PROTOCOL MANAGER: DISTRIBUTED MANAGEMENT OF UNIVERSITY'S OFFICE THROUGH WEBSERVICES

Giovanni Altamore, Paolo Giuffrida, Giuseppe Moscato

University of Catania
Catania, Italy
giovanni.altamore@alice.it, paologiuffrida@gmail.com, moscatog@yahoo.it

Abstract

In this article the development of a software to manage the university's Office protocols will be discussed. In order to develop the various university's office, we used a system based on web services that enable a simple and standard interface for applications that will be implemented to manage the system.

The structure of the server is based on a network ring: each server has a database of its users, a db for the protocols and it also keeps track of the next server. The server has the addresses of all servers connected to the network (in order to have an easier and faster information flow). The ring structure is mainly used in search protocols and users. We have instead preferred a direct exchange of information in the entry and exit process from the ring.

For both server and client, the implementation has been developed through .NET and C # language.

I. INTRODUCTION

A Web Service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network"[1]. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. Other approaches with nearly the same functionality are Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) or SUN's Java/Remote Method Invocation (RMI).

The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate over the HTTP protocol used on the Web. Such services tend to fall into one of two camps: Big Web Services and RESTful Web Services.

"Big Web Services" use XML messages that follow the SOAP standard and have been popular with traditional enterprise. In such systems, there is often a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks (frameworks such as Spring, Apache Axis2 and Apache CXF being notable exceptions). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a Web service.

More recently, RESTful Web services have been regaining popularity, particularly with Internet companies. These also meet the W3C definition, and are often better integrated with HTTP than SOAP-based services. They do not require XML messages or WSDL service-API definitions.

II. PROTOCOL MANAGER ARCHITECTURE

To every office belonging to the university a server is assigned; all servers are linked together through a network ring (fig. 1).

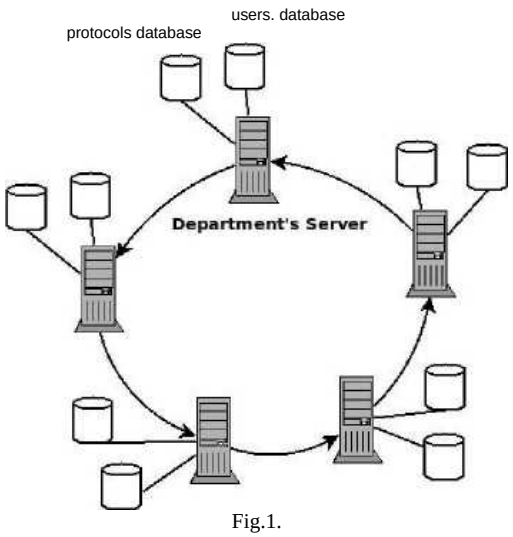

Fig.1.

[1] *"Web Services Glossary"* - W3C Working Group Note 11 February 2004 http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/ Editors: Hugo Haas, W3C, Allen Brown, Microsoft (until June 2002

Each server has a database for office's users, a db for the protocols, a database for the informations on servers belonging to the ring and finally a db with the information of the server (id, url, and url of the next server).

To implement the database we used the XML files because we need to have a simple configuration and easy access to data.

Four Graphical User Interface (GUI) have been performed in order to allow the management of the system by the administrators and to access to data of the network.

For each type of user (administrator and normal user) we have created two types of GUI, a Windows Forms application and an application for Smart Handheld Device. The administrators of the various offices can access the database and modify them: they can insert or delete protocols or users and search a specific protocol or user according to the parameter they prefer (id, date, abstract, signature ...).

III. SOFTWARE IMPLEMENTATION

In this section we analyze characteristics of the server and of the client applications and the criteria used for their implementation.

For both server and client, the implementation has been developed through .NET and C # language.

Server

The structure of the network is a ring in which each server can know the next server and the overall structure of the network.

Each server has the following databases (XML):

```
Database.xml (protocols db)
Users.xml (users db)
me.xml (server info)
DataServer.xml (network info)
```

The file "Database.xml" is structured as follows:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes" ?>
<protocols>
    -<protocol>
          <id></id>
          <Year></Year>
          <Date></Date>
          <Department></Department>
          <Signature></Signature>
          <KeyWords></KeyWords>
          <Abstract></Abstract>
          <Validate></Validate>
    </protocol>
</protocols>
```

The "id" identifies only a protocol, the Validate field is a bool (true or false). The file "Users.xml 'is structured as follows:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes" ?>
<users>
    <user>
          <id></id>
          <Name></Name>
          <Surname></Surname>
          <Password></Password>
          <UrlServer></UrlServer>
          <Admin></Admin>
    </user>
</users>
```

The password is stored encrypting with MD5. The file "me.xml" is structured as follows:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes" ?>
<infos>
    <me>
        <id></id>
        <url>/url>
        <urlNext ><urlNext />
    </me>
</infos>
```

And, finally, the file DataServer.xml is as follows:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes" ?>
<Servers>
    <server>
        <id></id>
        <url></url>
        <urlNext><urlNext />
    </server>
</Servers>
```

Functions development

The structural features of the network are: a function that allows the entry of the server in the ring and a function that allows the exit.

An important aspect of these functions is to change the file "DataServer.xml" and the file "me.xml" of the previous server.

The file "DataServer.xml" describes the network. It contains the general structure of the network. When a server becomes part of the ring it must be included in this database and the reference "urlNext" of the previous server must be changed and the file modified must be redistributed across the network.

The steps to entry in the ring are:

- check if the field nextserver is setted;
- if nextserver == "" then there is a single server;
- check if there is another server with the id chosen;
- check if the database of the protocols id already resent and otherwise create a new db;
- check if a users database exists and otherwise create a new db;
- create the file me.xml with the settings of the server;
- if you create the first server, you must create the atabase server;
- if you do not create the first server, you must receive "DataServer.xml" by the next server;
- receive DataServer.xml through getDataServer(id, url, next) function;
- scroll the server database (except from the server you are putting into the ring) and every server

must save the new server db.

The exit function is perfectly dual to the entry function.

The ring structure is used in the search function through the network; when you want to search a users or a protocols the request is transmitted to the next server until you do not find something.

It is necessary to put into the various search functions the URL of the server which call the function, to avoid infinite loops in the network and to know when the network ring of the servers is been covered.

An example of a search function is "getXbyY".

The "getXbyY" is so declared:

```
public string[] GetXbyY(string X,
string Y, string nome, string first Url)
```

The documentation is:

```
<summary>
It returns X, given Y
</summary>
<param name="X">X (what you want to
return)</param>
<param name="Y">Y (the key to find a
protocol) </param>
<param name="nome">nome (value of Y)
</param>
<param name="firstUrl">url of the first
server</param><returns>array with the
results</returns>
```

All these search functions use the library "LinQ to perform queries on XML databases, for example:

```
var query = from c in
XDocument.Load(Server.MapPath(InfoFile)
).
Element("infos")
Elements("me")
select c;
    foreach (var ci in query)
    (
        nextServer = (string)ci.Element
("urlNext").Value;
    )
```

The most important part to search in the network is as follows:

```
if (!nextServer.Equals("") && !
nextServer.Equals(firstUrl))
{
// I send the request to the next
server
Funzioni.Server server1 = new
Funzioni.Server();
server1.Url = nextServer;
risultato.AddRange(server1.GetXbyY(X,
Y, nome, firstUrl));
}
```

In this way we check that the search is transmitted only if there is not one server and if the next server is the same server which made the request.

An important aspect of this project is the security: we have decided to use soap header to manage the login.

The login function is as follows:

```
public myHeader sHeader;

/// <summary >
/// Method that controls the
authentication header
/// </summary>
/// <param name="firstUrl">url of the
first server that sends the request</param>
/// <returns>bool (checked) </returns>

[WebMethod (Description = "Method that
controls the authentication header")]
[SoapHeader("sHeader")]
public bool checkHeader(string
firstUrl)
{
string usr = sHeader.Username;
string pwd = sHeader.Password;
bool result;

result = AuthHere(usr, pwd);

if (result)
{
return true;
}
else
{
string nextServer = "";

var query3 = from c in
XDocument.Load(Server.MapPath(InfoFile)).
Element("infos")
Elements ("me")
select c;
foreach (var ci in query3)
{
next Server =
(string)ci.Element("urlNext") .Value;

}
if (!nextServer.Equals("") && !
nextServer.Equals(firstUrl))
{
Funzioni.Server service = new
Funzioni.Server();
service.Url = next Server;
Funzioni.myHeader header = new
Funzioni.myHeader();
header.Username = usr;
header.Password = pwd;
service.myHeaderValue = header;
try
{
result = service.checkHeader(firstUrl);
}
catch(System.Exception)
{
return false;
}
}
else
{
result = false;
}
}
return result;
}


/// <summary>
/// Local authentication
/// </summary>
/// <param name="usr">user id</param>
/// <param name="pwd">user password
(md5)</param>
```

```
      /// <returns></returns>
      private bool AuthHere(string usr,
string pwd)
      {
      var query = from c in
      XDocument.Load(Server.MapPath(DataUsers
)).Element("users"). Elements("user")
      where (string)c.Element("id") .Value ==
usr & & (string)c.Element("Password").Value
== pwd select c;
      if (query.Count()  != 0) return true;
      else return false;
      }
```

Client

In GUI user authentication is as follows:

```
      [SoapHeader("sHeader")]
      private void entra_Click(object sender,
EventArgs e)
      {
      bool login = false;
      Funz   ioni.Server   service   =   new
Funzioni.Server();
      Funzioni.myHeader    header    =    new
Funzioni.myHeader();
      // bytes stream - Md5
      Byte[] originalBytes;
      Byte[] encodedBytes;
      //user header
      header.Username = log.Text; //username
dell'utente
      System.Security.Cryptography.MD5CryptoS
erviceProvider   x   =   new   System.Security.
Cryptography.MD5CryptoServiceProvider();
      originalBytes                          =
ASCIIEncoding.Default.GetBytes(psw.Text);
encodedBytes = x.ComputeHash(originalBytes);
      StringBuilder    strCriptata    =    new
StringBuilder();
      for(int i=0; i<encodedBytes.Length; i++)
      strCriptata.Append(encodedBytes[i].
ToString("x2"));
      //pass header
      header.Password = strCriptata.ToString();
      service.myHeaderValue = header;
      login=service.checkHeader(service.Url);
      if (login &&
      service.GetXbyUser("Admin",log.Text,ser
vice.Url)=="true")
      {
      FormPrincipale     frmprin     =     new
FormPrincipale();
      frmprin.Show() ;
      this.Hide();
      }
      else
      {
      if (! login)
      {
      MessageBox.Show("Login/Password  errati",
"Ufficio  Protocolli",  Mes  sageBoxButtons.OK,
Mes sageBoxI con.Error) ;
      }
      else
      {
      MessageBox.Show("Non    hai    permessi
sufficienti   per   effettuare   il   login",
"Ufficio  Protocolli",  MessageBoxButtons.OK,
MessageBoxIcon.Error);
      }
      }
      }
```

IV. GRAPHICAL USER INTERFACE (GUI)

Let's go now to analyze the graphical interface of our simulator. The main window is shown in the next picture.

We note first that the graphical interface presents only two drop-down and, under these, a great toolbar.

For each type of user (administrator and normal user) we have created two types of GUI, a Windows Forms application and an application for Smart Handheld Device.

The start window of the Windows Form Application is shown in fig. 2. and fig.3.

Fig. 2.

The window is dedicated to user login. Once you logged the main menu will open. There are two areas: one dedicated to manage the protocols and the other to manage users.

Fig. 3.

In the protocols area there are three buttons that allow connection to three forms: to enter a new protocol, to search a protocol and to remove a protocol.

In the area dedicated to user management the buttons have functions similar to those relating to the protocols: to enter a new user, to search a user by name and surname, to search a user by id and to delete user.

Below there are the windows forms linked to each button (fig. 4 – fig. 12).

Fig. 4.

Enter a new protocol

To insert into the database a new protocol all the fields in the form must be complete: protocol number, date (just click on the calendar), abstract, keywords (chosen from a dropdown menu), signature, and then the validate field.

Fig. 5.

Search a Protocol

To search for a specific protocol you must write: in the first field the field of protocol that you want returned (id, Year, Date, Signature, KeyWords, Abstract, Validate), in the second the search criterion (id, Year, Date, Signature, KeyWords, Abstract, Validate) and the third the value of the criterion.

In the text to the right you will see the results.

An important aspect is that when you enter a new protocol it is stored in the database of the server you are connected, while the search is done on all databases in the network.

Fig. 6.

Delete a Protocol

To delete a protocol from the database simply enter the number (id) into the text area and press the button "Delete" We will see now the form for the management of users.

Fig. 7.

Enter a new user

To insert a new user in the database you must complete all the fields and specify if the new user has administrator permissions (checkbox "administrator"). To insert a new user there is a control on "id" which must be unique.

Fig. 8.

To search a user into the databases of the networks you must enter in the first field the field that you want returned (id, UrlServer, Admin) and in the other two fields Name and Surname. The result will appear in the text on the right

Fig. 9.

Search a user by id

You can search by id.

Fig. 10.

Delete user

To delete a user you must enter user id and click on "Delete."

In the Windows Forms application for a normal user you have only the form to search protocols or user. This is the main menu.

Fig. 11.

We have developed two applications for Smart Handheld Device: the first dedicated to the administrator of university's office and the another to a normal user.

The functions are identical to those provided by the Windows Form Application

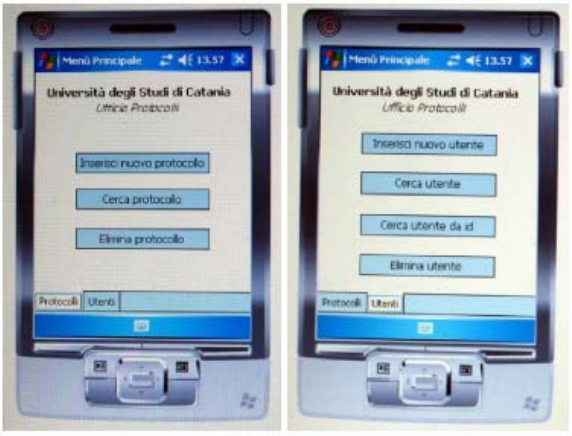

Fig. 12.

V. CONCLUSION

The power and versatility of web services have allowed us to imagine future scenarios to use our software with enhancements that concern not only the simple functionality but also the security. It would be possible to use secure channels of communication client/server to exchange information, from login to the transition of data, using the encrypted data stored in the DB.

Our clients use, as we have seen, the windows form to create easy and intuitive GUI, but this windows form do not allow interoperability between systems. A possible solution would be to use Open technology (python + GTK or java) to create exportable GUI client.

The web services were born to be standard and to enable the interoperability of systems