WEB SERVICES INTEGRATION WITH DISTRIBUTED APPLICATIONS

Marilena DUMITRACHE
PhD Student, Faculty of Cybernetics Statistics and Economic Informatics, Academy of Economic Studies, Bucharest, Romania

E-mail: dm.marilena@gmail.com

Stelian DUMITRA
PhD Student, Faculty of Cybernetics Statistics and Economic Informatics, Academy of Economic Studies, Bucharest, Romania

E-mail: stelu20d@yahoo.com

Mircea BACIU
PhD Student, Faculty of Cybernetics Statistics and Economic Informatics, Academy of Economic Studies, Bucharest, Romania

E-mail: mircea.baciu@gmail.com

Abstract: *A Web service is a distributed application component. Web services distributed computing model allows application-to-application communication. There is nothing fundamentally new about the basic concept and the related technologies. The innovative thing about this is the reach of Web services and its ubiquitous support by literally all major vendors. Most likely, heterogeneity will at the end no longer be an obstruction for distributed applications. This paper describes the concept of service-oriented architecture (SOA) in conjunction with the Web services technology and covers the core Web services specifications which form a powerful and robust foundation for building distributed systems. It is presented a case study regarding the integration of the Web services with the SAP system for handling interoperability issues.*
The conclusions and the future proposed developments are presented in the end of the paper.

Key words: *distributed application; web service; sap; protocols; security*

1. Introduction

Because of the level of the application's integration, the Web services have grown in popularity and are beginning to improve the business processes. In fact, the Web services are being called the next evolution of the Web [1].

Web services provide a promising framework for development, integration, and interoperability of distributed software applications. Wide-scale adoption of the web services technology in critical business applications will depend on the feasibility of building highly dependable services. Web services technology enables interaction of software components across organizational boundaries. In such distributed environment, it is critical to eliminate errors at the design stage, before the services are deployed. Web services provide a promising framework for development, integration, and interoperability of distributed software applications. Wide-scale adoption of the web services technology in critical business applications will depend on the feasibility of building highly dependable services.

The remainder of the paper is structured as follows: section 2 provides information about service-oriented architecture (SOA) in conjunction with the Web services technology and the core Web services specifications; section 3 describes a case study on web services and associated key technologies. The application shows how to integrate Web Services on different platforms and how they allow the interoperability between applications running on these platforms, using the specific web services protocol stack presented in section 2. Section 4 concludes the paper and presents future proposed developments.

2. Service Oriented Architecture vs. Web Services

Distributed software systems, and the interactions between components within these systems, can exhibit a high level of complexity and lead to difficulty in the assessment of what system behavior is possible in multiple scenarios [2].

2.1. SOA

Nowadays, service-oriented architecture (SOA) and web services that enable flexible and loose integration of applications within and across enterprises have become one of the most phenomenal subjects both in academia and in industry.

SOA is a software architectural concept that defines the use of services to support the requirements of software users. It is a system for linking resources on demand. In an SOA, resources are made available to other participants in the network as independent services that are accessed in a standardized way. This provides for more flexible loose coupling of resources than in traditional systems architectures [3].

The SOA model treats three main elements that act as a find–bind/invoke-execute cycle as shown in Figure 1. The service provider offers a given service and publishes service description in a service registry. The service requester queries the registry to find a certain service. If founds, it retrieves the location of the service and binds to the service endpoint, where the requester can finally invoke the operations of the service [4].

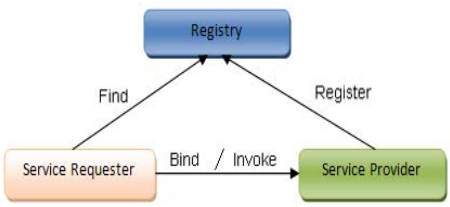

Figure 1. SOA triangle

Figure 2. Architecture of the Web Services framework

SOA-based applications are distributed multi-tier applications that have presentation, business logic, and persistence layers. Services are the building blocks of SOA applications. While any functionality can be made into a service, the challenge is to define a service interface that is at the right level of abstraction. Services should provide coarse-grained functionality [5].

Within SOA, web service providers describe their services in WSDL to designate what they are and how to invoke them, and then publish these descriptions via a public UDDI registry. On the other hand, a service requester subscribes those WSDL descriptions and selects such services that satisfy an integration need. The requester often has to compose several services to accomplish complex tasks. Then, the requester invokes selected web services using XML/SOAP messages. All these can be seen in figure 2 [6].

2.2. Web Services

The client's basic needs over time don't really change, but the essential tools that are required in order to fulfill these needs are constantly evolving. Not long ago in nodes, which were present in distributed systems, existed the need to control the applications in a distributed manner. [7] Basically if an application that was running in a node happened to go down, that application was suppose to be restarted at another node. The creation of these types of distributed applications was nearly impossible. These days, it's routine and in fact there are many choices. The essential problem is not if it is possible for the components of distributed applications to communicate between them, but to choose the best technology in order to hold them together.

For example the .NET Framework, introduces good support for the two ways to architect a distributed application. Remoting is the architectural descendant of DCOM, allowing an object on one computer to make proxy-based calls to the methods of an object on another computer. Web services use a completely different technique, based on open XML and SOAP protocols, WSDL and UDDI, which are used to invoke methods on a remote machine. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing available services.

"Web services are dynamic programs that enable data and applications to interact with each other on the Web through ad hoc connections—without any human intervention what so ever", said Sidharth, technical product manager for identity management at Sun. A Web service is normally intended to be a distributed application component. Its clients are other applications, not human beings. A Web Service is any piece of code that can communicate with other pieces of code via common Internet technology. A Web Service is a "virtual component" that hides "middleware idiosyncrasies" like the underlying component model, invocation protocol as far as possible.

The main advantages of Web services are flexibility and versatility: they support a lot of architecture and are independent of platforms and designs. Web services are built on several technologies that work in conjunction with emerging standards to ensure security and manageability, and to ensure that Web services can be combined to work independent of a vendor. Also Web services win on ease of development and interoperability.

Web services distributed computing model allows application-to-application communication. For example, one purchase-and-ordering application could communicate to an inventory application that specifies the items that need to be reordered or a Web service from a credit bureau which requests the credit history from the loan services, for prospective borrowers. In both cases, the data interaction must be protected to preserve its confidentiality.

Web Services are considered to be the future of the Internet. They are independent of the platform and also of the technology, but in reality they are XML/SML collections of standards which allow the interaction between systems (programs). Heather Kreger, one of the IBM's lead architects for SOA Standards which developed the standards for Web services, thought that Web Services are like an interface which describes a collection of operations, network accessible throughout the XML standard messages. Web Services have the main role to access different services and different data from different machines, and so they offer to the clients a single public interface.

2.3. The Web Services Architecture

Figure No. 3 describes the architecture of the Web Services. The architecture of the Web Services resembling with the TCP/IP reference model is presented in five levels: Network, Transport, Packing, Description and Discovery. Each level is represented by different basic protocols. The network level concurs with the network level from the TCP/IP [8] reference model, offering basic communication, addressing and rooting. Above the network level there is the transport level that offers the opportunity of direct communication between the existing applications from the network. The most important protocols are TCP/IP, UDP, FTP, HTTP, SMTP, Jabber [9].

Figure 3. Web Services Architecture

The web services can be implemented above any of the other protocols. The Packing level, which is above the Transport level, "packs" the data in the XML format – a format known by all the other parties involved in communication. XML and SOAP – Simple Object Access Protocol, are basic protocols of the Packing Level and are produced by the W3C standard.

2.4. SOAP – Simple Object Access Protocol

The role of SOAP is to encode the data in XLM format and to make the message exchange possible between the applications in XML format. It uses the model request-answer, where the request is placed by the SOAP client, and the answer is given by the service provider, named SOAP server. Everything is shown in the below situated Figure 4.

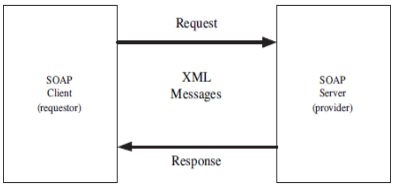

Figure 4. SOAP's basic request-response model

The protocol is used both to send and to receive messages from the Web Service. One advantage is to encapsulate the functionality of the RPC (Remote Procedure Call) using the extensibility and the functionality of the XML. SOAP defines a format for both messages, and a model for their processing by the receiver. In addition, SOAP – may also define a framework for protocol links, so that the SOAP messages can be transferred using the protocol stack from the transport level.

A SOAP message consists of a SOAP envelope [10], the root of the message, which in turn contains an optional header, and, necessarily, a body, independent of each other. SOAP message passes on its way from sender to receiver through many SOAP nodes, which can change the message. All the SOAP nodes form SOAP message path.

The Header contains general information about security – authentication and session, and about the message processing by the intermediary nodes. The data regarding the authentication usually is encrypted using WS-Security standard. The tag - "body" never misses from a SOAP message. Most of the times it is the last child of the "Envelope" node and it contains the information that is going to be transferred between applications (Web service input or output).

In Figure 5 it is shown an example of a SOAP message.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <GetMaterialsResponse xmlns="http://tempuri.org/">
      <GetMaterialsResult>xmlxml</GetMaterialsResult>
    </GetMaterialsResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 5. SOAP Message

In the above example, in the SOAP envelope the XML namespace and the type of the used message encoding are not specified. The Header node is missing in this example, and the body node contains the result of a "*GetMaterials*" method, which is a serialized Data Table type object in XML format.

SOAP was originally an acronym for Simple Object Access Protocol, But since SOAP Version 1.2 (SOAP 1.2 Part 0, 2003; SOAP 1.2 Part 1, 2003) it is technically no longer an acronym.

2.5. WSDL – Web Services Description Language

The description level, located above the packing level, is represented by the WSDL protocol, being based on the XML standard.

WSDL is a language written in XML [11], used as a model for describing Web services. WSDL reached version 2.0, but in version 1.1 the **D** stood for Definition. Version 1.2 of WSDL was renamed WSDL 2.0 because of the major differences between the two versions.

2.5.1. New features in WSDL 2.0

Nowadays, W3C recommends using WSDL 2.0 [11], but the problem is that it is not fully supported in all developing environments. The main differences between the two versions are:

- in WSDL 2.0 there's binding to all the HTTP request methods, whereas in WSDL 1.1 only the GET and POST methods;
- in WSDL 2.0 further semantics were added to the description language;
- WSDL 2.0 offers better support for RESTful web services;
- renaming of PortTypes (WSDL 1.1) into Interfaces (WSDL 2.0);
- renaming of Ports (WSDL 1.1) into Endpoints (WSDL 2.0);
- WSDL 2.0 can be implemented in a much simpler way.

WSDL is used in combination with SOAP and the XML schema representing the web service description. The main purpose of WSDL is that it leverages the connection between a client program and a web service, by determining the server available operations.

2.5.2. WSDL Components

Port/Endpoint – defines the address or connection to a web service; usually, it is represented by a simple URL.

Service – consists of a set of ports/endpoints, meaning the system functions exposed to the web based protocols.

Binding – defines a concrete message format and transmission protocol which may be used to define a port/endpoint.

PortType/Interface – defines a web service, all the operations that can be performed, and the messages used to perform the operation.

Operation – is an interaction with the service (a method) formed by a set of messages exchanged between the service and the other programs involved in the interaction.

Type – describes the data type definitions that are relevant for the exchanged messages.

Components 1-3 represent the concrete section of a WSDL, and components 4-6 represent the abstract section.

In Figure 6 it is shown an example of a WSDL message.

```
- <definitions name="ServiceSAP" targetNamespace="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx?WSDL">
  - <wsdl:message name="GetMaterialsSoapIn">
      <wsdl:part name="parameters" element="tns:GetMaterials"/>
    </wsdl:message>
  - <wsdl:message name="GetMaterialsSoapOut">
      <wsdl:part name="parameters" element="tns:GetMaterialsResponse"/>
    </wsdl:message>
  - <wsdl:portType name="ServiceSAPSoap">
    - <wsdl:operation name="GetMaterials">
        <wsdl:input message="tns:GetMaterialsSoapIn"/>
        <wsdl:output message="tns:GetMaterialsSoapOut"/>
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="ServiceSAPSoap" type="tns:ServiceSAPSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    - <wsdl:operation name="GetMaterials">
        <soap:operation soapAction="http://tempuri.org/GetMaterials" style="document"/>
      - <wsdl:input>
          <soap:body use="literal"/>
        </wsdl:input>
      - <wsdl:output>
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="ServiceSAP">
    - <wsdl:port name="ServiceSAPSoap" binding="tns:ServiceSAPSoap">
        <soap:address location="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx"/>
      </wsdl:port>
    </wsdl:service>
```

Figure 6. Example of a WSDL

2.6. UDDI – Universal Description, Discovery and Integration

UDDI represents a platform-independent framework [10], based on Extensible Markup Language (XML), a directory service where businesses can register and search for web services. UDDI is meant to be open for businesses, enabling them to publish and discover services, and to discover the interaction between them over the Internet.

2.6.1. UDDI – Characteristics of UDDI [8]

UDDI stores information about web services, it consists of web services interfaces written in WSDL. UDDI can be interrogated via SOAP messages and provides access to WSDL documents describing certain protocol bindings and message formats to interact with the web services. UDDI terminology contains also the following:

- Nodes – servers which support UDDI specifications nodes belong to a registry;
- Registries – collections of one or more UDDI nodes.

2.6.2. Benefits of UDDI

All businesses can benefit of UDDI because it solves the following problems:

- Descovering the right business from millions of online businesses;
- Once the preferred business is discovered, UDDI enables how to enable commerce;
- New customers cand be reached and access to current customers can be increased;
- Market reach and offerings can be expanded;
- Barriers are removed to allow rapid participation in the global Internet economy;
- Services and business processes are programatically described in a single, open, and secure environment.

In Figure 7 it is shown an example of a UDDI message.

```
- <discovery>
    <contractRef ref="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx?wsdl" docRef="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx" />
    <soap address="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx" binding="q1:ServiceSAPSoap" />
    <soap address="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx" binding="q2:ServiceSAPSoap12" />
  </discovery>
```

Figure 7. Example of a UDDI

Although the basic specifications of Web Services: XML, SOAP, WSDL and UDDI provide an acceptable level of interoperability and integrity [11] a significant effort has made to increase the applications area of Web Services, and to address to higher various issues from the real world. Thus new specifications emerged for Web Services' reliability, security, metadata management, transactions and orchestration, all of which have extended the Web Services' architecture. Among the new specifications it is worth to be mentioned:

- Metadata Management: WS-Addressing, WS-Policy, WS-MetadataExchange;
- Reliable Messaging: WS-Reliability, WS-ReliableMessaging, WS-Eventing, WS-Notifications;
- Security: WS-Authorization, WS-SecurityPolicy, WS-Trust, WS-SecureConversation, WS-Federation, WS-Privacy, XML Encryption, XML Signature;
- Transactions: WS-Transactions family(WS-AtomicTransaction, WS-BusinessActivity, WS-Coordination), WS-Composite Application Framework(WS-CAF);
- Orchestration
- Choreography

3. Case study regarding the integration of Web Services on different platforms

This section describes a case study on web services and associated key technologies, offering practical tests in order to support the previous presented. The application shows how to integrate Web Services on different platforms and how they allow the interoperability between applications running on these platforms, using the specific web services protocol stack presented in previous sections.

The problem we have modeled is as follows: "A company that sells used hardware in the fields of retail and food industry wants to integrate some mobile terminals, which are going to be used in order to scan the bar codes of the store's hardware equipments with an ERP system, in our case SAP."

Using a mobile terminal [12], we want to make different types of storage-specific operations, such as: reception, delivery and material's inventory.

The solution we present for integrating the mobile devices with SAP - consists in developing a web service on a .NET platform, that can be used as a proxy between the two platforms.

Figure 8 presents the data flow between these distributed systems, where the interoperability occurs based on the Web services.

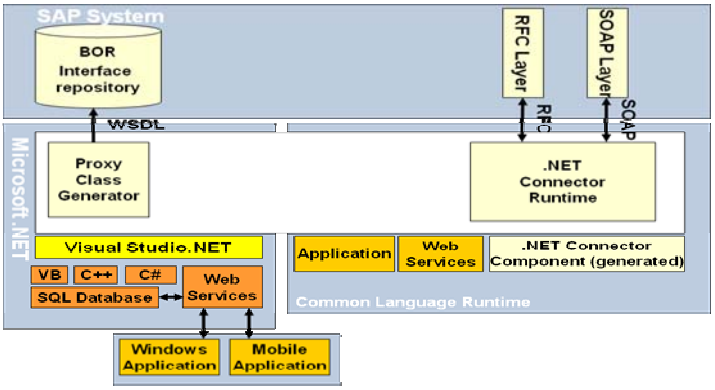

Figure 8. Interfacing .NET Applications with SAP via Web Services

Brief description of the scheme.

Integrated with SAP Business Object Repository (BOR) [13] is an object-oriented repository that contains SAP objects and SAP interfaces, and also their components, such as methods, attributes and events. In SAP Web Application Server, SOAP runtime provides us the mechanism of using SOAP protocol to call and access the RFC functions (Remote Function Call) [14] via HTTP. A web service in SAP can be seen as an RFC function.

The interoperability between the Microsoft .NET platform and the SAP system is done with the help of a component, the SAP Connector for Microsoft .NET [14]. It supports SAP Remote Function Call (RFC) and Web Services and it allows the development of various applications on the .Net platform. The Proxy class generation, which enables the calling for web service (RFC function) is based on WSDL. In this case, it can be said that SAP connector acts as a proxy between SAP and the .Net platform.

In order to integrate the mobile terminal with SAP, it has been developed a web service that becomes the wrapper class for the SAP connector and also it becomes a proxy between the application from the terminal and SAP.

In order to integrate the Web Service with both distributed systems some steps must be taken:

Stage 1: Developing and publishing the Web Services

Using the developing platform Visual Studio.NET, a series of appropriate methods for the data exchange is constructed. In this initial step the .Net connector is integrated to SAP. If the SAP authentication was successfully done, the description of the available RFC functions it is brought from SAP with the help of UDDI and WSDL. Since WSDL –is based on the XML standard the description of the functions is brought into a file with a .sapwsdl extension in XML format. Based on the WSDL and on the .NET connector, the proxy class, that contains the signature of the functions from the file, is being generated in order to become available for calling by other work methods.

In the example from below is shown a description of a RFC function from SAP on a proxy call basis, function that allows the over taking of the materials from the system:

```
/// <summary>
/// Remote Function Module Z_RFC_MATERIALE.
///
/// </summary>
/// <param name="Gv_Eroare"></param>
/// <param name="Gv_Mtart"></param>
/// <param name="Gt_Matnr"></param>
[RfcMethod(AbapName = "Z_RFC_MATERIALE")]
[SoapDocumentMethodAttribute("http://tempuri.org/Z_RFC_MATERIALE",
 RequestNamespace = "urn:sap-com:document:sap:rfc:functions",
```

```
RequestElementName = "Z_RFC_MATERIALE",
ResponseNamespace = "urn:sap-com:document:sap:rfc:functions",
 ResponseElementName = "Z_RFC_MATERIALE.Response")]
 public virtual void Z_Rfc_Materiale (
 [RfcParameter(AbapName = "GV_MTART",RfcType=RFCTYPE.RFCTYPE_CHAR, Optional =
false, Direction = RFCINOUT.IN, Length = 4, Length2 = 8)]
[XmlElement("GV_MTART", IsNullable=false, Form=XmlSchemaForm.Unqualified)]
 string Gv_Mtart,
[RfcParameter(AbapName = "GV_EROARE",RfcType=RFCTYPE.RFCTYPE_CHAR, Optional =
true, Direction = RFCINOUT.OUT, Length =  80, Length2 = 160)]
[XmlElement("GV_EROARE", IsNullable=false, Form=XmlSchemaForm.Unqualified)]
 out string Gv_Eroare,
[RfcParameter(AbapName = "GT_MATNR",RfcType=RFCTYPE.RFCTYPE_ITAB, Optional = true,
Direction = RFCINOUT.INOUT)]
 [XmlArray("GT_MATNR", IsNullable=false, Form=XmlSchemaForm.Unqualified)]
 [XmlArrayItem("item", IsNullable=false, Form=XmlSchemaForm.Unqualified)]
 ref ZMATNRTable Gt_Matnr)
 {
 object[]results = null;
  results = this.SAPInvoke("Z_Rfc_Materiale",new object[] {Gv_Mtart,Gt_Matnr });
  Gv_Eroare = (string) results[0];
  Gt_Matnr = (ZMATNRTable) results[1];
 }
```

In the following example it is shown a method of web service "GetMaterials" used to call RFC function rendered above:

```
[WebMethod]
public DataTable GetMaterials()
{
   try
   {
      string error = "";
      SAPProxy proxy = new     SAPProxy(this.BuildConnectionString());
      ZMATNRTable materialsERSA = new ZMATNRTable();
      proxy.Z_Rfc_Materiale("ERSA", out error, ref materialsERSA);
      if (error != String.Empty)
         throw new Exception(error);
      Access access = new
Access(ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString);
         for (int i = 0; i < materialsERSA.Count; i++)
            access.SynchronizeMaterials(materialsERSA[i], "ERSA");
         return materialsERSA.ToADODataTable();
      }
      catch (Exception exc)
      {
         throw (exc);
      }
   }
```

Stage 2: Rendering the Web Service

At this stage, after the construction of the web service, it will be rendered in the application from the mobile terminal which is developed on the .Net platform. If the service is available, similar to Stage 1, the description of the service in the application is brought with the help of UDDI and WSDL.

Stage 3: Calling methods of service

After rendering the contract and after the generation of the proxy class to call the service, in the mobile device application one can call the service methods.

Exchanging data between applications in the background, in the form of SOAP messages is done asynchronously and at runtime. If there is a large volume of data, due to the interoperability based on XML standard, the process may be slowed because of the need for parsing of the XML messages in different types of objects. This exchange of messages is done by following these steps:

Step 1: Request the mobile application service

While calling a method from the proxy class, on a SOAP framework basis, a request is developed, in XML format. This message is attached to the SOAP - and will on a HTTP protocol basis to the service. If the service is not available and if there are no different restrictions on the network, next will be fallowed step 2, otherwise the application ends.

Step 2: Request service from SAP

The request was received by the service and it will be forwarded to SAP through connector and the proxy class. Based on the SOAP Runtime protocol, which provides access and the calling to the RFC functions, all the messages are sent in XML format via HTTP, and they are parsed as objects and BOR specific structures.

Step 3: The SAP's answer to the service

The result of the request from Step 2 is sent in XML format by the SOAP Runtime and it is parsed by the .NET connector into the service's specific objects.

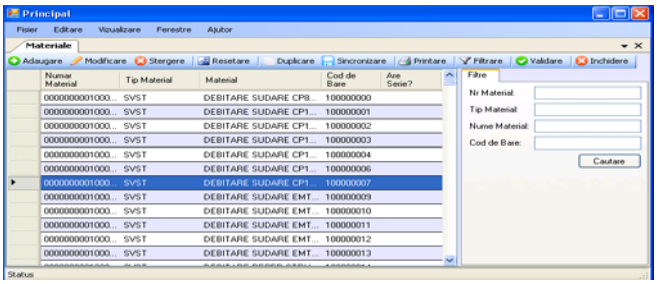

Figure 9. Simple call web service from Windows application

Step 4: The answer of the service to the mobile application

The result from Step 3, as a response from service to customer, it is sent like in Step 1. The XML message is transformed by SOAP's framework into specific objects from the development environment, in our case a DataTable object type which can be seen in Figure 9.

4. Conclusions

Web services play a similar role with older technologies such as Remote Procedure Call (RPC), Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), but also offers several advantages over these. The great advantage of Web Services is that they can integrate different platforms and allow the interoperability between different distributed systems. The Integrity and the interoperability can be addressed with the SOA's help in a two-stage process that involves publishing and orchestrating the web services. The Independence with respect to various platforms is due to common standard that they all have as a basis, the XML. Using Web services provides to the developers the opportunity to create high quality applications more quickly.

In this paper, the authors came up with a pattern of using and integrating the Web services, proofing the interoperability between multiple distributed applications running on different platforms.

References

1. * * * **Understanding Web Services** http://www.webopedia.com/DidYouKnow/Computer_ Science/2005/web_services.asp]
2. Foster, H., Uchitel, S., Magee, J. and Kramer, J. WS-Engineer **A Model-Based Approach to Engineering Web Service Compositions and Choreography**, Springer Berlin Heidelberg, 2007, pp. 87-119
3. Ciurea, C., DUMITRACHE, M. and Doinea, M. **Distributed collaborative systems security**, KEPT 2009 Conference The second edition, Cluj - Napoca, July 2009, pp. 20-24
4. Kovač, D. and Trček, D**. Qualitative trust modeling in SOA, Journal of Systems Architecture, in Secure Service-Oriented Architectures**, vol. 55, Issue 4, April 2009, pp. 255-263
5. * * * http://java.sun.com/developer/technicalArticles/WebServices/soa/
6. Buhwan, J., Hyunbo, C., Choonghyun, L**. On the functional quality of service (FQoS) to discover and compose interoperable web services**, Expert Systems with Applications, vol. 36, Issue 3, Part 1, April 2009, pp. 5411-5418
7. Ivan, I., Vintilă, B., Palaghiță, D., Pavel, S. And Doinea, M. **Risk estimation models in distributed informatics applications**, Globalization and Higher Education in Economics and Business Administration (GEBA 2009), Iasi, Romania, 22nd to 24th of October 2009, pp. 72-92
8. Tidwell, D. and Snell, J. **Programming Web Services with SOAP**, O'Reilly, 2002, 225 pp.
9. Smeureanu, A., Dumitrescu, S.D. **Metode de îmbunătățire a performanțelor aplicațiilor GIS,** Journal of doctoral research in economics, vol 1, no 1, 2009, pp. 22-29
10. Hollar, R., Murphy, R. **Enterprise Web Services Security**, Charles River Media, 2006, pp. 433
11. Tzima, F. and Mitkas, P. **Web Services Techology**, Information Science Reference, 2008, pp. 25-44
12. * * * http://sdn.sap.com
13. SAP-BC ABAP Programming Release 4.6B, SAP-AG, 2001, pp. 1540
14. Nielsen, E.S., Ruiz, S.M. and Rodriguez-Pedrianes, J. **Mobile and Dynamic Web Services**, Birkhauser Verlag, Basel, 2007, pp. 117-133