes can be added to data elements during their declaration. By this method extra information can be assigned to data element indicating how to operate with the element [12].

To simplify and automate stateful web service development the *Store* attribute was designed. This attribute says that the data element defined with it has to be saved in data storage, so it can be later restored. The definition of such a data element is simple, as shown in fig. 8.

```
[Store()]
private int _counter;
```

Fig. 8 The definition of data element together with *Store* attribute.

Futhermore the *ServiceState* attribute is defined. This attribute can be applied on the service web method (the service public method that is accessible through a network). Through the *ServiceState* attribute we can control the state of session when the method is called. We can set whether the called method uses, creates or destroys the session ensuring stateful communication. The usage of the *ServiceState* attribute can be seen in Figure 9.

3.4 Stateful web service implementation

The implementation of ASP.NET stateful web service using the WSE library, state filters, *Store* and *ServiceState* attribute, is almost same as the implementation of standard stateless service. The first step is to integrate the WSE library together with a state filter on both the server and client side. This step can be done through the xml configuration files or programmatically. This procedure can be found in the WSE library documentation [11]. What then follows is the development of the web service itself, its class respectively.

The development of stateful web service class is the same as the stateless service class but data variables whose content should be held between individual client calls are defined with the *Store* attribute. Furthermore the methods using state session are defined with the *ServiceState* attribute. The definition of such stateful web service class is depicted in figure 8. It is the simple web service holding state of counter (the *_value* variable) among individual client calls.

The client code of this web service can be as given in figure 10.

```
// Counter service.
public class RemoteCounterService :
    StateFulWebService
{
    [Store()]
    private int _value; // counter value –
                        // state variable

    // ctor
    public Counter () : base ()
    {
    ...
    }

    ...

    // Sets the counter value.
    [WebMethod]
    [ServiceState(
      ServiceStateUsage.Apply)]
    public void SetValue (int theValue)
    {
        _value = theValue;
    }

    // Gets the counter value.
    [WebMethod]
    [ServiceState(
       ServiceStateUsage.Apply)]
    public int GetValue ()
    {
        return _value;
    }

    // Adds the value.
    [WebMethod]
    [ServiceState(
       ServiceStateUsage.Apply)]
    public void AddValue (int theValue)
    {
        _value += theValue;
    }
}
```

Fig. 9 The code example of stateful web service written in the C# language.

```
RemoteCounterService myService =
    new RemoteCounterService ();

// set counter, ie. calling web method
// that is defined as method for applying
// service state
myService.SetValue (7);
// add value, ie. calling web method
// that is defined as method for applying
// service state
myService.AddValue (4);
// get value – returns number 11, ie.
// the web service holds its state
// between client request
int counter = myService.GetValue();
```

Fig. 10 The example of client code using the stateful web service defined in figure 8.

4. Conclusion

Stateful communication with the web service is usually based on application of cookies. The disadvantage of this solution is the necessary support of cookies processing on the client side, which need not always to be fulfilled. The described solution uses the possibility of extending the SOAP message by a new header describing the created session for stateful communication and by the WSE library for processing this header, storing and restoring stateful data of the service. The solution is characterized by its simplicity and the possibility of setting which data will be stored through the *Store* attribute. The only necessary steps are to integrate the WSE library and to add filters providing stateful communication. The development and communication with such service seems to be same as the communication with a standard stateless web service.

The importance of the web service will increase in future. This is also evidenced by efforts aimed at creating a unified interface for communication and access to remote services via a technology called Indigo in which the support of stateful communication with the web service should be already implemented [13].

Acknowledgement

This paper has been prepared with the support of the Grant Agency of the Czech Republic within grant No. 102/03/1033 and 1645/2004/G1: "Conception of modules for web service stateful communication".

References

[1] Tidwell D., Snell J., Kulchenko P., *Programming Web Services with SOAP*, O'Reilly 2001, ISBN: 0-596-00095-2.

[2] World Wide Web Consortium, *SOAP specifications*, available at http://www.w3.org/TR/soap/.

[3] *The MS.NETGrid Project*, available at http://www.epcc.ed.ac.uk/~ogsanet/

[4] Java Developer Center, *How-to: create Stateful Web Service from Basic Java Class*, available at www.oracle.com/technology/tech/java/oc4j/1003/how_to/how-to-ws-basic-stateful.html

[5] Powell M., *Using ASP.NET Session State in a Web Service*, available at http://msdn.microsoft.com.

[6] Prosise J., *Foiling Session Hijacking Attempts*, available at http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/

[7] Hines B., Alcott T., Barcia R., Batzum K., *IBM WebSphere Session Management*, available at http://www.informit.com/articles/article.asp

[8] Czajkowski K., Ferguson F. D., et al, *The WS-Resource Framework*, available at http://schiermike.sc.funpic.de/published/PDFs/wsrf-arbeit.pdf
[9] Ewald T., *Programming with web service enhancements 2.0*, available at http://msdn.microsoft.com/webservices/building/wse/.
[10] Ewald T., *Inside the Web Services Enhancements Pipeline*, available at http://msdn.microsoft.com/webservices/building/wse/.
[11] WSE library documentation, *Creating Custom Filters with WSE*, available at http://msdn.micorosft.com.
[12] .NET Framework Developer's Guide, *Writing Custom Attributes*, available at http://msdn.microsoft.com
[13] Januszewski K., *Writing Asynchronous, Bidirectional, Stateful, Reliable Web Services with Indigo*, available at http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo/default.aspx.
[14] Web Services Addressing Working Group, *Web Services Addressing 1.0 - Core*, available at http://w3.org/2002/ws/addr/.