Processing ASP.Net Web Services Using Generic Delegation Approach

Vilakshan Saxena, Harshit Santosh and Chittaranjan Pradhan

Abstract For the electronic business applications, web services are usually considered as the design models. Here, our aim is to design an efficient model to deal with both the distributed applications and cooperative applications. In both the cases when it comes to implementation of web services in respective applications, the consumer (developer) has to put an effort to manually provide the reference of the respective web service through a specific set of steps depending upon the target IDE. But what if we have a technique to perform the above mentioned approach in a dynamic and generic fashion without manually adding the web reference for any web service. In this paper, we will represent an efficient approach for interacting with any web service irrespective of its syntax (WSDL) and semantics without adding its web reference. Through this approach the consumer of the web service can access the respective web service dynamically by just mentioning its URL in his/her code and through a little object oriented methodology. Our approach is based on accessing the particular web service by automatic generation of proxy class, delegation, dynamic data type handling through reflections API and producing the desired output in a generic fashion.

Keywords Web services · Reflections API · Proxy class · Web service authentication

Vilakshan Saxena (✉) · Chittaranjan Pradhan
KIIT University, Bhubaneswar, India
e-mail: vilakshankiit@gmail.com

Chittaranjan Pradhan
e-mail: chitaprakash@gmail.com

Harshit Santosh
MIT, Manipal, India
e-mail: harshitsantosh@gmail.com

J.K. Mandal et al. (eds.), *Proceedings of the First International Conference on Intelligent Computing and Communication*, Advances in Intelligent Systems and Computing 458, DOI 10.1007/978-981-10-2035-3_16

1 Introduction

Web service is a technique in which one electronic device can communicate to another over a network. It is provided as a software functionality at web node with the services available as utility computing. It describes a methodology of integrating web-based applications using the internet protocol. For data tagging, XML standard is used [1]. To transfer data, SOAP standard is used. Similarly, for the description of the services, WSDL can be used. To get the available services, UDDI standard is used [2].

Using the web service, two software packages exchange data between themselves over the internet. The requesting software system is called as service requester and the processing software system is considered as the service provider [3, 4]. We need a generic software system which is free from any specific programming language. Since XML tags are interpreted by almost all software, it can be used as the data exchange tags in web services. The set of rules used for data communication are defined in Web Services Description Language (WSDL) file [5]. Universal Description, Discovery and Integration (UDDI) is used as the directory of data types and compatible software systems. Once the service provider validates the service requester through WSDL file, Simple Object Access Protocol (SOAP) can be used for data transfer [6, 7].

In the proposed approach, we access the particular web service by automatic generation of proxy class, delegation, dynamic data type handling through Reflections API and producing the desired output in a generic fashion.

2 Related Work

In 2005, John B. Oladosu et al. have done a study on web services; which includes the advantages of the web services over the previous services such as CORBA, COM etc. The study also includes the different components of web service along with the different standards. The application of web services in the e-health domain has also been discussed in the paper [1].

In 2009 C. Boutrous Saab et al. developed a technique on processing of web services by focusing on the two basic features of web service. The first feature deals with the interaction problem provided in the interaction protocol. The second feature deals with the design process of a web service [3].

3 Proxy Class

A client can communicate with the web service using SOAP message. This message encapsulates the input parameters and output parameters in the XML form. A proxy class maps the input and output parameters to the XML elements and sends

the message over the network. Using this technique, the proxy class frees the consumer from the communication with web services at SOAP level and permits the consumer to invoke the web services from the development environments supporting web service proxies and SOAP. The proxy class can be added to the development environment/project in Microsoft.NET Framework: (i) using WSDL tool and/or (ii) adding web reference in Microsoft Visual Studio. The proxy classes can also be processed in the similar way in J2EE Framework.

3.1 Using WSDL Tool

The Web Services Description Language (WSDL) tool of. NET Framework SDK allows us to create and use web service proxies. This tool accepts a set of arguments to generate a proxy. When the proxy is generated, the proxy class can be compiled to an assembly file and added as a project item [5, 7].

3.2 Adding Web Reference in Microsoft Visual Studio

When a project needs to consume one or more number of web services, web reference methodology is used. Once the web reference is added to the respective project, then it can be used easily to access corresponding web services over the net using object oriented methodology [8].

4 C# and VB Reflection API

The reflection API is used to create an instance of a type, bind the type to or extract the type from an existing object, and access the methods and its characteristics [9]. The attributes can also be invoked using the reflection. The reflection is used due to these:

- When the attributes of program's metadata needs to be accessed.
- When the types in an assembly needs to be examined and instantiated.
- When new types are built using System.Reflection.Emit class.
- When late binding needs to be performed at run time.

5 NET Web Service Authentication

Web service authentication is performed in order to secure the web service from been hijacked by third party contenders as it travels from source to destination over the network. Web service authentication can be performed using many ways [10]. In .NET framework authentication can be performed using many ways as follows:

- Basic authentication using web service Credentials property.
- Custom Authentication using several security extensions.
- Simple authentication of web services using SOAP and authorization headers which focus on object oriented methodology of creation SOAP header classes and objects and usage of individual member variables for username and password, in which password can be clear text or MD5 hashed.

6 Proposed Algorithm

Here, we propose an algorithm representing an interface for catering of any .NET web services using generic delegation approach. Unlike creating separate classes and objects for each of the web references corresponding with different web services, this interface acts as a delegate for any target web service resulting in creation of one point two way communication for separate web references. Let WS.asmx is the target web service that we want to access with this interface. The steps of the algorithm are as follows:

a. Receive the input parameters regarding the target web service such as

   i. Web Service URL. (www.abc.com/WS.asmx?wsdl).
   ii. Service Name (Can be retrieved from the WSDL file of the web service).
   iii. Method Name (Desired function name to be called).
   iv. Input parameters.
   v. Authentication Class Name.
   vi. Username.
   vii. Password.

   The last three parameters are optional and are required only if there is authentication header associated with the web service with a username and a password.

b. Create an instance of the proxy class and associate it with the provided web service URL as mentioned in (6.a.i) and store the compiled results in 'Compiler Results' class.
c. Using the object mentioned in (6.b), dynamically create instance of the target 'Service Name' as mentioned in (6.a.ii).
d. Once the instance of the target service name is created extract the structure of the method (function) as provided by Method Name parameter as mentioned in (6.a.iii).
e. Using the Reflection API extract the target data type of the Input parameters provided. (As mentioned in (6.a.iv)) as they are boxed into a generic data type. *Note* Here we don't know the data type of the input parameters provided to the interface due to which we have used the Reflection API to detect the data type of the parameters dynamically.

f. After extracting the target data type (As mentioned in (6.e)) extract all the input parameters from provided Input parameters and store it in an array.
g. Search the extracted method structure (As mentioned in (6.d)) among all the methods present in the target web service (As web service can be a collection of many methods) until there is a proper match.
h. Once the target method is found and analyzed process the generic input parameters (As extracted in (6.e)) and store there values in the desired input parameters of the target method to be called.
i. As the data type of the input parameters is determined dynamically, so the provided input parameters are sequentially captured according to the data type in the target method parameters.
j. Once all the parameters are mapped (As mentioned in (6.h)), call the target method.
k. Once the target methods is successfully processed (As mentioned in (6.j)) store the desired output in a generic variable using Reflection API and return it to the calling interface. This output variable will acts as a parent variable which can be correspondingly unboxed into respective data type by the client.

7 Experimental Results

The scope of effectiveness of this approach can be evaluated on the basis of checking this approach in different type of scenarios.

7.1 Authenticated Web Service

Authenticated web services are those web services which accept a valid username and password for connecting a client to its internal resources for various operations. There are various ways to provide an authentication firewall for the web service like:

i. *Windows authentication*:—Authenticity is checked on basis of valid accounts on which clients are logged into using windows user id and password.
ii. *SOAP headers*:—SOAP headers can be defined as a custom way to apply authenticity to a web service as the owner of the web service implements a separate class using SOAP API and SOAP base classes along with the functional classes of the web service for storing and verifying username and password.
iii. *DB driven*:—Owner of the web service stores and processes the valid username and password by storing it in a database table using database level security like encryption and hashing (Fig. 1).

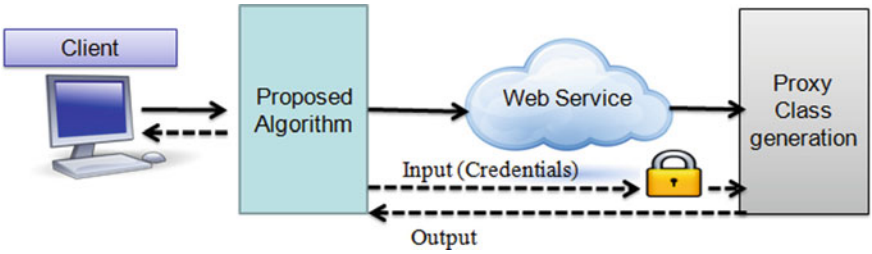

Fig. 1 Authenticated web service

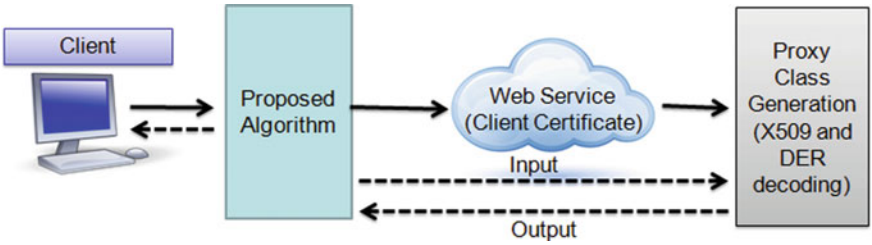

Fig. 2 Web services using client security certificate

Our algorithm handles the authentication of target web service in all the three cases:

i. For windows authentication (7.a.i) our algorithm implements the built in feature of the underlying framework and enforces it to check for valid windows username and password.
ii. For handling SOAP headers authentication (7.a.ii), our algorithm dynamically creates instance of the SOAP header class using 'Compiler Result' support class as mentioned in (6.b) and (6.c). After creating the instance, the algorithm extracts the desired username and password attribute along with their corresponding data type using the 'Reflection API' as mentioned in (6.e). It then sets the username (6.a.vi) and password (6.a.vii), through the input parameters passed to the algorithm.

7.2 Web Services with Client Security Certificate

When the applications call the web services, they must be authenticated by the web services. The authentication checking must be performed before the web service authorization. One of the authentication techniques is achieved by using client certificate. The user may receive an "access denied" error message when a client

application tries to call a web service. But, when a console application tries to call the same web service, the error message may be omitted. In these cases, the computer system keeps two different certificate stores:

iii. The local machine store, which is used for web applications.
iv. The local user store, which is used for interactive applications.

To enable an application to use a client certificate, you must install the client certificate in the local machine store and the user logged into that machine should also have proper permission to access that certificate (Fig. 2). Our algorithm handles the authentication of the client certificate for accessing web services, (given that the valid client certificate is already installed on client machine) by using a specific validation call back approach in which it checks for the valid client certificate using X509 client certificate base classes and parsing of the DER encoded file. (Key for parsing the DER encoded file is to be exported at the time of installing the client certificate).

8 Catering of the Algorithm

As the effectiveness of the proposed algorithm is observed, here is a brief description about how to cater the algorithm in respective target projects.

a. Add the reference of the algorithm to the target project.
b. Create an object of the respective class associated with the algorithm as shown below:

$$WebServiceAlgorithmobj = new\ WebServiceAlgorithm()$$

c. After creation of the object (8.b), call the algorithm (target function) for processing of the web service.
d. For normal ASP.Net web services, the target function (8.c) can be called as shown below:

$$Return\ Object = obj.TargetFunction$$
$$(Web\ Service\ \ URL,\ Service\ Name,\ Method\ Name,\ Input\ parameters)$$

where:

(i) First parameter is the URL of the web service.
(ii) Second one is the service name (class of the web service).
(iii) Third one is name of the method to be called to get the output.
(iv) Fourth parameter (8.g) represents collection of input parameter to be used in the method.

e. For authenticated web service, the target function (8.c) can be called as shown below:

Return Object = obj.TargetFunction
(Web Service URL, Service Name, SOAP Authentication
Header Name, Username, Password, Target Function
Name, Input parameters)

where:
All other input parameters are same (8.d) except for three input parameters:—(SOAP Authentication Header Name) which is the name of the authentication header associated with the web service and contains the respective username and password, (Username) is the desired username and (Password) is the desired password.
Note Above mentioned approach (8.e) highlights authentication using SOAP header. As far as Windows and DB driven authentication is concerned our algorithm does not need SOAP header, only username and password is required.

f. As far as web services with client certificate are concerned, the proposed algorithm is developed in such a way that it caters these cases implicitly.
g. For populating input parameters for the desired function client can form the input structure as follows:

(i) Create a class for the input parameter.
(ii) Declare variables for each input with required data types as needed by the web service.
(iii) Create public caterers for each variable for accessing them.

```
Public Class inputPar
        Data Type Param 1
        Data Type Param 2
        Public Caterer getsetParam1()
                Get
                        Return Param 1
                Set()
                        Param 1= value
                End Set
        End Caterer
Public Caterer getsetParam2()
        Get
                Return Param 2
        Set()
                Param 2= value
                End Set
End Caterer
```

(iv) Assign the variables with the desired values through the public caterers defined.

(v) Pass the object of this class as a parameter in the target function (8.c) of the web service.

h. The output of the proposed algorithm returns a parent object which can be correspondingly unboxed into respective data type.

9 Conclusion

In the proposed algorithm, we have presented an approach for processing ASP.Net web services using delegation technique, in which one common delegate will act as medium of communication between clients and any number and type of ASP.Net web services. The proposed algorithm encourages reusability, modularity and generic development. It also reduces time and effort of referencing each web service explicitly and makes target application adaptive to new changes with respect to web services. We can conclude that the proposed algorithm is a generic, robust and effective algorithm for any.Net web service. In future we will try to increase the scope of the algorithm to different additional platforms.

References

1. John B. Oladosu, Funmilola A. Ajala, Olukunle O. Popoola, "On the Use of Web Services Technology in E-Health Applications", Journal of Theoretical and Applied Information Technology, Vol. 12, No. 2, 2010, pp. 94–103.
2. M. Vasko, S. Dustdar, "An Analysis of Web Services Workflow Patterns in Collaxa", Web Services, Lecture Notes in Computer Science, Springer, 2004, pp. 1–14.
3. C. Boutrous Saab, D. Coulibaly, S. Haddad, T. Melliti, P. Moreaux, S. Rampacek, "An Integrated Framework for Web Services Orchestration", International Journal of Web Services Research, Vol. 6, No. 4, 2009.
4. S. Mokarizadeh, P. Kungas, M. Matskin, "Utilizing Web Services Networks for Web Service Innovation, International Conference on Web Services, IEEE, 2014, pp. 646–653.
5. K. Elgazzar, A.E. Hassan, P. Martin, "Clustering WSDL Documents to Bootstrap the Discovery of Web Services", International Conference on Web Services, IEEE, 2010, pp. 147–154.
6. M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, "Semantic Matching of Web Services Capabilities", International Semantic Web Conference, LNCS, Springer Verlag, 2002, pp. 333–347.
7. McGrawhill Company Inc., "What the heck are Web Services?" Business Week, 2005.
8. A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. K. Liu, R. Khalaf, D. Konig, M. Marin, V. Mehta, S. Thatte, D. V. R. Rijn, P. Yendluri, A. Yiu, "Web Services Business Process Execution Language Version 2.0.", Technical report, OASIS WSBPEL Technical Committee, 2007.
9. C. Peiris, "Creating a .net Web Service", Caulfield, Australia, 2005.
10. Guofeng Yan, Yuxing Peng, Shuhong Chen, Pengfei You, "QoS Evaluation of End to End Services in Virtualized Computing Environments", International Journal of Web Services Research, Vol. 12, No. 1, 2015, pp. 27–44.