UNIVERSITY “POLITECHNICA” OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

Secure access to cloud services using Blobseer

Supervisors:

Prof.dr.ing. Valentin Cristea

As.drd.ing. Cătălin Leordeanu

Author:

Maria Dumbravă

-2010-

Table of Contents

1. Introduction

We need to deploy Web Services in Cloud in a safe way and we need the data management on which those services rely on to be trustful, therefore we have chosen BlobSeer as a storage platform. In a distributed data environment, BlobSeer is the best solution.

Users can create the Web Services they want over BlobSeer and deploy them in a safe platform that doesn't allow unwanted clients to log into the system or to access the Web Services of the users.

Access to the platform is granted after the authentication of the client. A client has to register to the system first, then authenticate, in order to get a Session ID (SID), which allows his unrestricted access to the platform.

Access to the Web Service deployed into the system is decided by the ACLs. Each user possesses a list containing the services he has certain rights to. There are four levels of rights: 0 – administrator rights, 1 – owner rights, 2 – user rights, 3 – client rights (a client has practically no rights; in order to obtain rights to certain services, he must send request messages to the owners or the administrator).

By using this secure platform, a user is able to deploy the web services he desires, with the certitude that the respective services will be accessed only by the clients he wishes to grant access to and by no other person.

We are proposing a reliable, scalable, easy-to-use distributed system, that can be applied successfully in small or large-scale projects, in business or in research, for home users or for large companies.

2. BlobSeer

2.1 What is BlobSeer?

BlobSeer is a data storage service. It is designed to deal with the requirements of large-scale data-intensive distributed applications. Data is abstracted as large sequence of bytes and stored as a **blob** (binary large object), which can be seen as an infinite large-scale file.

. A blob is a data type that can store binary data. Blobs are typically images, audio or other multimedia objects, though sometimes binary executable code is stored as a blob. Basically every type of data can be stored as blobs. In fact, originally "blob" was used as a term for moving large amounts of data from one place (database) to another without filters or error correction. This sped up the process of moving data by putting the responsibility for error checking and filtering on the new host of the data. The act of moving huge amounts of data was called "blobbing". This is in fact the aspect we are looking for in this work.

The architecture:

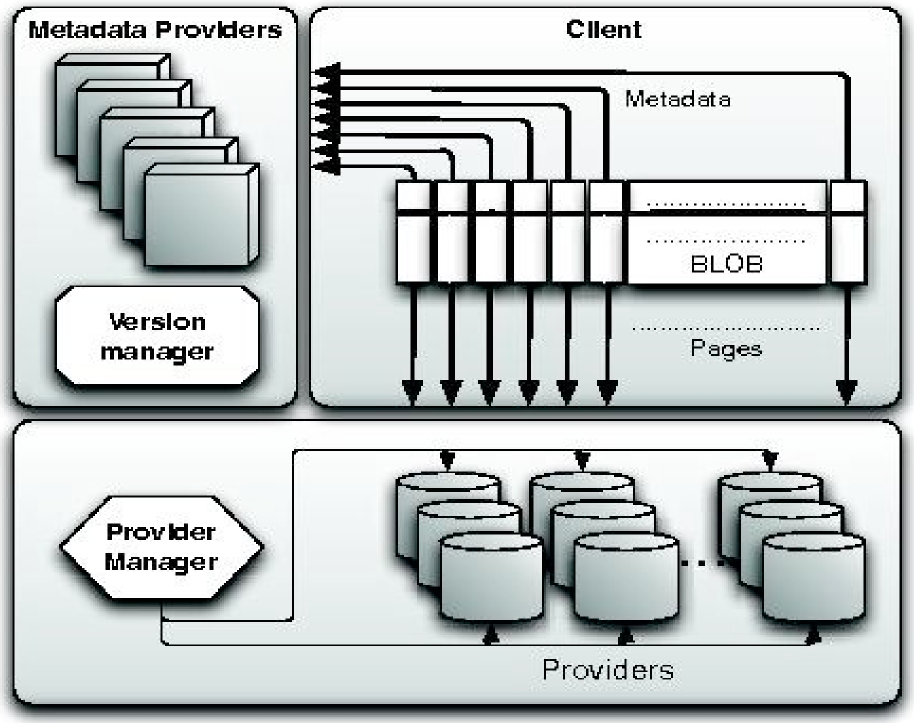


fig1: BlobSeer architecture

There are 3 entities: Clients, Version manager and Provider managers. Clients can CREATE blobs, then READ, WRITE and APPEND data to them. There can be multiple concurrent clients, and their number can dynamically vary in time. Data providers physically store the pages generated by WRITE and APPEND commands. New data providers can dynamically join and leave the system transparent for clients.

The provider manager keeps information about the available storage space and schedules the placement of newly generated pages according to a load balancing strategy.

Metadata providers physically store the metadata allowing clients to find the pages corresponding to the blobs version.

The version manager is the key actor of the system. It registers update requests (APPEND and WRITE), assigns version numbers, end eventually publishes new blob versions, while guaranteeing total ordering and atomicity.

A client of BlobSeer manipulates a blob through a simple access interface. All the actions (reading/writing) can be specified by a subsequence of size bytes from/to the blob starting at offset and appending a sequence of size bytes to the blob. This access interface is designed to support versioning explicitly: each time a write or append is performed by the client, a new version of the blob is generated and it's not overwriting any existing data.

Each blob is made up of fixed-sized chunks (pages) that are distributed among data providers. Metadata is used to map ranges defined by (offset, size) to the data providers where the corresponding chunks are stored, for any existing version of a blob. This metadata is stored and managed by metadata providers through a decentralized, DHT-based infrastructure. A central version manager is responsible of providing cosistency by assigning versions to writes and appends and exposing these versions to reads. Finally, a provider manager decides which chunks are stored on which data providers when writes or appends are issued by the clients.

The pages of a blob (chunks of a fixed size) are specified at the time the blob is created. Typically the size of the page matches or is a multiple of the size of the data the client is expected to process in one step. These chunks are distributed all over the storage space providers of BlobSeer.The data distribution strategy has a major role when concurrent clients access different parts of the blob. Since each blob is spread across a large number of storage space providers, BlobSeer maintains metadata that maps subsequences of the blob to the corresponding chunks. BlobSeer uses a distributed metadata management scheme, avoiding having the metadata server act as a single point of failure.

Versioning is not only provided at client level through the access interface, but is also used to increase the number of operations performed in parallel by avoiding synchronization as much as possible, so no existing data or matadata is ever overwritten, new data or matadata are added and never modified. This enables the readers to be completely decoupled from the writers.

How it works?

Reading data: To read data, clients need to provide a blob id, a specific version of that blob, an offset and a size. The client first contacts the version manager. If the version has been published, the client then queries the metadata providers for the metadata indicating on which providers are stored the pages corresponding to the required blob. Finally, the client fetches the pages in parallel from the data providers. If the range is not page-aligned, the client may request only the required part of the page from the page provider.

Writing and appending data: To write data, the client provides the size and knowing the page size it can be determines the number of pages that cover the range to be written. It then contacts the provider manager and requests a list of page providers able to store the pages. For each page in parallel, the client generates a globally unique page id, contacts the corresponding page provider and stores the contents of the page on it. After that, the client contacts the version manager to registers the new writes. The version manager assigns a new version and communicates it to the client. Finally, the client notifies the version manager of success, and returns successfully to the user. Next, the version manager publishes the version of the blob.

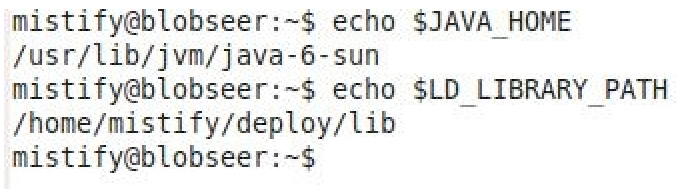
2.2 Using BlobSeer

BlobSeer is pretty easy to use. There are a few steps to follow in order to have a fully functional program.

1. Verify if there are installed basic applications and libraries like: gcc, g++, java (it is not necessary to be the last version, but it must be one of the latest), cmake and svn. Because all of them are in the repository it is simple to install.

apt-get install *name*

2. The next step is to download BlobSeer. The BlobSeer repository is at https://gforge.inria.fr/scm/viewvc.php/?root=blobseer . It can be downloaded with command: svn checkout svn://scm.gforge.inria.fr/svn/blobseer/tags/release-**V** , where **V** will be replaced with the last version available. When this work has been written the last version was "0.3.3 "
3. Next, export the path to JAVA_HOME and LD_LIBRARY_PATH. To do that the .bashrc file must be changed, adding the following command lines: export JAVA_HOME= "where jdk is installed" and export LD_LIBRARY_PATH = $HOME/deploy/lib. This is the place were all the project libraries will be. If everything was configured well, after the command source .bashrc, the echo $JAVA_HOME should return the path to the jdk and echo $LD_LIBRARY_PATH should return the path to the lib directory.



```
mistify@blobseer:~$ echo $JAVA_HOME
/usr/lib/jvm/java-6-sun
mistify@blobseer:~$ echo $LD_LIBRARY_PATH
/home/mistify/deploy/lib
mistify@blobseer:~$
```

4. The libraries BobSeer depends (Boost, Libconfig and Berkley DB) on must be installed. Boost can be downloaded from http://www.boost.org/ . It imperative to download and install the last version. Older versions will not work. After the download was finished the istalation should be done like that:

/bootstrap.sh --prefix=$HOME/deploy –with libraries=system,thread,serialization,filesystem,date_time -libdir=$HOME/deploy/lib

After that you should be asked to run ./bjam and ./bjam install to complete the instalation. It the installation was successful, in the $HOME/deploy/lib directory should appear the new libraries.

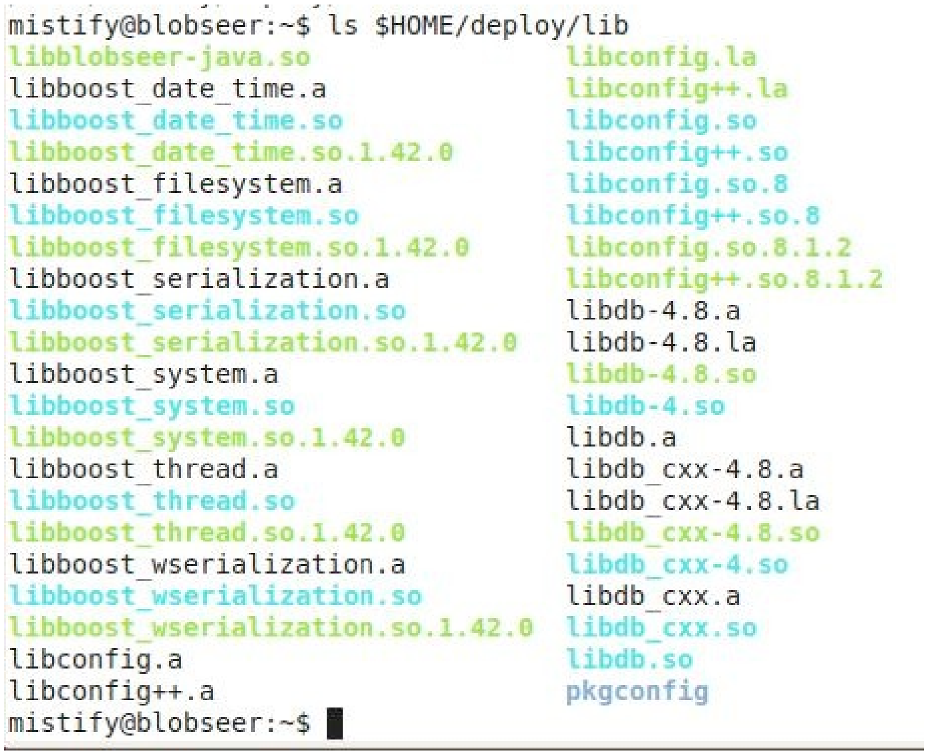
The Libconfig library can be downloaded from http://www.hyperrealm.com/libconfig/. The instalation is very simple, after downloading it, run

./configure –prefix=$HOME/deploy then make and make install

Berkley DB is found at http://www.oracle.com/technology/software/products/berkeley-db/index.html To install it, in the build_unix folder is the script it should be run.

../dist/configure --prefix=$HOME/deploy –enable-cxx then make and make install

If everything went well, ls $HOME/deploy/lib should look like this



```
mistify@blobseer:~$ ls $HOME/deploy/lib
libblobseer-java.so                   libconfig.la
libboost_date_time.a                  libconfig++.la
libboost_date_time.so                 libconfig.so
libboost_date_time.so.1.42.0          libconfig++.so
libboost_filesystem.a                 libconfig.so.8
libboost_filesystem.so                libconfig++.so.8
libboost_filesystem.so.1.42.0         libconfig.so.8.1.2
libboost_serialization.a              libconfig++.so.8.1.2
libboost_serialization.so             libdb-4.8.a
libboost_serialization.so.1.42.0      libdb-4.8.la
libboost_system.a                     libdb-4.8.so
libboost_system.so                    libdb-4.so
libboost_system.so.1.42.0             libdb.a
libboost_thread.a                     libdb_cxx-4.8.a
libboost_thread.so                    libdb_cxx-4.8.la
libboost_thread.so.1.42.0             libdb_cxx-4.8.so
libboost_wserialization.a             libdb_cxx-4.so
libboost_wserialization.so            libdb_cxx.a
libboost_wserialization.so.1.42.0     libdb_cxx.so
libconfig.a                           libdb.so
libconfig++.a                         pkgconfig
mistify@blobseer:~$
```

5. Building BlobSeer: In order to compile the project type

cmake -G "Unix Makefiles" then make

6. In order to log in to the system without requiring any password it is necessary to build a pair of keys. First install ssh server:

   sudo apt-get install openssh-server

then, generate the keys:

ssh-keygen –t rsa

Last, copy the public key in the authorized_keys file from the $HOME/.ssh folder and add StrictHostKeyChecking no in the config file from the same directory

7. Before running, a path to BlobSeer should be added in .bashrc file

```
export BLOBSEER_HOME=$HOME/release-0.3.3
```

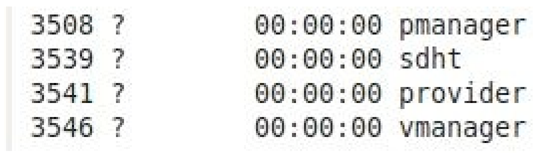
8. From $BLOBSEER_HOME run

```
./scripts/local-deploy.sh
```

If everything went well you should get the message "All done" and the command ps -A | grep provider should return something

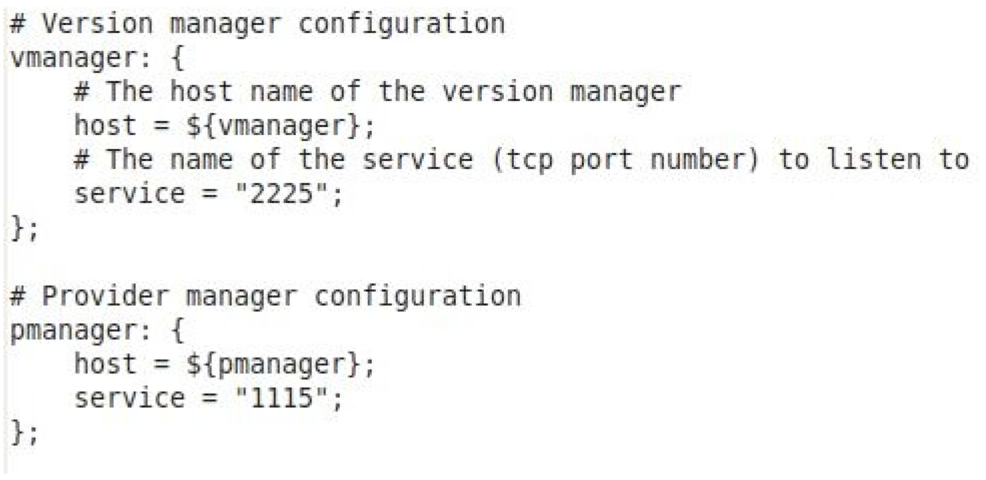
9. To create a blob run ./test/create_blob test/test.cfg 65536 1 from the same folder. The 65536 value represents the page size (in this case 64 Kb) and the 1 value represents the replica count To write to the new blob run ./test/test W 1 test/test.cfg if you get "result: 1" it meand the write was successful. The 1 value is the version number. If we run the create_blob command again and we want to write in the second blob we will change this value to 2. To read what has been written run ./test/test R 1 test/test.cfg . To append type ./test/test A 1 test/test.cfg .

The local-deploy script starts all the managers needed in order for BlobSeer to work: pmanager, vmanager, provider and sdht.



```
3508 ?        00:00:00 pmanager
3539 ?        00:00:00 sdht
3541 ?        00:00:00 provider
3546 ?        00:00:00 vmanager
```

This script is run when we want to deploy on localhost, that means all the managers are working on our computer. If we would want to access them over the network we would change the blobseer_template.cfg file. It should look like this:



```
# Version manager configuration
vmanager: {
    # The host name of the version manager
    host = ${vmanager};
    # The name of the service (tcp port number) to listen to
    service = "2225";
};

# Provider manager configuration
pmanager: {
    host = ${pmanager};
    service = "1115";
};
```

For localhost, the host variable is changed to:

host = "localhost"; or host = "127.0.0.1";

We can manually deploy each oane by running the pmanager script from the pmanager folder, the vmanager script from vmanager folder, the provider script from provider folder and the sdht script form the provider folder.

The communication between the managers and the BlobSeer client is done using TCP sockets.

3. Web Services

3.1 What are Web Services?

Web services are typically application programming interfaces (API) or web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. The term refers to clients and servers that communicate over the Hypertext Transfer Protocol (HTTP) protocol used on web. The basic Web services platform is XML + HTTP. XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions. The HTTP protocol is the most used Internet protocol. Web services platform elements are: SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration), WSDL (Web Services Description Language). Web services use XML to code and to decode data, and SOAP to transport it (using open protocols). Or more simple: SOAP is a protocol for accessing a Web Service (a communication protocol).SOAP is also a format for sending messages, designed to communicate via Internet. It is platform and language independent (based on XML, so it is extensible). SOAP is a W3C standard. Basically SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. WSDL is an XML-based language for locating and describing Web services. It is also a W3C standard. UDDI is a directory for storing information about web services, it is described by WSDL and it communicates via SOAP.

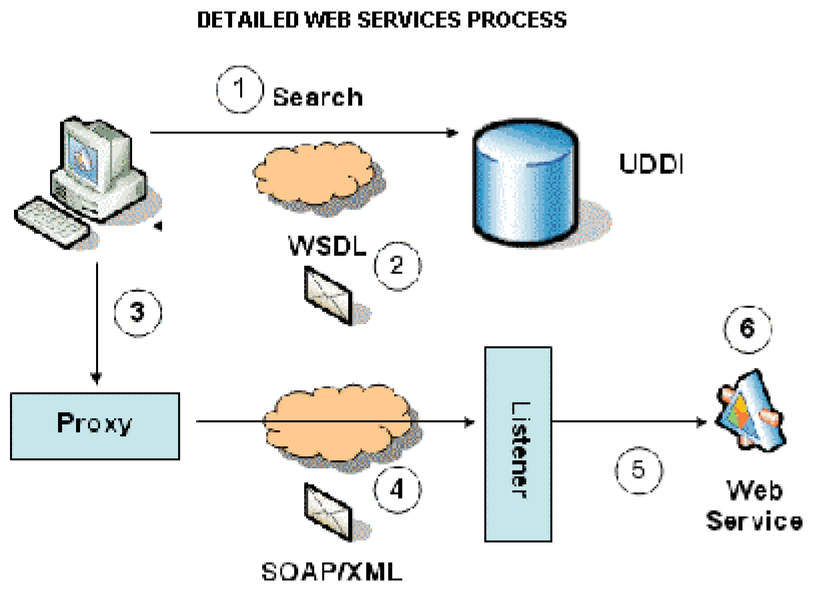


Figure 1: The process flow of a Web service

The process flow of a Web service is:

1. Discovery - Search UDDI site(s) for the proper Web service.
2. Description - A description of the selected Web service is returned to the client application as a Web Services Description Language (WSDL) file.
3. Proxy creation - A local proxy to the remote service is created.
4. Soap Message Creation - a Soap/XML message is created and sent to the URL specified in the WSDL file.
5. Listener - A Soap listener at the host site receives the call and interprets it for the Web Service.
6. The Web service performs its function, and returns the result back to the client, via the listener and the proxy.

In conclusion, the best way for clients cu communicate with servers via HTTP is by web services. Clients can access only services already deployed, it is called invoking a service. The services must be deployed on a server so that clients can be see and access (invoke) the service.

3.2 What is Axis2?

Apache Axis2 is a core engine for Web services. Implementations of Axis2 are available in Java and C.

Axis2 not only provides the capability to add Web services interfaces to Web applications, but can also function as a standalone server application.

Apache Axis2 is more efficient, more modular and more XML-oriented than other platforms. It is carefully designed to support the easy addition of plug-in "modules" that extend their functionality for features such as security and reliability.

Supported Specifications:

- SOAP 1.1 and 1.2

- Message Transmission Optimization Mechanism (MTOM), XML Optimized Packaging (XOP) and SOAP with Attachments

- WSDL 1.1, including both SOAP and HTTP bindings

- WS-Addressing (submission and final)

- WS-Policy

- SAAJ 1.1

Transports:

- HTTP

- SMTP

- JMS

- TCP

Supported Data Bindings:

- Axis Data Binding (ADB)

- XMLBeans

- JibX

- JaxBRI (Experimental)

Axis2 has a new high performance object module: AXIOM (AXIS Object Model), it is an asynchronous and message based model and it have flexible data binging and REST support.

AXIOM is the heart of AXIS2. The model is guided by the following ideas:

- Only build a tree when requested
- Only build as much as requested
- Only parse the stream when needed
- Only parse as much as is needed

With Axis2 the deployment is very easy. Create an AAR (Axis2ARchive) and copy it into the repository folder.

Writing a new Web Service with Axis2 involve four steps:

1. Write the Implementation Class
2. Write a services.xml file to explain the Web Service
3. create a *.aar archive (Axis Archive) for the Web Service
4. Deploy the Web Service

Step1:

Provides an implementation class that provides the logic for the Web Service. The signature of the methods can have only one parameter of type OMElement.

```
public class MyService{
    public void ping(OMElement element){
     ......
    }
    public OMElement echo(OMElement element){
     ......
    }
}
```

Step2:

Each Web Service deployed in Axis2 needs a "services.xml" containing the configurations.

The structure of the xml file is: First we have the <description> tag, where you can write a

```
<service >
    <description>
        This is a sample Web Service with two operations, echo and ping.
    </description>
    <parameter name="ServiceClass" locked="false">userguide.example1.MyService</parameter>
    <operation name="echo">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
        <actionMapping>urn:echo</actionMapping>
    </operation>
     <operation name="ping">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver"/>
        <actionMapping>urn:ping</actionMapping>
    </operation>
 </service>
```

small description of the service. Then we have the service class into the <parameter> tag. After that there are the operations (the <operation> tag). Every operation must have a corresponding MessageReceiver class. When Axis2 engine receives a message, after the message is being processed by the handlers, it will be handed over to a MessageReceiver.

In the services.xml file you can have a <serviceGroup> tag witch includes a group of services.

Step3:

Axis2 uses ".aar" (Axis Archive) file as the deployment package for Web Services. You can create the archive by running the ant command (it is necessary to have an makefile file)

Step 4:

Deploying the service is just a matter of dropping the ".aar" in to "services" directory that can be found in the "\webapps\axis2\WEB-INF" of your servlet container. Once these steps are completed, start the servlet container (if you have not already started) and check the link http://localhost:8080/axis2/index.jsp and see whether your service is deployed properly.

There are cases when we want an Web Service to communicate with another Web Service. Web Services communicate using a Reliable Messaging Model. The primary goal of this specification is to create a modular mechanism for reliable message delivery. It defines a messaging protocol to identify, track, and manage the reliable delivery of messages between exactly two parties, a source and a destination. It also defines a SOAP binding which is required for interoperability. Additional bindings may be defined.

This mechanism is extensible allowing additional functionality, such as security, to be tightly integrated. This specification integrates with and compliments the WS-Security, WS-Policy, and other Web services specifications. Combined, these allow for a broad range of reliable, secure messaging options.

The Reliable Messaging model provides the guarantee that messages sent by the initial sender will be delivered to the ultimate receiver

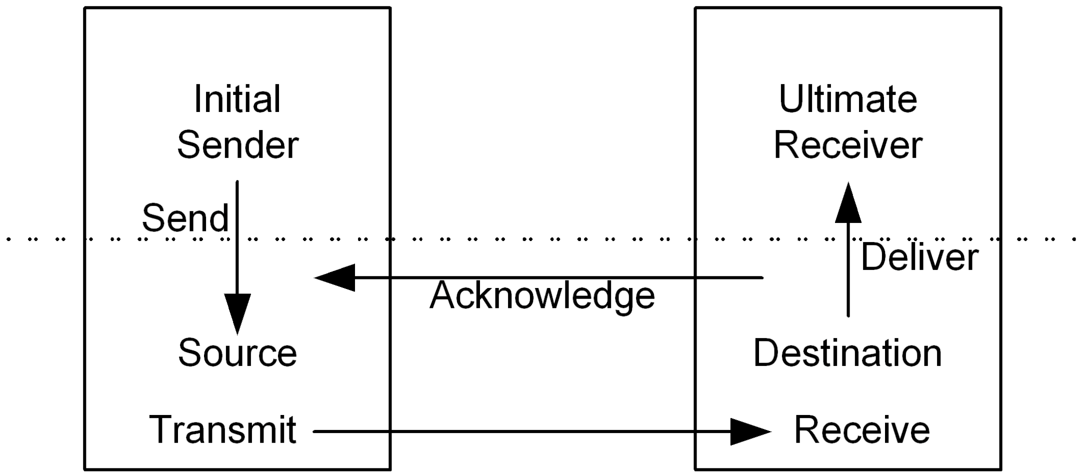


fig3: Reliable Massaging model

The diagram above illustrates the entities and events in a simple reliable message exchange. First, the Initial Sender sends a message for reliable delivery. The Source accepts the message and transmits it one or more times. After receiving the message the Destination acknowledges it. Finally, the Destination delivers the message to the Ultimate Receiver. The exact roles the entities play and the complete meaning of the events will be defined throughout this specification.

Endpoints which implement the WS-ReliableMessaging protocol provide delivery assurances for the delivery of messages sent from the initial sender to the ultimate receiver. The protocol supports the endpoints in providing these delivery assurances. It is the responsibility of the source and destination to fulfill the delivery assurances in the Sequence's policy declarations, or raise an error and terminate the Sequence. These delivery assurances begin with a successful send by the Initial Sender and end with a successful delivery. There are four basic delivery assurances that endpoints can provide:

AtMostOnce Messages will be delivered at most once without duplication or an error will be raised on at least one endpoint. It is possible that some messages in a sequence may not be delivered.

AtLeastOnce Every message sent will be delivered or an error will be raised on at least one endpoint. Some messages may be delivered more than once.

ExactlyOnce Every message sent will be delivered without duplication or an error will be raised on at least one endpoint. This delivery assurance is the logical "and" of the two prior delivery assurances.

InOrder Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions.

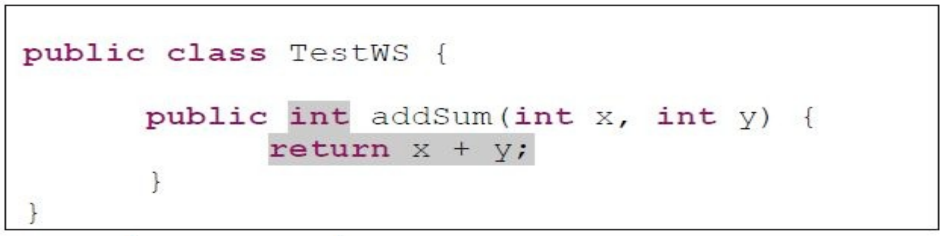
3.3 Services

Any Web Service has a Client side and a Service side. The Service side is that part of an web service that is responsible with the business logic part. Basically the Client sends an request via HTTP or other transport protocols for web and the Service resolves that request. The Service of an Web Service can act like a Client for another Services... it can access another Service in order to get all the information it needs to resolve the request.

We have 2 ways to create an web service: starting with the WSDL or starting with the classes and generate the xml. In this work we've studied the first one.

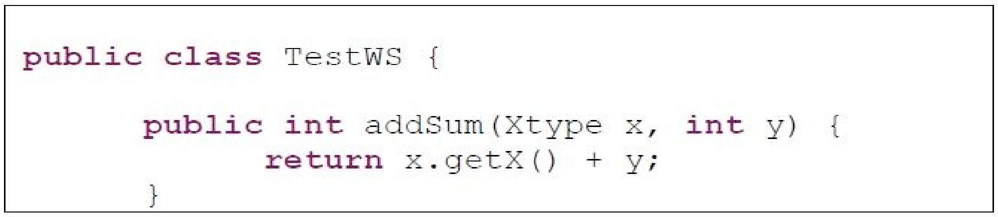
3.3.1 Create a service

We create a Java class to provide the service we want. For example, accept two integers and return the sum of the two. The source file can be like this:



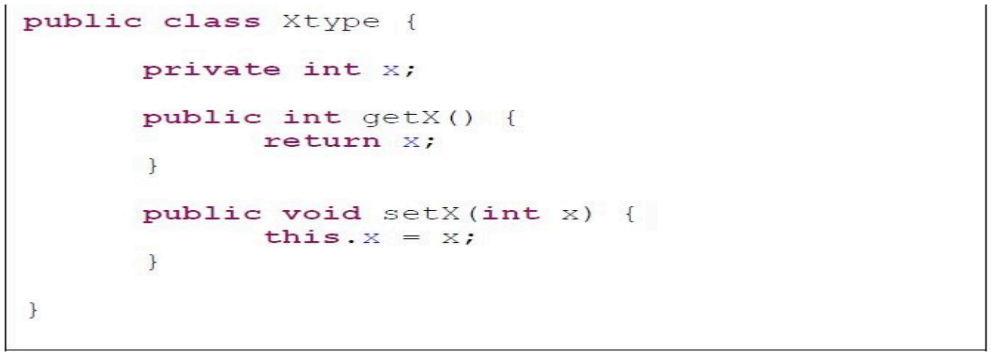
```
public class TestWS {

        public int addSum(int x, int y) {
                return x + y;
        }
}
```

Axis2 needs the according wsdl:type so instead of int we will put Xtype.



```
public class TestWS {

        public int addSum(Xtype x, int y) {
                return x.getX() + y;
        }
```

Now we will need to create the class Xtype:



```
public class Xtype {

        private int x;

        public int getX() {
                return x;
        }

        public void setX(int x) {
                this.x = x;
        }

}
```

Let's take a look at an example:

Here you will see how to develop a StockQuoteService with two operations, an In-Only subscribe() operation and an In-Out getQuote() operation. The subscribe() operation will subscribe for hourly quotes for the given symbol and getQuote() will get the current quote for the given symbol.

```
package stock;
import org.apache.axis2.om.OMAbstractFactory;
import org.apache.axis2.om.OMElement;
import org.apache.axis2.om.OMFactory;
import org.apache.axis2.om.OMNamespace;

public class StockQuoteService {

  public void subscribe(OMElement in){
    String symbol = in.getText();
    System.out.println("Subscription request for symbol ="+symbol);
    // put the actual subscribe code here...
  }


  public OMElement getQuote(OMElement in){

    // Get the symbol from request message
    String symbol = in.getText();

    int quote = 0;
    if(symbol.equals("IBM")){
      quote = 100;
    }
    // Put more quotes here ...

    // Create response
    OMFactory fac = OMAbstractFactory.getOMFactory();
    OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
    OMElement resp = fac.createOMElement("getQuoteResponse", omNs);
    resp.setText(String.valueOf(quote));
    return resp;
  }
}
```

3.3.2 Deploy a service

We will need to create an .xml file from the java sources we have because In Axis2, service deployment information is contained in the services.xml file.

Listing 2. Services.xml

```
<service name="StockQuoteService">
  <parameter name="ServiceClass" locked="xsd:false">
    stock.StockQuoteService
  </parameter>

  <operation name="getQuote">
    <messageReceiver
       class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
  </operation>

  <operation name="subscribe">
    <messageReceiver
       class="org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver"/>
  </operation>
</service>
```

The name attribute of the service defines the name of the service. Axis2 uses the name of the service to create the endpoint address of the service as http://localhost:<port>/axis2/services/<nameofservice>. So for the StockQuoteService, the service endpoint will be http://localhost:<port>/axis2/services/StockQuoteService. The ServiceClass parameter specifies the service implementation class.

Each <operation> element defines the configuration of an operation in the service. The name attribute of the <operation> should be set to the name of the method in the service implementation class. The messageReceiver element defines the message receiver to be used for handling this operation. Axis2 provides two built-in MessageReceivers for In-Only and In-Out operations without data binding; org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver for the In-Only operation and org.apache.axis2.receivers.RawXMLINOutMessageReceiver for the In-Out operation. When no messageReceiver is specified, Axis2 will try to use org.apache.axis2.receivers.RawXMLINOutMessageReceiver as the default. The RAWXML message receivers mentioned above pass the content of the <Body> of the incoming SOAP message to the service implementation as an OMElement (OMElement is the AXIOM abstraction of an XML element). The operation should return the XML content that goes in the <Body> element of the SOAP response as an OMElement. This explains why subscribe() and getQuote() operations take and return OMElement.

The services.xml can also contain multiple services grouped as a servicegroup.

Axis2 provides two mechanisms for packing and deploying JAX-WS services:

The service may be packaged and deployed as an AAR, just like any other service within Axis2. Like with all AARs, a services.xml file containing the relevant metadata is required for the service to deploy correctly.

The service may be packaged in a jar file and placed into the servicejars directory. The JAXWSDeployer will examine all jars within that directory and deploy those classes that have JAX-WS annotations which identify them as Web services.

We will use the easiest way: the first way: first type ant to compile and to create an AAR file an them copy the AAR file into the webapps folder.

3.4 Clients

The Client side of an Web Service is responsible with taking the request from the Client and transferring it to the Server and also to take the Server reply and transferring it to the Client.

The Java API for XML-Based Web Services (JAX-WS) Web service client programming model supports both the Dispatch client API and the Dynamic Proxy client API. The Dispatch client API is a dynamic client programming model, whereas the static client programming model for JAX-WS is the Dynamic Proxy client. The Dispatch and Dynamic Proxy clients enable both synchronous and asynchronous invocation of JAX-WS Web services.

Dispatch client: Use this client when you want to work at the XML message level or when you want to work without any generated artifacts at the JAX-WS level.

Dynamic Proxy client: Use this client when you want to invoke a Web service based on a service endpoint interface.

We will study a Dynamic Proxy client.

A client can invoke a service in many ways:

The nature of Web service invocation is decided by the MEP (a template that establishes a pattern for the exchange of messages between SOAP nodes), the transport protocol, and the synchronous and / or asynchronous behavior of the client API. Axis2 currently supports In-Only and In-Out MEPs defined by WSDL 2.0. The Axis2 client API supports synchronous and asynchronous invocation of services. Asynchronous behavior is provided at the API level and transport level when invoking In-Out operations. At the API level it is achieved by callback and uses a single transport connection for transferring both request and response (for example, request and response transferred over a single HTTP connection). In transport level, separate transport connections are used for sending the request and receiving the response -- for example, when you use SMTP for transport.

3.4.1 Create a client

The static client programming model for JAX-WS is the called the Dynamic Proxy client. The Dynamic Proxy client invokes a Web service based on a Service Endpoint Interface (SEI) which must be provided. The Dynamic Proxy client is similar to the stub client in the Java API for XML-based RPC (JAX-RPC) programming model. Although the JAX-WS Dynamic Proxy client and the JAX-RPC stub client are both based on the Service Endpoint Interface (SEI) that is generated from a WSDL file , there is a major difference. The Dynamic Proxy client is dynamically generated at run time using the Java 5 Dynamic Proxy functionality, while the JAX-RPC-based stub client is a non-portable Java file that is generated by tooling. Unlike the JAX-RPC stub clients, the Dynamic Proxy client does not require you to regenerate a stub prior to running the client on an application server for a different vendor because the generated interface does not require the specific vendor information.

The Dynamic Proxy instances extend the java.lang.reflect.Proxy class and leverage the Dynamic Proxy function in the base Java Runtime Environment Version 5. The client application can then provide an interface that is used to create the proxy instance while the runtime is responsible for dynamically creating a Java object that represents the SEI.

JAX-WS provides a new dynamic Dispatch client API that is more generic and offers more flexibility than the existing Java API for XML-based RPC (JAX-RPC)-based Dynamic Invocation Interface (DII). The Dispatch client interface, javax.xml.ws.Dispatch, is an XML messaging oriented client that is intended for advanced XML developers who prefer to work at the XML level using XML constructs. To write a Dispatch client, you must have expertise with the Dispatch client APIs, the supported object types, and knowledge of the message representations for the associated WSDL file.

The Dispatch API can send data in either PAYLOAD or MESSAGE mode. When using the PAYLOAD mode, the Dispatch client is only responsible for providing the contents of the <soap:Body> and JAX-WS includes the input payload in a <soap:Envelope> element. When using the MESSAGE mode, the Dispatch client is responsible for providing the entire SOAP envelope.

The Dispatch client API requires application clients to construct messages or payloads as XML and requires a detailed knowledge of the message or message payload. The Dispatch client can use HTTP bindings when using Source objects, Java Architecture for XML Binding (JAXB) objects, or data source objects. The Dispatch client supports the following types of objects:

javax.xml.transform.Source: Use Source objects to enable clients to use XML APIs directly. You can use Source objects with SOAP and HTTP bindings.

JAXB objects: Use JAXB objects so that clients can use JAXB objects that are generated from an XML schema to create and manipulate XML with JAX-WS applications. JAXB objects can only be used with SOAP and HTTP bindings.

javax.xml.soap.SOAPMessage: Use SOAPMessage objects so that clients can work with SOAP messages. You can only use SOAPMessage objects with SOAP version 1.1 or SOAP version 1.2 bindings.

javax.activation.DataSource: Use DataSource objects so that clients can work with Multipurpose Internet Mail Extension (MIME) messages. Use DataSource only with HTTP bindings.

The Dispatch API uses the concept of generics that are introduced in Java Runtime Environment Version 5. For each of the invoke() methods on the Dispatch interface, generics are used to determine the return type.

3.4.2 Deploy a client

A JAX-WS client may be started from the command line like any other Axis2-based client, including through the use of the axis2 shell scripts in the bin directory of the installed runtime.

Invoking JAX-WS Web services asynchronously:

When calling Web services asynchronously, both the callback model and the polling model are available on the Dispatch client and the Dynamic Proxy client.

Using the callback asynchronous invocation model:

To implement an asynchronous invocation that uses the callback model, the client provides an AsynchHandler callback handler to accept and process the inbound response object. The client callback handler implements the javax.xml.ws.AsynchHandler interface, which contains the application code that is executed when an asynchronous response is received from the server. The javax.xml.ws.AsynchHandler interface contains the handleResponse(java.xml.ws.Response) method that is called after the run time has received and processed the asynchronous response from the server. The response is delivered to the callback handler in the form of a javax.xml.ws.Response object. The response object returns the response content when the get() method is called. Additionally, if an error was received, then an exception is returned to the client during that call. The response method is then invoked according to the threading model used by the executor method, java.util.concurrent.Executor on the client's java.xml.ws.Service instance that was used to create the Dynamic Proxy or Dispatch client instance. The executor is used to invoke any asynchronous callbacks registered by the application. Use the setExecutor and getExecutor methods to modify and retrieve the executor configured for your service.

Using the polling asynchronous invocation model:

Using the polling model, a client can issue a request and receive a response object that can subsequently be polled to determine if the server has responded. When the server responds, the actual response can then be retrieved. The response object returns the response content when the get() method is called. The client receives an object of type javax.xml.ws.Response from the invokeAsync method. That Response object is used to monitor the status of the request to the server, determine when the operation has completed, and to retrieve the response results.

Using an asynchronous message exchange:

By default, asynchronous client invocations do not have asynchronous behavior of the message exchange pattern on the wire. The programming model is asynchronous; however, the exchange of request or response messages with the server is not asynchronous. To use an asynchronous message exchange, the org.apache.axis2.jaxws.use.async.mep property must be set on the client request context with a boolean value of true. When this property is enabled, the messages exchanged between the client and server are different from messages exchanged synchronously. With an asynchronous exchange, the request and response messages have WS-Addressing headers added that provide additional routing information for the messages. Another major difference between asynchronous and synchronous message exchange is that the response is delivered to an asynchronous listener that then delivers that response back to the client. For asynchronous exchanges, there is no timeout that is sent to notify the client to stop listening for a response. To force the client to stop waiting for a response, issue a Response.cancel() method on the object returned from a polling invocation or a Future.cancel() method on the object returned from a callback invocation. The cancel response does not affect the server when processing a request.

4. Cloud

Cloud computing is a technology that uses the internet and central remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. This technology allows for much more efficient computing by centralizing storage, memory, processing and bandwidth.

A simple example of cloud computing is Yahoo email or Gmail etc. You don't need a software or a server to use them. All a consumer would need is just an internet connection and you can start sending emails. The server and email management software is all on the cloud (internet) and is totally managed by the cloud service provider Yahoo , Google etc. The consumer gets to use the software alone and enjoy the benefits.

Cloud computing is broken down into three segments: "applications," "platforms," and "infrastructure." Each segment serves a different purpose and offers different products for businesses and individuals around the world.

Cloud Computing Segments

Applications: It's all On Demand

So far the applications segment of cloud computing is the only segment that has proven useful as a business model. The Cloud Wars: $100 Billion at Stake, Published by Merrill Lynch, May 7, 2008. By running business applications over the internet from centralized servers rather than from on-site servers, companies can cut some serious costs. Furthermore, while avoiding maintenance costs, licensing costs and the costs of the hardware required to run servers on-site, companies are able to run applications much more efficiently from a computing standpoint.

On Demand software services come in a few different varieties which vary in their pricing scheme and how the software is delivered to the end users. In the past, the end-user would generally purchase a license from the software provider and then install and run the software directly from on-premise servers. Using an On-Demand service however, the end-user pays the software provider a subscription fee for the service. The software is hosted directly from the software providers' servers and is accessed by the end user over the internet. While this is the most common platform for On Demand software services, there are also some slightly different offerings which can be described as a hybrid of these two platforms. For instance, a program through which the end user pays a license fee, but then accesses the software over the internet from centralized servers is considered a hybrid service.

4.1 Approaches for Cloud Services

Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). The name cloud computing was inspired by the cloud symbol that's often used to represent the Internet in flowcharts and diagrams.

A cloud service has three distinct characteristics that differentiate it from traditional hosting. It is sold on demand, typically by the minute or the hour; it is elastic -- a user can have as much or as little of a service as they want at any given time; and the service is fully managed by the provider (the consumer needs nothing but a personal computer and Internet access). Significant innovations in virtualization and distributed computing, as well as improved access to high-speed Internet and a weak economy, have accelerated interest in cloud computing.

A cloud can be private or public. A public cloud sells services to anyone on the Internet. (Currently, Amazon Web Services is the largest public cloud provider.) A private cloud is a proprietary network or a data center that supplies hosted services to a limited number of people. When a service provider uses public cloud resources to create their private cloud, the result is called a virtual private cloud. Private or public, the goal of cloud computing is to provide easy, scalable access to computing resources and IT services.

Infrastructure-as-a-Service like Amazon Web Services provides virtual server instances with unique IP addresses and blocks of storage on demand. Customers use the provider's application program interface (API) to start, stop, access and configure their virtual servers and storage. In the enterprise, cloud computing allows a company to pay for only as much capacity as is needed, and bring more online as soon as required. Because this pay-for-what-you-use model resembles the way electricity, fuel and water are consumed, it's sometimes referred to as utility computing.

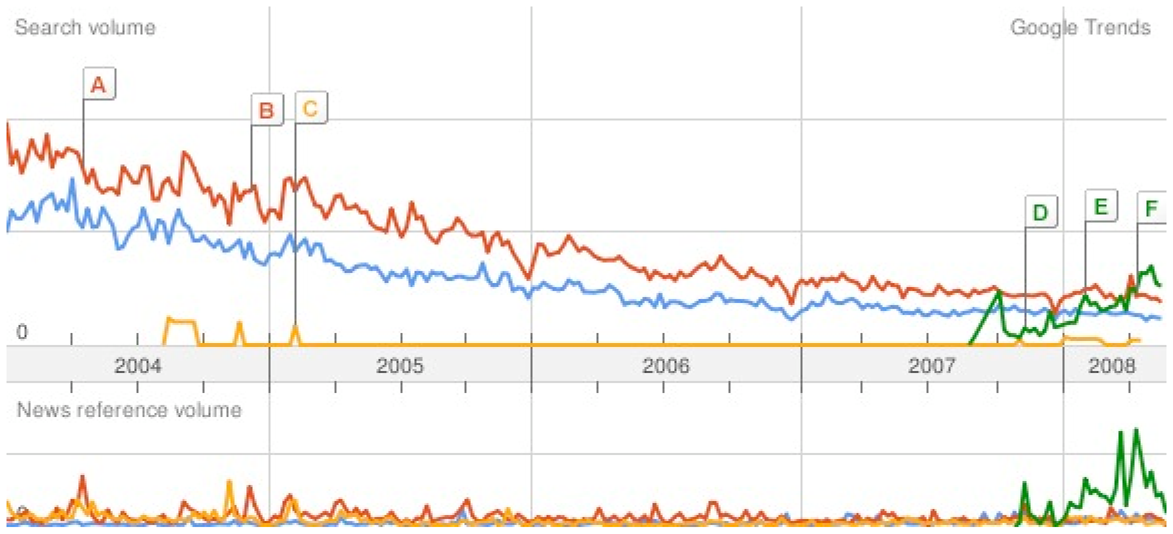
Actors: Ips (Infrastructure Providers) manage a large set of computing resources, such as storing and processing capacity. Through virtualization, they are able to split, assign and dynamically resize these resources to build ad-hoc systems as demanded by customers, the Sps (Service Providers). They deploy the software stacks that run their services.

Platform-as-a-service in the cloud is defined as a set of software and product development tools hosted on the provider's infrastructure. Developers create applications on the provider's platform over the Internet. PaaS providers may use APIs, website portals or gateway software installed on the customer's computer. Force.com, (an outgrowth of Salesforce.com) and GoogleApps are examples of PaaS. Developers need to know that currently, there are not standards for interoperability or data portability in the cloud. Some providers will not allow software created by their customers to be moved off the provider's platform.

Scenario: Cloud systems can offer an additional abstraction level: instead of supplying a virtualized infrastructure, they can provide the software platform where systems run on. The sizing of the hardware resources demanded by the execution of the services is made in a transparent manner.

In the software-as-a-service cloud model, the vendor supplies the hardware infrastructure, the software product and interacts with the user through a front-end portal. SaaS is a very broad market. Services can be anything from Web-based email to inventory control and database processing. Because the service provider hosts both the application and the data, the end user is free to use the service from anywhere.

There are services of potential interest to a wide variety of users hosted in Cloud systems. This is an alternative to locally run applications. An example of this is the online alternatives of typical office applications such as word processors



- distributed computing * grid computing * utility computing * cloud computing

fig4: Google Trends

Using Cloud Computing

Cloud computing delivers flexible applications, web services, and IT infrastructure as a service, over the Internet, using a utility pricing model. Cloud computing allows businesses to instantly scale their technology requirements to meet new demands. The Cloud is a cost-effective approach to technology because businesses don't need to make usage predictions, upfront capital investments, or over-purchase hardware or software to meet the demands of peak periods.

Examples of this include pharmaceutical companies using the Cloud to perform drug research analysis and online retailers trying a new strategy for surges in seasonal website traffic.

Important to notice is that the participants are consumers and producers of Cloud Computing services. As such, they represent also a very complex system.

Computing clouds are huge aggregates of various grids (academic, commercial), computing clusters and supercomputers. They are used by a huge number of people either as users (300 million users of Microsoft’s Live) or developers (330.000 application developers of Amazon EC2).

As hundreds of millions of users are connecting to Google daily (Google users have created 20 Petabytes of data) or using eBay services (1 Terabyte of logs is created by embay users in 2000 application servers per day) they are creating torrents of data and information. Thus, we can conclude
that C2 denotes massive participation, collaboration and content creation by 100s of millions of users on an omnipresent, always available mega-structure. This poses big challenges, but offers some unprecedented opportunities.

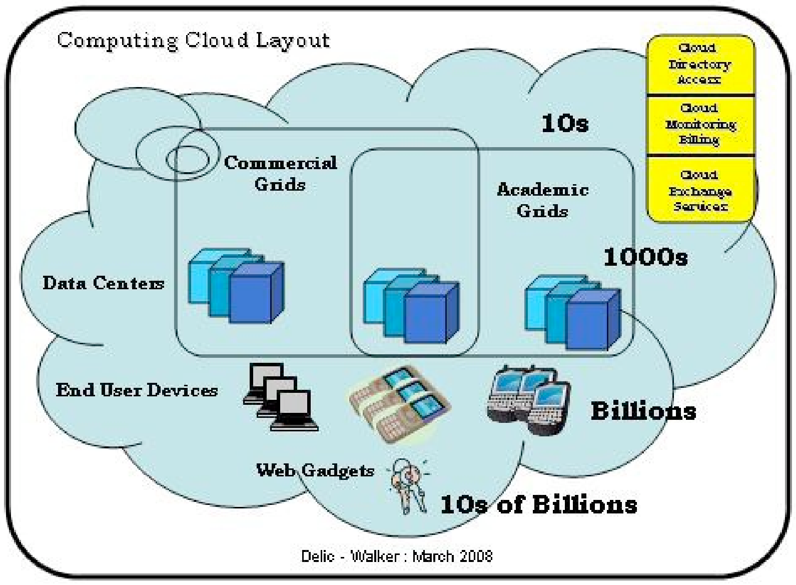


Fig5: Cloud computing Layout

Cloud Computing is associated with a new paradigm for the provision of computing infrastructure. This paradigm shifts the location of this infrastructure to the network to reduce the costs associated with the management of hardware and software resources The Cloud is drawing the attention from the Information and Communication Technology (ICT) community, thanks to the appearance of a set of services with common characteristics, provided by important industry players. However, some of the existing technologies the Cloud concept draws on (such as virtualization, utility computing or distributed computing) are not new The variety of technologies in the Cloud makes the overall picture confusing.

Advantages and disadvantages:

Cloud as a term is new, but the concepts and requisite technologies have been evolving for years (many years in some cases). Cloud computing continues to emerge as a game-changing technology, with high adoption rates and investment. Gartner Research predicts that by 2012, 80% of Fortune 1000 enterprises will be paying for some form of cloud computing services. Cloud computing is here to stay.

Public clouds are fundamentally multi-tenant to justify the scale and economics of the cloud. As such, security is a common concern. Whereas the traditional security perimeter is a network firewall, the cloud security perimeter now becomes the hypervisor and/or underlying cloud application. So far, security in the cloud has been good, but this is very cloud-dependent and requires a solid design and operational rigor that prioritizes security. Also, handing your data and systems to someone else requires proper internal controls to ensure that not just anyone has access. Be sure to ask potential cloud computing providers about security from technical, operational, and control perspectives, as well as what experience they have being stewards of customer systems and data. If the public cloud is fundamentally not secure enough, consider an on-premise cloud, virtual private cloud, or some sort of hybrid cloud solution that allows you to maintain the level of security you require.

No system has 100% uptime, and neither does the Cloud. Given the scale, however, cloud computing services are typically designed to provide high redundancy and availability. While this same level of redundancy/availability is possible to achieve in-house or with dedicated hosting, it's generally cost prohibitive except for the most critical systems. The cloud enables a higher level of reliability at a fraction of the cost.

There are different types of clouds and use cases. In many instances, performance is higher in the cloud because there is more available capacity and scalability. In other cases (most notably running a database server), performance may be less than a traditional server. It's best to benchmark your application in the cloud to determine any performance impact (good or bad). If performance is an issue, consider a hybrid solution that allows you to synergize the best of both worlds: the scalability and cost efficiencies of cloud computing and the performance of dedicated servers.

There are different types of clouds that offer different levels of customization and flexibility. Clouds that implement standard technology stacks and are participating in cloud standardization efforts are the best bet to enable application mobility. Traction for open clouds is gaining momentum and the future will involve federation between public-to-public as well as public to on-premise/hosted private clouds.

Cloud can be simple or complex. It depends on the types of Cloud you want to access. Many clouds simplify management and involve little to no change in your application to move it to the cloud. Other clouds offer more power and control, but involve a change in application architecture. Simplicity and control are often at odds and the cloud is no different. Depending on your needs, the cloud can offer you a good balance.

Cloud computing has huge economies of scale that get passed on to the consumer. In addition, cloud computing transfers what is typically CapEx (large upfront expenditures) into OpEx (ongoing operational costs) and enables pricing to be commensurate with usage. If pricing variability and budgeting are a concern, consider a pricing plan that offers a predictable price. Also, don't just look at raw cost. Generally, best value solutions are superior to lowest cost. Consider all the factors including support, customer service, reputation, reliability, etc. when measuring value.

Cloud is not too hard to integrate with other applications. Many applications are stand-alone and can be moved independent of other existing systems. For integrated applications that are service oriented, integration is relatively simple. For non-service oriented applications that require tight integration, hybrid solutions are designed to simplify integration with the cloud. As with all integration considerations, latency is likely a concern, so transparency about where your cloud application lives is important.

It is not only for enterprises. The benefits of cloud computing apply equally to enterprises as they do to SMBs (small businesses), startups and consumers. Since enterprises are typically more risk averse, new technologies are generally adopted by small business first. That said, overall cloud adoption rates are increasing substantially and we are seeing enterprise adoption today. Expect to see a significant inflection point in the next several years where cloud is a standard enterprise fixture.

Not all applications are suitable for cloud computing. While the Cloud is here to stay, it will not replace traditional hosting or on-premise deployments, but rather complement them. There will always be situations where security requirements, flexibility, performance or control will preclude the cloud. In those cases, a hybrid solution involving both cloud and either traditionally hosted or on-premise servers may make sense. Beware of vendors who promote pure cloud for all applications. Instead, look for a cloud provider who can offer you hosting options that best fit your application needs. If management services are important to you, consider the ramifications of a move to the cloud and look for a cloud provider that will provide the level of support and service necessary for you to be successful.

4.2 Why Ubuntu Cloud?

The main reasons are:

1. Providing a self-service IT capability that enables new applications to be rapidly deployed whenever needed.
2. With Ubuntu Enterprise Cloud offering the same APIs as the dominant public cloud offering, Amazon EC2, we can build your applications to run on either platform.
3. We can create the initial cloud infrastructure in minutes and grow it over time. The current record stands at 25 minutes to have a private cloud running.
4. Building a private cloud enables existing hardware and network infrastructure to be used. Providing the benefits of a cloud while maximising the return on existing investments.
5. Applications can dynamically use more resources within the cloud when required ensuring users needs are met immediately.
6. Overloaded applications running on your private cloud can expand to use resources from the public cloud.
7. Data is kept behind the firewall on company infrastructure, requiring fewer changes to existing governance, security and audit procedures.
8. Uses Ubuntu's trusted, stable and lean operating system within the cloud environment.

Management tools:

Landscape is an easy-to-use systems management and monitoring service which enables you to deploy, manage and monitor Ubuntu instances in the cloud. Landscape gives you centralised control over your entire computing environment. It is the only management solution that manages both your cloud and physical systems on one console.

Landscape is designed to simplify the management of both private clouds and public clouds.

With Landscape, we can:

- Manage multiple server machines through a single web-based interface
- Create, start, stop and decommission cloud instances from a simple GUI console
- Store your Amazon EC2 credentials, helping you to access instances quickly
- Manage package and status information on your cloud and physical machines in a single console

Landscape makes it easy to manage:

- Updates and provisioning of packages
- System monitoring
- User control
- Process management
- Inventory control
- Support enhancement tools

The RightScale Cloud Management Platform lets any organization tap the enormous power of cloud computing for scalable, cost-effective IT infrastructure on demand, with complete control and portability. By choosing RightScale, Ubuntu Enterprise Cloud users will be able to maximize the cost efficiency and flexibility of their cloud applications by either testing in their own IT environment and deploying on the public cloud, developing on a public cloud and deploying in their own data center, or deploying some applications in a private cloud and some in a public cloud. Through the RightScale Platform, users will be able to view and manage all their clouds – public, private, and hybrid – with one common interface.

RightScale is the leading provider of cloud management solutions that enable you to design, deploy, manage, and automate business-critical applications on the cloud. To date, hundreds of thousands of deployments have been launched on the RightScale Cloud Management Platform – running everything from scalable websites to complex grid applications. Cloud computing represents a tidal shift in the way IT infrastructure operates, enabling greater agility and lower costs across company sizes. RightScale delivers the power of the cloud to every business.

The RightScale Difference

What sets RightScale apart is our unique approach of managing complete deployments – comprising multiple servers and the connections between them – across one or more clouds. Using the RightScale Platform, you'll be able to operate at a level above the underlying cloud infrastructure with solutions that are architected using best practices for cloud environments. Your deployments will run smoothly, managed by an automation engine that adapts resource allocation as required by system demand, system failures or other events – all based on active monitoring to ensure real-time response to triggers you define.

At every step of the way, you remain in complete control with total visibility into all layers of the cloud infrastructure. This transparency gives you the power to avoid lock-in. Test on one cloud, roll out on another. Architect deployments across clouds to implement disaster recovery plans, provide low-latency access to data, meet security or SLA requirements, or take advantage of better pricing. Maintain your freedom to choose the best cloud environment for your requirements, and the freedom – if needed – to change.

RightScale gets you on the cloud fast with reliable solutions that take less time to manage.

Why RightScale?

Thousands of companies use RightScale every day to run critical business applications in the cloud, leveraging these key benefits:

Cloud-Ready ServerTemplates – Even if you have minimal knowledge of cloud architectures, you can take advantage of pre-packaged solutions for common application scenarios to get up and running on the cloud fast. Our wide variety of ServerTemplates incorporate best practice design architectures – so you can focus on your own application and trust in the reliability, resiliency, and performance of other more standard components.

Managing Deployments – Rather than manage individual servers, the RightScale Platform enables you to manage entire deployments – groups of servers architected to work together – saving valuable systems administration time and reducing costly errors.

Full Automation Across the Deployment Lifecycle – From autoscaling to remediation to automatic configuration, you can achieve a new level of ease in system administration of production deployments – while also providing tools for managing multi-server deployments over their entire lifecycle.

Total Control and Transparency – With RightScale’s open architecture, you retain visibility into all levels of your cloud application. Manage, monitor, test, troubleshoot, and re-launch applications with total control. You can also store valuable operations runbook knowledge and break free from the limits of virtual machine images.

Eliminate Lock-in with Cloud Portability – Gain the freedom and flexibility to choose among a variety of development languages, software stacks, data stores and cloud providers. Choose the right cloud infrastructure for your unique requirements, whether that involves security and regulatory compliance, geographical location, special features or pricing. Manage and migrate deployments across clouds – public or private – so that you never get locked in to a single provider.

Whether you are just getting started and need a simple on-ramp to the cloud or require support for complex deployments spanning multiple clouds, the RightScale Cloud Management Platform provides complete automation, thereby reducing the administration and complexity of managing cloud deployments, yet giving you the flexibility, control, and portability you need.

Cohesive Flexible Technologies provides onboarding solutions for virtual and cloud computing infrastructures like those powered by the Ubuntu Enterprise Cloud (UEC). CohesiveFT's Elastic Server® platform is a web-based factory for dynamic custom multi-sourced server assembly and fast deployment to virtual formats, public clouds, and/or private UECs. The Elastic Server also includes a Cloud Manager that gives users the ability to manage their cloud servers. Additionally the VPN-Cubed® packaged service gives customers control of networking in the clouds, across clouds, and between their private data center and the clouds.

5. Deploying web services in BlobSeer

5.1 The architecture:

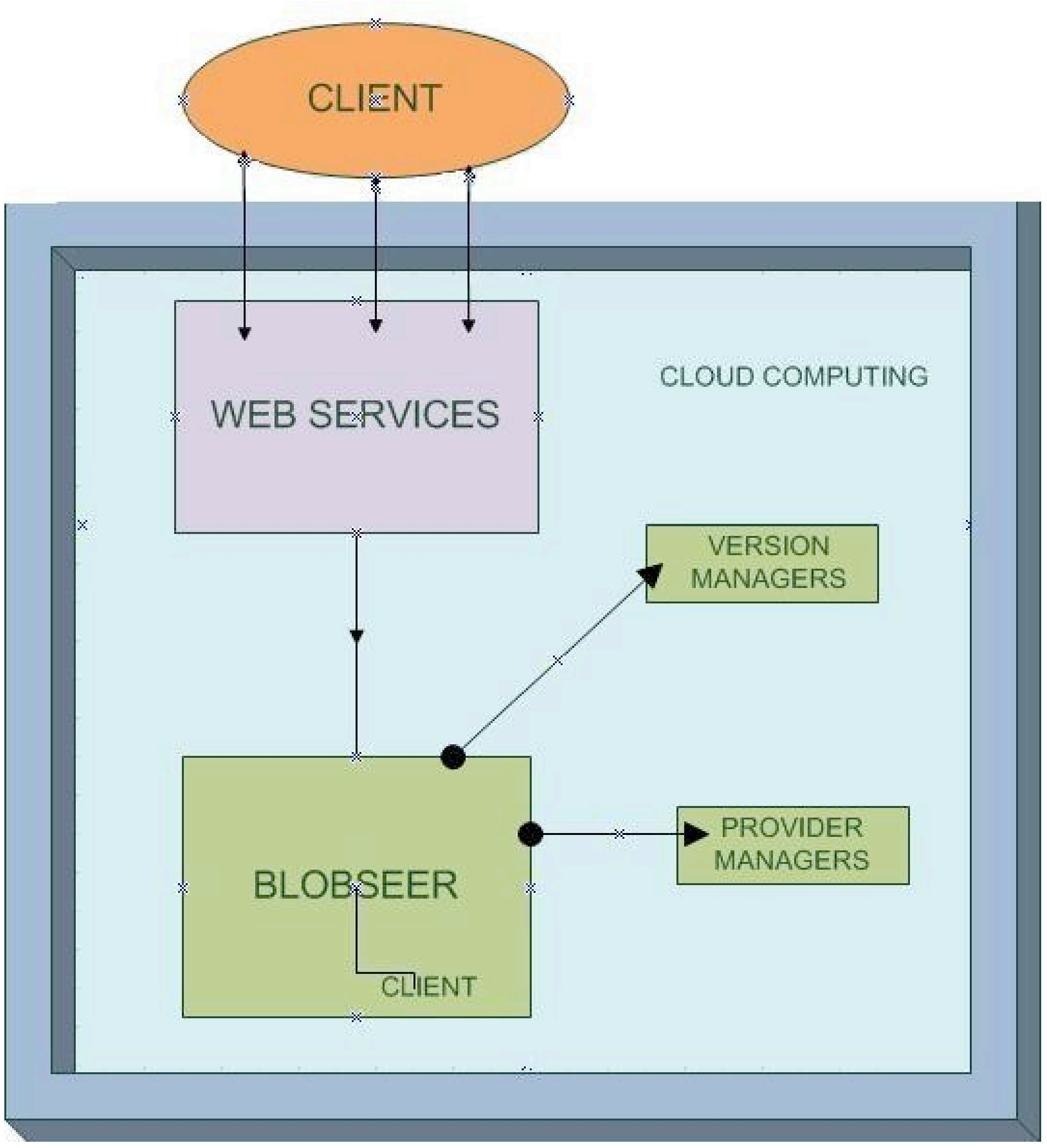


Fig6: Project architecture

Clients can have direct access only to web services. They can create, delete, invoke, deploy web services, but they cannot access the BlobSeer client directly. The only way clients can

write, append or read data is by invoking web services that are specialized with these operations or by creating new web services.
All the BlobSeer managers are in the Cloud. BlobSeer Client can communicate with the managers (Provider Managers, Version Managers, Matadata Providers) using TCP Sockets. The cfg file will specify each manager’s ip address.

```
-----------------------------------------------------------------------------------------------------------
# Version manager configuration
vmanager: {
   # The host name of the version manager
   host = ${vmanager};
   # The name of the service (tcp port number) to listen to
   service = "2225";
};

# Provider manager configuration
pmanager: {
   host = ${pmanager};
   service = "1115";
};
…..........................................................................................................................................
```

The line “host = ${pmanager};” can be changed with “host = “localhost”;” or “host = “127.0.0.1”;” if the provider/manager is in the same place with the client (localhost). If one of the managers is somewhere in cloud, the “$(vmanager)” will be seen as the ip address where that manager is deployed to. The “service=1115” line specify the tcp port number to losten to. The BlobSeer Client will be reached only by the services created. The client cannot directly access the BlobSeer client or its functionalities. The client will see only an interface. First he will have to chose if he wants to register of authenticate, after that he will need to fulfill some fields in order to register into the system or authenticate to it. Only after an successful authentication, the client will be able to see the services, to invoke them accordingly to his rights and deploy his own service.

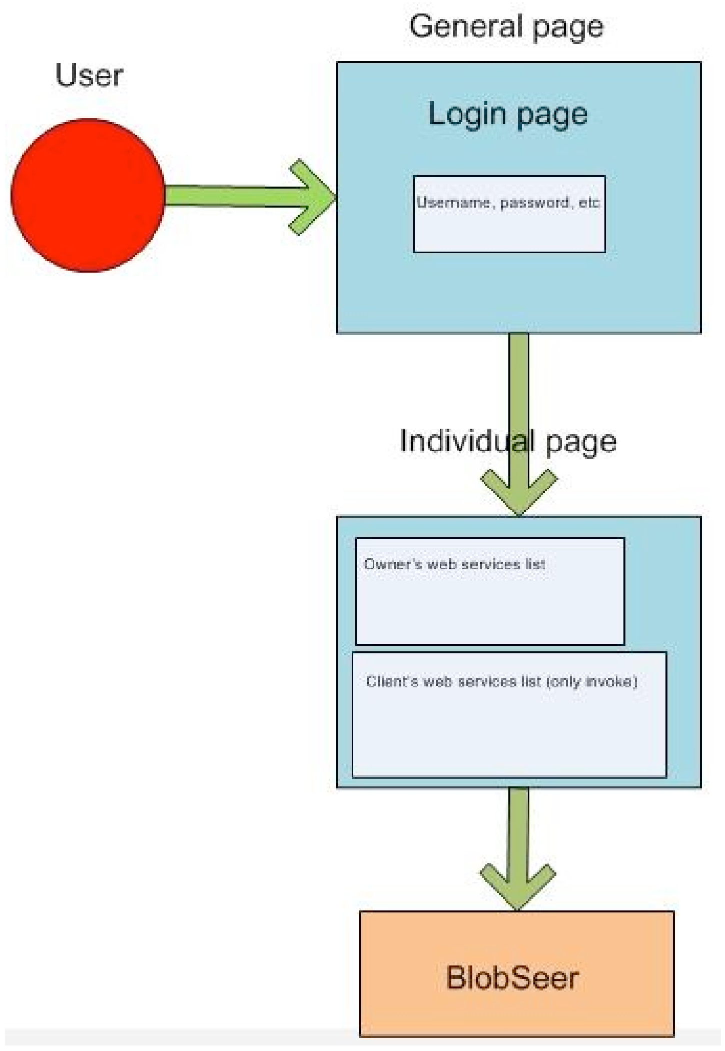


Fig7:Project data flow

As one can see, a User must log in in order to be able to access his own web services. This is the primary security issue. The logged in users can then see their web services as owners and the others web services as clients.

The web pages will be published by an application that will communicate with a Service Manager (a web service that will make the connection between clients and web services). The Service Manager processes the information given by the application of the client and will return the Session ID and the web services the client has access to.

5.2 Types of users

The first security issue is to allow only registered users to access the Web Services. This will be done by an authentication web page. Users are asked to register before they can log in into the Cloud. Therefore there are two types of users: registered users and unauthorized users.

The second issue are the access rights. To prevent unwanted users from accessing blobs data, we will create an ACL.

There will be 2 main types of users:

1. Administrator
2. User

A User can be :

1. Owner of a blob
2. Client

The administrator is the most important user. It can be an individual process or a real person. The administrator can see, access, modify or even remove other users' rights.

The owner has more rights than an ordinary client. He can access, modify or delete his own web services. For other web services he is just a client.

The client has the fewest rights. He can invoke only the web services to which he is granted access.

5.2.1 The Administrator

The main rights of an administrator are:

a. To block the access to a blob

b. To limit the number of services or blobs created

When a client is in the “black list”, his trust level is very low or he has attacked the system, the administrator can block his access to one or more of the blobs.

Due to the fact that a client can create as many blobs and services as he desires, in order to stop a DOS attack from happening, the administrator is able to limit the number of the web services or blobs.

5.2.2 The Owner

The owner of a web service has the following rights:

a. To create blobs

b. To read/write data to the blob they've created

c. To deploy services over the blob they've created

d. To allow access to clients to invoke their web services

Only an owner can create blobs. Once a client creates a web service, he becomes an owner. He has full access to the blob he has created (he can read/write/append data to that blob).

A client, in order to use (invoke) the web service created by an owner, has to be granted access to that web service. Access can only be granted by the owner.

5.2.3 The Client

A Client has only two rights:
a. To invoke only the web services they have access to.
b. To request access to other web services from owners.

If a client wants to have access to a web service (to invoke it) he has to ask the owner for it first, then he can invoke the service. A client cannot modify anything in that web service and he doesn't have any read/write rights on any blob.

5.3 Use cases

There are three main types of actors: Clients, Owners and the Administrator. The unregistered clients have to register first, in order to be able to log into the system. The Clients and the

Owners have to be authenticated in order to access the information. Only after a successful authentication can they see a list of the web services they have access to and other deployed Web Services. A client needs to request the invoke right from the owner of the service he wants to access. An owner of a service also needs to request the invoke right if he desires to access a service other than his own.

Both the client and the owner can deploy their own Web Services if their requests are granted by the administrator. The owner of a service is able to grant access to other clients for his service only.

The administrator can grant the Deploy right and is the only one who can revoke any kind of right.

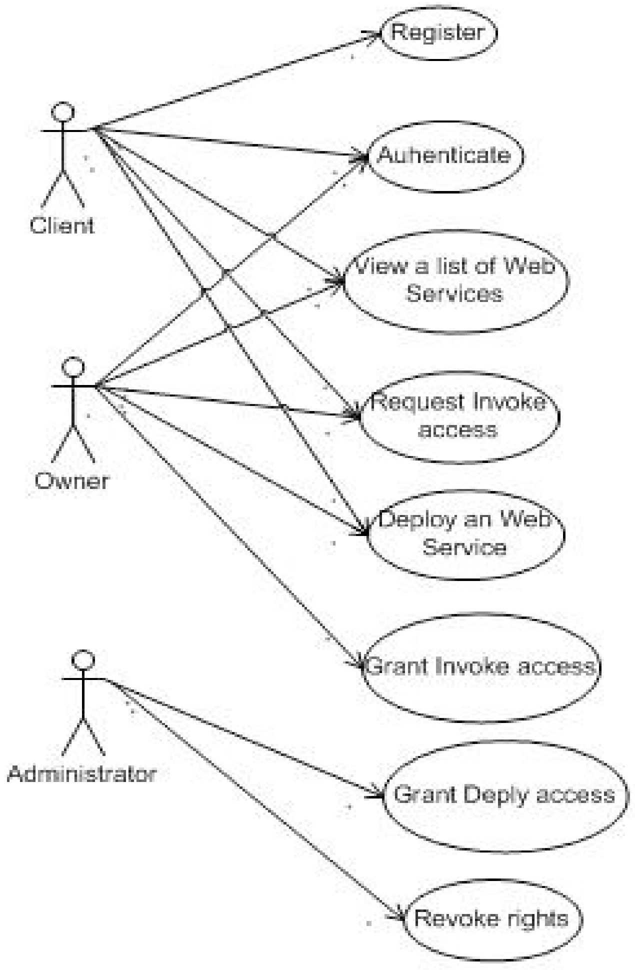


Fig. 8 Use Case

5.4 Secure access to web services

A user first accesses the Client's application in order to log in. The application then sends the information (username, password, the amount of time he wants to be logged in) to the Service Monitor. The Service Monitor receives the data from the application and searches into the database to check if the log in data is correct. If it is not correct, the SM sends back an error message If it is correct, the SM then returns a validation message and the SID (session id) and writes into the Conf File the SID, the expiration date and a list of services he has access to (client's ACL). The user can then get a list of services on which the client has rights (his ACL) and a list of services that are deployed into the Cloud. The list will be displayed in the client browser by the application. By using the application, the client is then able to access each service according to his rights or deploy his own Web Service.

In order to access a service a Client must have the Invoke right. To get this right he needs to send a Request message that will contain his name, his SID and the name of the service. If the SID is valid and corresponds to the name, the request will be listed for the Owner to see. The service owner can view all the request messages and choose which ones he wants to grant access to. After the owner grants the access, the ACL of the Users requesting the access will contain that service, with the right level of 2;.

In order to deploy a service a User must have the Owner right. For this right he needs to send a request message to the Administrator. This request will contain his name, his SID, the proposed name of the service, the methods and a short description. If the SID is valid and it corresponds to the name and if the proposed name of the service is unique the request will then be allowed. After the Administrator grants the access, the ACL of the Users requesting the access will contain that service, with the right level of 1.

Basically, a client can access only the services from his ACL, only the services he has rights to. The invoke rights permits to access the methods of the service. A client who has the invoke right cannot modify the service classes, only the owner has this right.

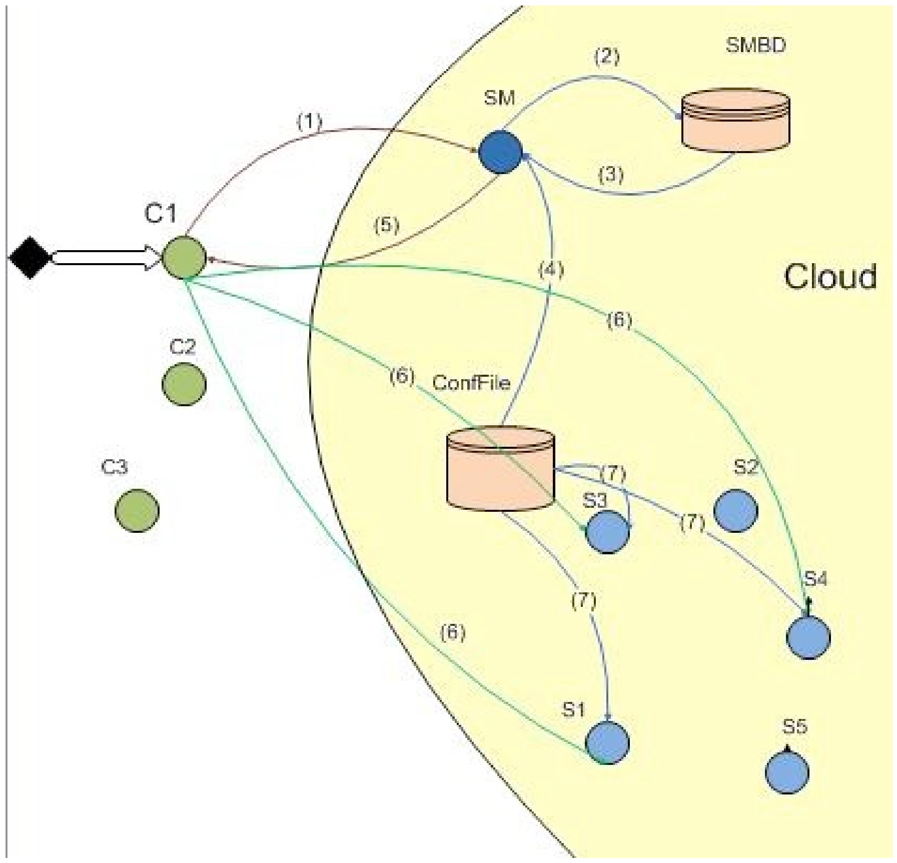


Fig. 9 The actions

The following steps are:

(1). If the client is new to the system: The client sends a request to the SM in order to Register. The request contains the name of the client, the username and the password the client will use to authenticate into the system. If the client has already registered: The client sends a request to the SM, in order to Authenticate. This request contains: the username, the password and the amount of time the client desires to be logged into the system.

(2). If it is a new client, the SM writes into the SMBD file the name, username and password of the client. If it is a registered client, the SM searches into the the SMBD file and checks the username and the password.

(3). The SM reads the SMBD file and extracts the username of the client, then checks if the password is correct.

(4). If the data are correct, the SM writes into the Conf file the name of the client, a Session ID (SID) that is unique for each client, the date when the entry was written, the expiration date (the day the client authenticates + the time the client desires to stay logged in) and the ACL of the client.

(5). The SM sends back either an error code and an error message in case the client cannot be authenticated, or the SID, a list of the services the client has certain rights to and a list with the services that are deployed into the system, in case the client is successfully authenticated.

(6). The client is then able to send requests directly to the Services he wants to access. If the client doesn't have any rights to that service he sends an invoke request with his name, his SID and the name of the service. If the client already has the invoke right, he is able to access the desired functions.

(7). The service then processes all the requests and returns only the valid requests to the owner of the service (a request is valid if the SID has not expired, if the SID belongs to that specific client and if the request is for that service). All this information is taken from the Conf file. The owner of the service can then choose whom he wants to grant access to. This can be done manually or by writing a function to automatically accomplish that.

If a client wants to deploy his own Web Service, the steps (6) and (7) are applied for the AdminService.

5.4.1 ACLs

Each client is able to access a list containing the services on which he has certain rights. There are 4 levels of rights: level 0 – the Administrator level; level 1 – the Owner level; level 2 – the User level and level 3 – the Client level.

A service from such a list contains: the name of the service, its IP address, a short description and the level of rights the user is granted access to.

The list for a client who has just registered will contain no entry. A new client can only view a list of all the services with short descriptions and methods belonging to each service, but he cannot invoke or edit any of them.

In order to obtain the rights, he has to ask for permission to the owners and the administrator. To obtain the right to invoke a certain service, the client needs to send a request to the owner of that service. In the request he will specify the name of the service, its SID and the name of the client. In order to deploy his own web service, the client needs to send a request to the administrator. The request includes the proposed name of the service, the methods contained in the web service and a short description of it. The administrator will check if the name of the service is unique and if all the other requirements are fulfilled. He will then decide whether to approve the service or not. In case the request is approved, the client will be asked to deploy the service. The client will then become the Owner of that particular service.

After getting the invoke right, the client will have into his list of services (his ACL) the service he just got the right to invoke. This right allows the client to access any function of the service. A good idea of developing this project is to permit the client to give different level fights for each function in his service, but this will be discuss more elaborate in the future work chapter.

The owner of a service is able to approve invoke requests from other users. The owner possesses a list containing the names of all the users that are granted the right to invoke his web service. The list of a user that has just deployed his service will be empty.

The administrator possesses a list containing the names of all the clients in the system. For each client, the administrator is able to view the list of the services they own or can invoke, if any.

We could have chosen for a level representation of the services. This kind of representation is simpler than the one we've chosen but it doesn't offer us the possibility for clients to have the services they want with the rights they want. Instead we've chose n a matrix representation: for every client and every service is a right level form 0 to 3. In this representation the client has the possibility of choosing any services he wants. There for Clients, owners and the administrator has there own ACL with the services deployed into the system.

6. Implementation details.

We deploy the virtual machines on Cloud . The virtual machines can have any name. The first virtual machine created is “BlobSeer”. This virtual machine has the following contents (the other virtual machines are similar to this one):

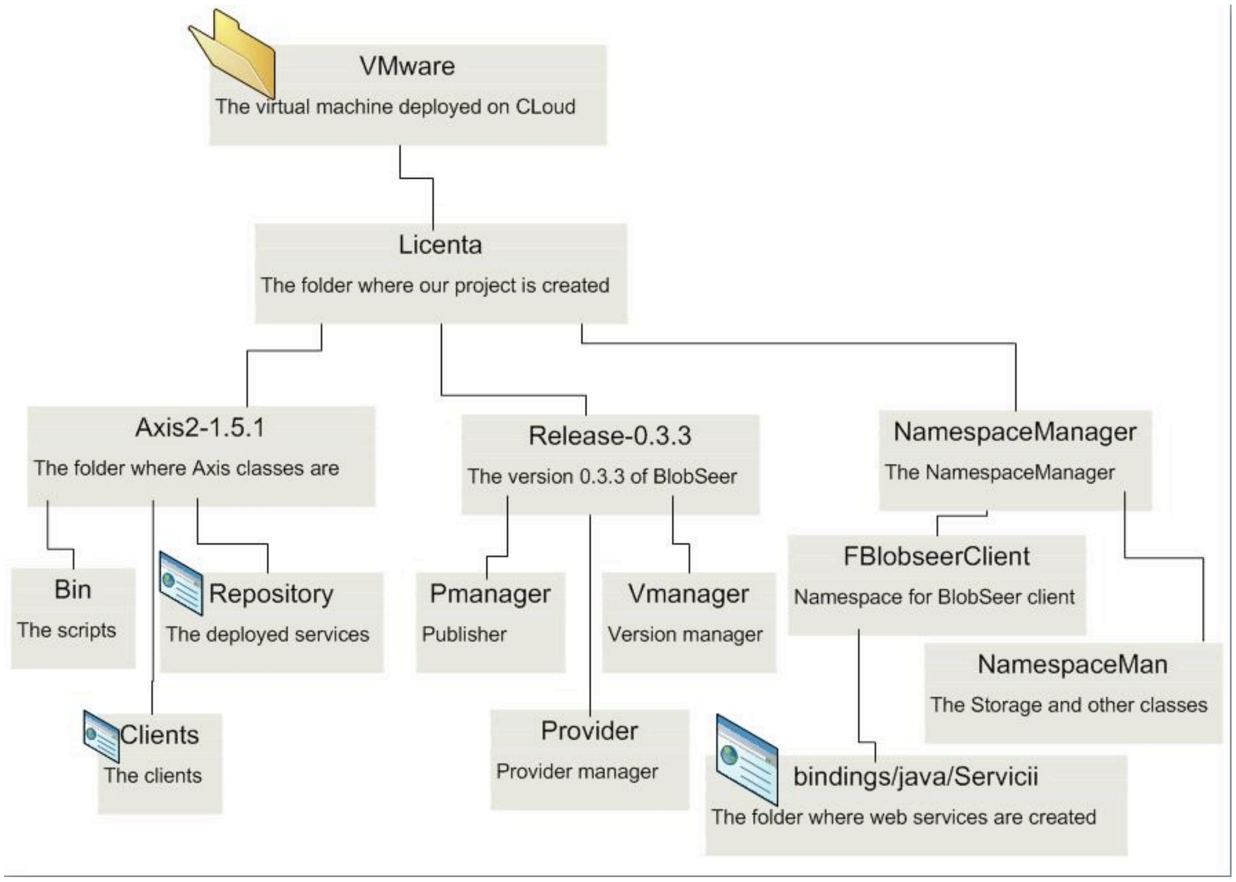


Fig. 10 System Architecture

The main folder is “Licenta”. This is where all the interesting data is located. In the folder “Release-0.3.3”, there are the BlobSeer classes. This folder contains all the managers (pmanager, vmanager, provider), the client and all the data the client needs in order to work properly. It should be noted that all these providers can be located anywhere into the Cloud.

The “Namespace Manager” folder contains the java bindings, as well as all the data needed to create the namespace. Basically, it is located between BlobSeer and the Web Services. We need the java bindings due to the fact that BlobSeer is created in C++ and we work with the

Web Services in java. Web Services can be written by using axis in C++, but it is easier for the clients who want to deploy services into this system to write them in java. We want to make this as simple as possible for everyone to use, therefore we have decided that a namespace is needed, so we have a Namespace Manager who does all the mapping between a folder and a blob. Basically, a blob is treated as a folder. The namespace has functions such as creating a directory or a file, deleting a directory or a file, reading, writing and appending to a file. It doesn't have the delete from file function, as BlobSeer doesn't have it. The main idea of the BlobSeer is that you don't modify or delete anything, you just keep adding new versions. The namespace doesn't have the possibility to view older versions of the same file. Having this possibility, however, could prove to be useful for the client... this issue will be addressed in the course of future work.

In the NamespaceManager folder, there is the "FblobSeerClient" folder, where are the bindings for the BlobSeer client and the "NamespaceMan" folder where the namespace classes are located. In the "FblobSeerClient/bindings/java/Servicii" folder are the java codes of the Web Services. After a web service is compiled, the resulting .aar (Axis2ARchive) file is moved in the "Repository" folder to be deployed in axis.

So we have reached the axis2-1.5.1 folder. The bin folder is located in here, together with all the scripts that are necessary to run axis. We are using the axis2server script because it has a servlet container in which we can deploy the services. The Repository folder contains all the Web Services as axis2ARchives (.aar). This is important, because all the services must be archived and dropped into this folder, in order to be deployed . The last one is the Clients folder, which contains the applications or scripts of the clients. These applications can be anywhere in the Cloud, but their source codes must be located in the axis folder in order to compile correctly.

7. Performance and experimental results

We've tested the main functions for one client, five clients and ten clients. The time is calculated in milliseconds. As it can be seen above, the most demanding function is the one using to deploy a service. On the other side is the RequestList function that took the less amount of time to run.

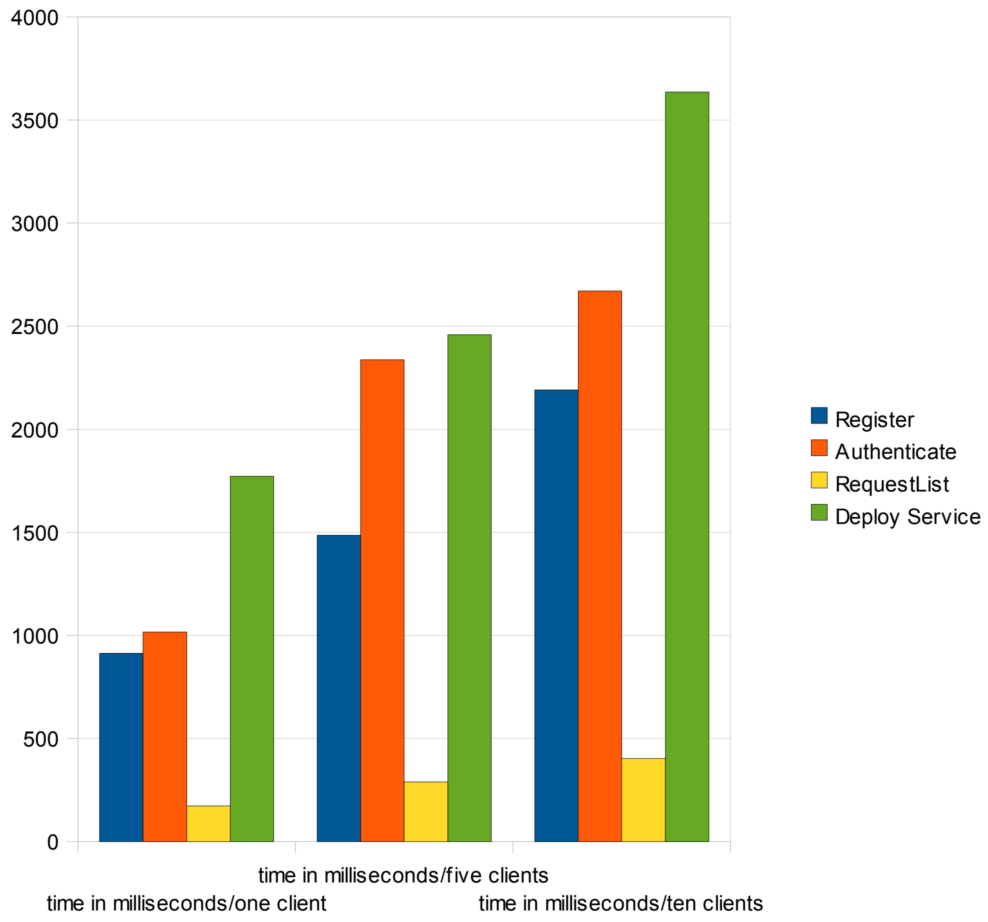


Fig.11 Test branch1

Next we've calculated the amount of time (in milliseconds) necessary for clients to do more then one function at a time.

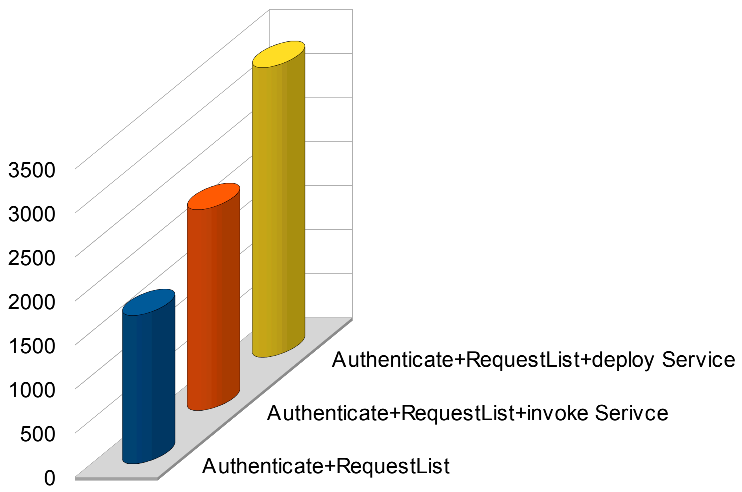


Fig.12 Test branch2

8. Conclusions and future work

We want to integrate this project with Ana-Maria Lepar's project and Cristina Basescu's project. Ana's project is designed to ascertain the trust levels of the clients. The output of her project consists of a list of clients and their respective trust levels. The administrator is then able to revoke certain rights, according to this list. For example, if a client has a very low trust level, he will not be ale to deploy any web service.

Cristina's project is about discovering and stopping clients to do DOS. The output of her project is a “black list” with people who have tried to break the system. The administrator is then able to revoke certain rights for those clients.

As we have previously mentioned, the namespace manager is not able to show older versions of the same file, therefore we intend to create such a function.

The invoke right offer the Client the possibility to access any function of that service. Our idea is to allow the owner of that server to put different level of rights for everyone of his functions. The idea of differencing the functions from a service is that some functions are more important (functions that allow the clients to modify the content of a file) and need to be accessed only by trusted users and others are less important (showing the content of a folder for example) and could be accessed by any kind of client.

In conclusion, the work for this project is only at the beginning. This is a very complex and elaborate platform that can be continuously developed

This project desires to become a viable solution for experienced programmers, as well as for those who do not have experience in working with web services.

9. References

1. Web Services Reliable Messaging Protocol (WS-ReliableMessaging), March 13, 2003, Authors: Ruslan Bilorusets, BEA; Adam Bosworth, BEA; Don Box, Microsoft; Felipe Cabrera, Microsoft; Derek Collison, TIBCO Software; Jon Dart, TIBCO Software; Donald Ferguson, IBM; Christopher Ferris, IBM (Editor); Tom Freund, IBM; Mary Ann Hondo, IBM; John Ibbotson, IBM; Chris Kaler, Microsoft; David Langworthy,

Microsoft (Editor); Amelia Lewis, TIBCO Software; Rodney Limprecht, Microsoft; Steve Lucco, Microsoft; Don Mullen, TIBCO Software; Anthony Nadalin, IBM; Mark Nottingham, BEA; David Orchard, BEA; John Shewchuk, Microsoft; Tony Storey, IBM.

2. Developing Web Services with Apache Axis2, 2005-2008, Author: Kent Ka Iok Tong
3. Axis2, Middleware for Next Generation Web Services, 2006, Authors: Srinath Perera, Chathura Herath, Jaliya Ekanayake, Eran Chinthaka, Ajith Ranabahu, Deepal Jayasinghe, Sanjiva Weerawarana,Glen Daniels.
4. Emergence of The Academic Computing Clouds , August 5 - 11, 2008, Authors: Kemal A. Delic; Martin Anthony Walker, Hewlett-Packard Co.
5. High Throughput Data-Compression for Cloud Storage, Jun 9 2010, Author: Bogdan Nicolae
6. BlobSeer: Efficient Data Management for Data-Intensive Applications Distributed at Large-Scale, Oct 30, 2010, Bogdan Nicolae
7. BlobSeer: how to enable efficient versioning for large object storage under heavy access concurrency, 2009, Bogdan Nicolae, Gabriel Antoniu, Luc Bougé
8. Distributed Management of Massive Data: An Efficient Fine-Grain Data Access Scheme, 2008, Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé.
9. Cloud Architectures, Jinesh Varia
10. Securing the Cloud:A Review of Cloud Computing, Security Implications and Best Practices, 2009, Hillview Ave Palo Alto
11. *Cloud Computing,Oct 8, 2007, Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni,* Harold Hall
12. Web Service Architectures, 2008,Judith M. Myerson

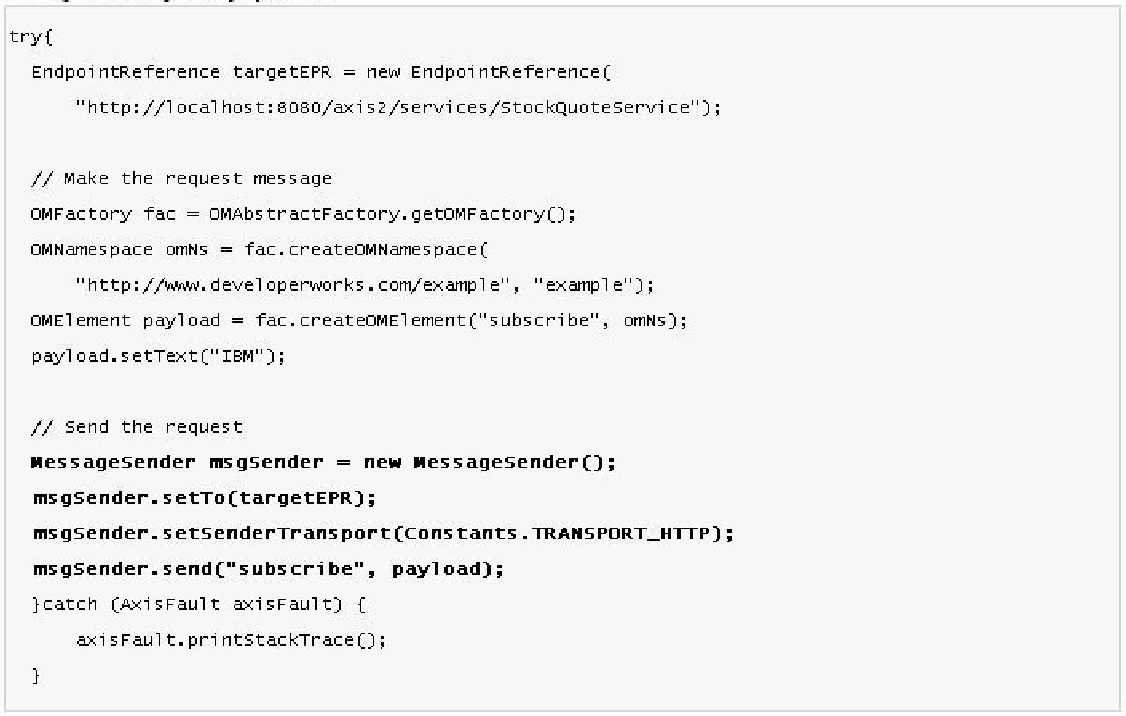
Annexes:

1. Given below are the details of invoking the In-Only and In-Out operations using the Axis2 client API.

MessageSender.send() sends the request message and returns it immediately. The transport to be used is specified by MessageSender.setSenderTransport(). This example sends the message over HTTP.

Invoking an In-Only operation:

Listing 3. Invoking In-Only operation



```
try{
  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");

  // Make the request message
  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
  OMElement payload = fac.createOMElement("subscribe", omNs);
  payload.setText("IBM");

  // Send the request
  MessageSender msgSender = new MessageSender();
  msgSender.setTo(targetEPR);
  msgSender.setSenderTransport(Constants.TRANSPORT_HTTP);
  msgSender.send("subscribe", payload);
  }catch (AxisFault axisFault) {
      axisFault.printStackTrace();
  }
```

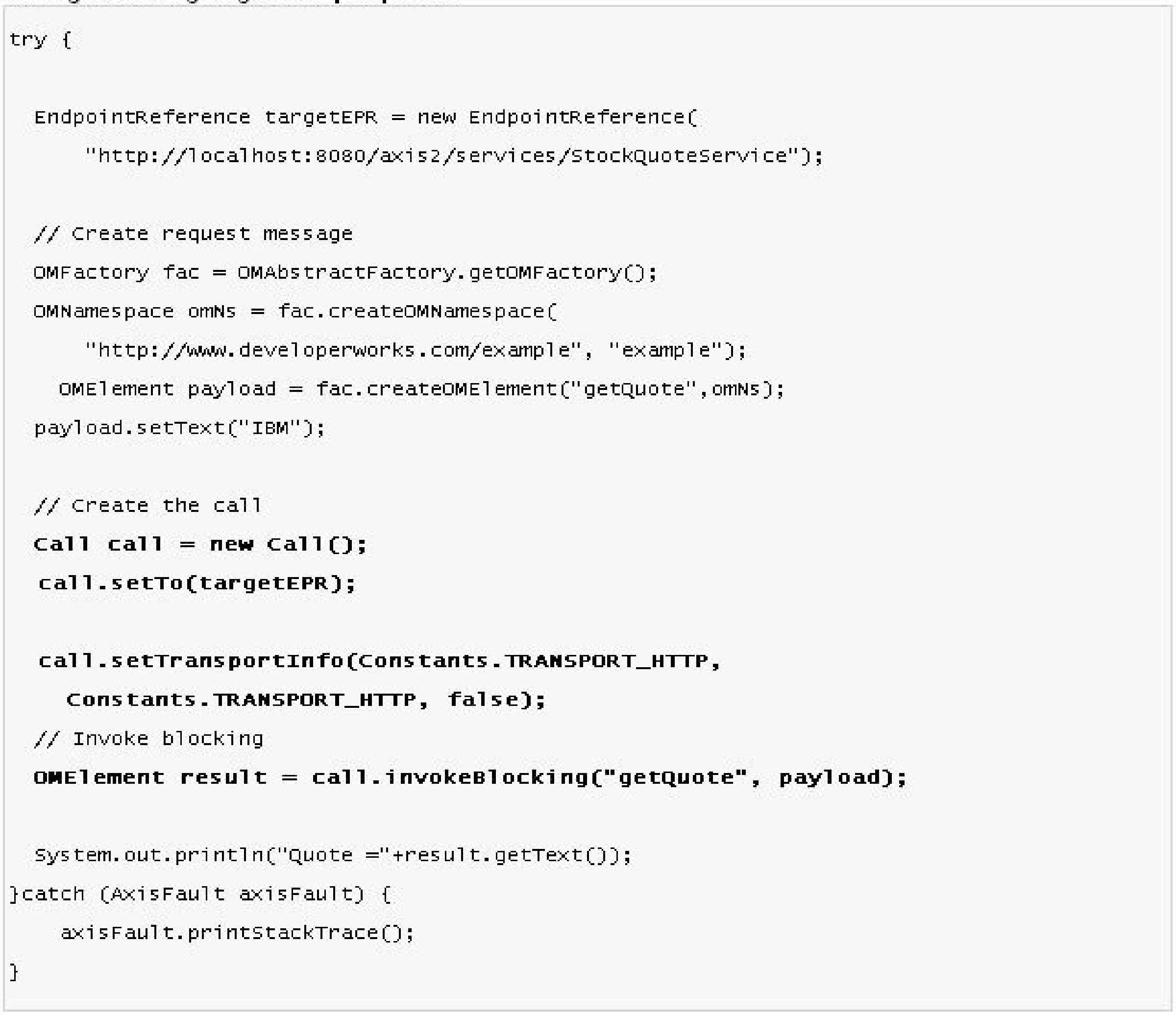
The In-Only operation is a very simple one. The service invocation flows to the sender, it doesn't have to wait until the operation completes and the response is received.

Another in-only function is fireandforget(): msgSender.fireAndForget(“subscribe”,payload);

Invoking an In-Out operation:

1. **Blocking Single Transport pattern:** This is the simplest way of invoking an In-Out Web service operation. The service invocation is blocked until the operation completes and the response or fault is received. It uses a single transport connection for sending and receiving the response.

Listing 4. Blocking Single Transport pattern

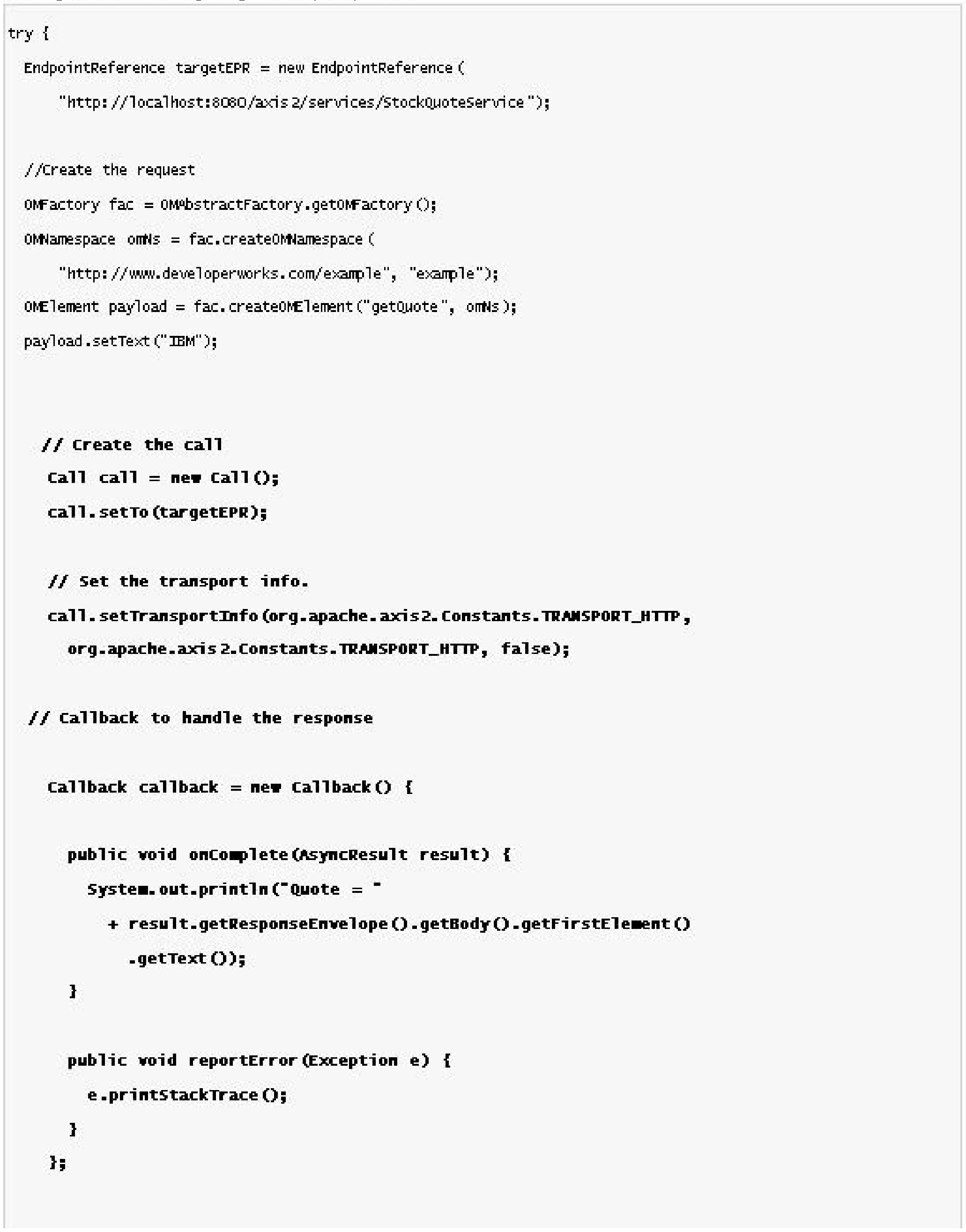


```
try {

  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");

  // Create request message
  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
    OMElement payload = fac.createOMElement("getQuote",omNs);
  payload.setText("IBM");

  // Create the call
  Call call = new Call();
  call.setTo(targetEPR);

  call.setTransportInfo(Constants.TRANSPORT_HTTP,
    Constants.TRANSPORT_HTTP, false);
  // Invoke blocking
  OMElement result = call.invokeBlocking("getQuote", payload);

  System.out.println("Quote ="+result.getText());
}catch (AxisFault axisFault) {
    axisFault.printStackTrace();
}
```

The first part of the code creates the request message using AXIOM. Call.setTransportInfo() sets the transports to be used for sending the request and getting the response. The Boolean argument of the Call.setTransportInfo() operation tells if a separate transport connection is to be used for sending the request and receiving the response. In this case, ask for a single HTTP connection for sending and receiving.

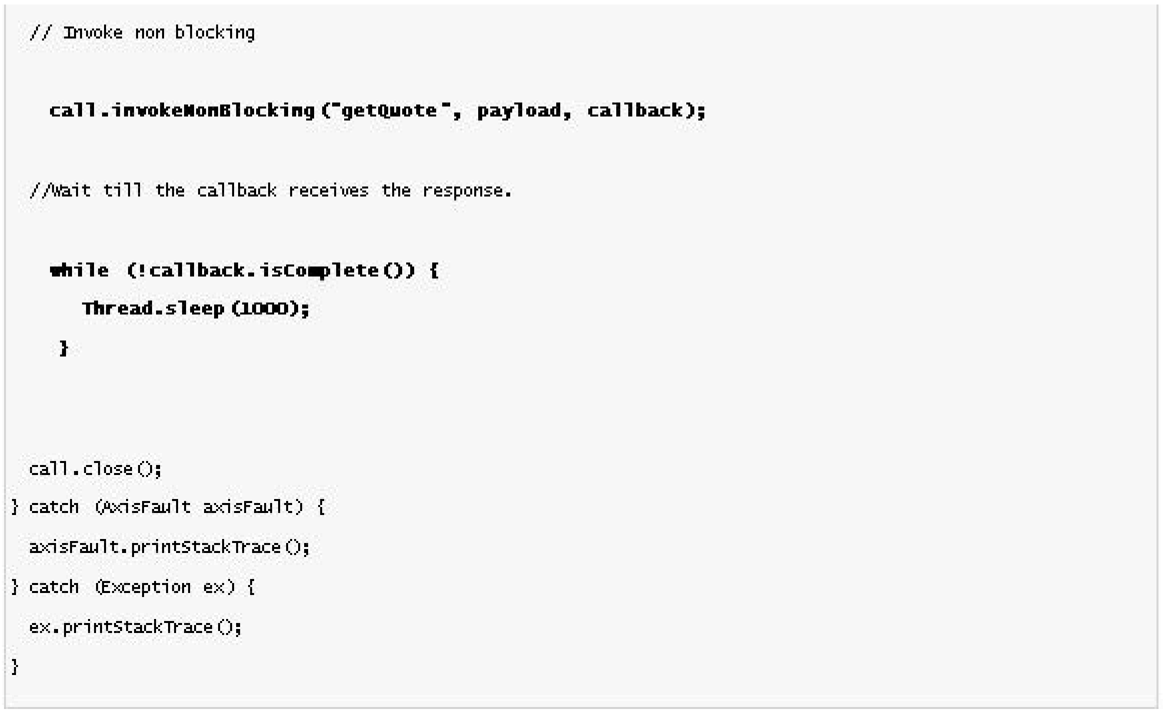
2. **Non-Blocking Single Transport pattern:** In this invocation pattern, you achieve a non-blocking call using only a single transport connection underneath. This kind of behavior is needed when there are many Web service invocations to be done in a single client application, and you don't want the client to be blocked for every invocation. Here the invocation returns immediately and the client gets a callback when the response is available.

Listing 5. Non-Blocking Single Transport pattern



```
try {
  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");

  //Create the request
  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
  OMElement payload = fac.createOMElement("getQuote", omNs);
  payload.setText("IBM");


    // Create the call
    Call call = new Call();
    call.setTo(targetEPR);

    // Set the transport info.
    call.setTransportInfo(org.apache.axis2.Constants.TRANSPORT_HTTP,
      org.apache.axis2.Constants.TRANSPORT_HTTP, false);

  // Callback to handle the response

    Callback callback = new Callback() {

      public void onComplete(AsyncResult result) {
        System.out.println("Quote = "
          + result.getResponseEnvelope().getBody().getFirstElement()
            .getText());
      }

      public void reportError(Exception e) {
        e.printStackTrace();
      }
    };
```

The Call.invokeNonBlocking() method returns immediately without blocking. Call.invokeNonBlocking() takes an object of org.apache.axis2.clientapi.CallBack, which will be triggered when the response comes from the service. The CallBack has two abstract methods, onComplete(AsynchResult) and reportError(Exception), that need to be implemented by the concrete CallBack class. The Axis2 engine calls the onComplete() method when the service invocation completes normally. On a fault message from the server, the reportError() method of the Callback is invoked. Callback.isComplete() tells you whether the operation invocation is completed or not.



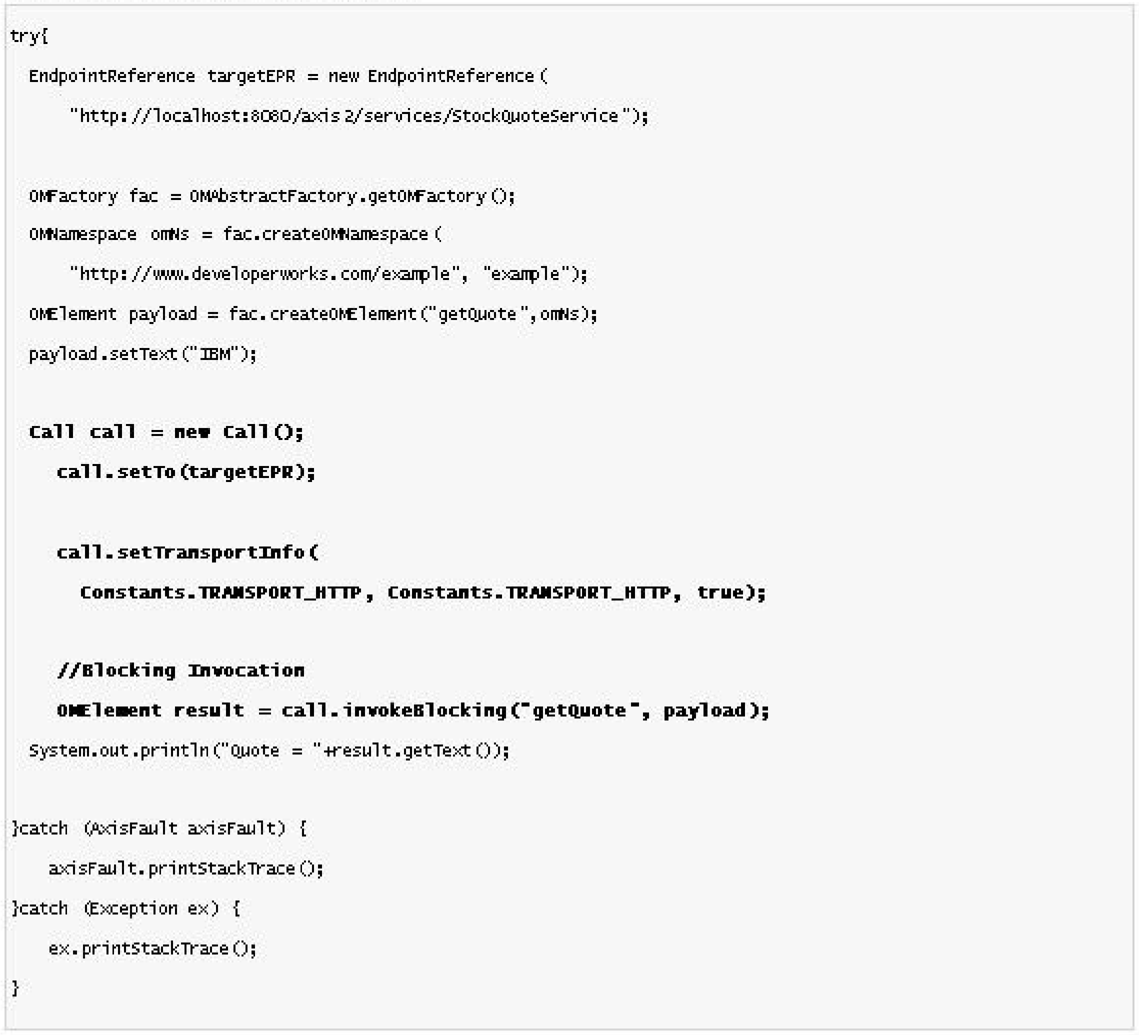
```
  // Invoke non blocking

    call.invokeNonBlocking("getQuote", payload, callback);

  //Wait till the callback receives the response.

    while (!callback.isComplete()) {
        Thread.sleep(1000);
     }


  call.close();
} catch (AxisFault axisFault) {
  axisFault.printStackTrace();
} catch (Exception ex) {
  ex.printStackTrace();
}
```

Since both of the above approaches use a single transport connection to send and receive messages, they are not suitable for long-running transactions. This is because the transport connection might timeout before the response is available. To address this problem, two separate transport connections can be used for request and response. But since a different transport connection is used to retrieve the response, the request and response need to be correlated. Axis2 supports WS-Addressing, which provides a solution to this problem by using

<wsa:MessageID> and <wsa:RelatesTo> headers. So addressing the module needs to be enabled when using dual transport, as shown in the two patterns below.

3. **Blocking Dual Transport pattern:** This pattern is useful when the service operation is In-Out in nature, but the transport used is One-Way (for example, SMTP) or when the service execution takes a long time and HTTP connection times out.

Listing 6. Blocking Dual Transport pattern



```
try{
  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");

  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
  OMElement payload = fac.createOMElement("getQuote",omNs);
  payload.setText("IBM");

  Call call = new Call();
    call.setTo(targetEPR);

    call.setTransportInfo(
      Constants.TRANSPORT_HTTP, Constants.TRANSPORT_HTTP, true);

    //Blocking Invocation
    OMElement result = call.invokeBlocking("getQuote", payload);
  System.out.println("Quote = "+result.getText());

}catch (AxisFault axisFault) {
    axisFault.printStackTrace();
}catch (Exception ex) {
    ex.printStackTrace();
}
```

4. **Non-Blocking Dual Transport pattern:** This pattern provides maximum flexibility, both in terms of not blocking at the API level and at the transport level.

Listing 7. Non-Blocking Dual Transport pattern

```
try {

  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");
  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://www.developerworks.com/example", "example");
  OMElement payload = fac.createOMElement("getQuote",omNs);
  payload.setText("IBM");


    Call call = new Call();
     call.setTo(targetEPR);

     call.setTransportInfo(
       Constants.TRANSPORT_HTTP, Constants.TRANSPORT_HTTP, true);
  // Callback to handle the response

    Callback callback = new Callback() {
     public void onComplete(AsyncResult result) {
       System.out.println("Quote = "
           + result.getResponseEnvelope().getBody().getFirstElement()
         .getText());
       }

     public void reportError(Exception e) {
         e.printStackTrace();
       }
     };


  // Non-Blocking Invocation

    call.invokeNonBlocking("getQuote", payload, callback);
  // Wait till the callback receives the response.

    while (!callback.isComplete()) {
       Thread.sleep(1000);
     }
     call.close();
}catch (AxisFault axisFault) {
      axisFault.printStackTrace();
}catch (Exception ex) {
      ex.printStackTrace();
}
```

The steps to creating a dynamic client are:

Determine if you want your dynamic client to send data in PAYLOAD or MESSAGE mode.

Create a service instance and add at least one port to it. The port carries the protocol binding and service endpoint address information.

Create a Dispatch<T> object using either the Service.Mode.PAYLOAD method or the Service.Mode.MESSAGE method.

Configure the request context properties on the javax.xml.ws.BindingProvider interface. Use the request context to specify additional properties such as enabling HTTP authentication or specifying the endpoint address.

Compose the client request message for the dynamic client.

Invoke the service endpoint with the Dispatch client either synchronously or asynchronously.

Process the response message from the service.

The following example illustrates the steps to create a Dispatch client and invoke a sample EchoService service endpoint.

```
String endpointUrl = ...;

QName serviceName = new QName("http://org/apache/ws/axis2/sample/echo/",

 "EchoService");

QName portName = new QName("http://org/apache/ws/axis2/sample/echo/",

 "EchoServicePort");

/** Create a service and add at least one port to it. **/

Service service = Service.create(serviceName);

service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

/** Create a Dispatch instance from a service.**/

Dispatch<SOAPMessage> dispatch = service.createDispatch(portName,
  SOAPMessage.class, Service.Mode.MESSAGE);

  /** Create SOAPMessage request. **/

  // compose a request message

  MessageFactory mf =
MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

  // Create a message.  This example works with the SOAPPART.

  SOAPMessage request = mf.createMessage();

  SOAPPart part = request.getSOAPPart();

  // Obtain the SOAPEnvelope and header and body elements.

  SOAPEnvelope env = part.getEnvelope();

  SOAPHeader header = env.getHeader();

  SOAPBody body = env.getBody();

  // Construct the message payload.

  SOAPElement operation = body.addChildElement("invoke", "ns1",

   "http://org/apache/ws/axis2/sample/echo/");

  SOAPElement value = operation.addChildElement("arg0");

  value.addTextNode("ping");

  request.saveChanges();

  /** Invoke the service endpoint. **/

  SOAPMessage response = dispatch.invoke(request);

  /** Process the response. **/
```

2. The steps necessary to invoke a Web service asynchronously are:

Determine if you want to implement the callback method or the polling method for the client to asynchronously invoke the Web service.

(Optional) Configure the client request context. Add the

org.apache.axis2.jaxws.use.async.mep

property to the request context to enable asynchronous messaging for the Web services client. Using this property requires that the service endpoint supports WS-Addressing which is supported by default for the application server. The following example demonstrates how to set this property:

```
Map<String, Object> rc = ((BindingProvider) port).getRequestContext();

rc.put("org.apache.axis2.jaxws.use.async.mep", Boolean.TRUE);
```

To implement the asynchronous callback method, perform the following steps.

Find the asynchronous callback method on the SEI or javax.xml.ws.Dispatch interface. For an SEI, the method name ends in Async and has one more parameter than the synchronous method of type javax.xml.ws.AsyncHandler. The invokeAsync(Object, AsyncHandler) method is the one that is used on the Dispatch interface.

(Optional) Add the service.setExecutor methods to the client application. Adding the executor methods gives the client control of the scheduling methods for processing the response. You can also choose to use the java.current.Executors class factory to obtain packaged executors or implement your own executor class. See the JAX-WS specification for more information on using executor class methods with your client.

Implement the javax.xml.ws.AsynchHandler interface. The javax.xml.ws.AsynchHandler interface only has the handleResponse(javax.xml.ws.Response) method. The method must contain the logic for processing the response or possibly an exception. The method is called after the client run time has received and processed the asynchronous response from the server.

Invoke the asynchronous callback method with the parameter data and the callback handler.

The handleResponse(Response) method is invoked on the callback object when the response is available. The Response.get() method is called within this method to deliver the response.

To implement the polling method,

Find the asynchronous polling method on the SEI or javax.xml.ws.Dispatch interface. For an SEI, the method name ends in Async and has a return type of javax.xml.ws.Response. The invokeAsync(Object) method is used on the Dispatch interface.

Invoke the asynchronous polling method with the parameter data.

The client receives the object type, javax.xml.ws.Response, that is used to monitor the status of the request to the server. The isDone() method indicates whether the invocation has completed. When the isDone() method returns a value of true, call the get() method to retrieve the response object.

Use the cancel() method for the callback or polling method if the client needs to stop waiting for a response from the service. If the cancel() method is invoked by the client, the endpoint continues to process the request. However, the wait and response processing for the client is stopped.

When developing Dynamic Proxy clients, after you generate the portable client artifacts from a WSDL file using the wsimport command, the generated service endpoint interface (SEI) does not have asynchronous methods included in the interface. Use JAX-WS bindings to add the asynchronous callback or polling methods on the interface for the Dynamic Proxy client. To enable asynchronous mappings, you can add the jaxws:enableAsyncMapping binding declaration to the WSDL file. For more information on adding binding customizations to generate an asynchronous interface, see chapter 8 of the JAX-WS specification.

Note: When you run the wsimport tool and enable asynchronous invocation through the use of the JAX-WS enableAsyncMapping binding declaration, ensure that the corresponding response message your WSDL file does not contain parts. When a response message does not contain parts, the request acts as a two-way request, but the actual response that is sent back is empty. The wsimport tool does not correctly handle a void response. To avoid this scenario, you can remove the output message from the operation which makes your operation a one-way operation or you can add a <wsdl:part> to your message. For more information on the usage, syntax and parameters for the wsimport tool, see the wsimport command for JAX-WS applications documentation.

The following example illustrates a Web service interface with methods for asynchronous requests from the client.

```
  @WebService
  public interface CreditRatingService {
        // Synchronous operation.
        Score getCreditScore(Customer    customer);
        // Asynchronous operation with polling.
        Response<Score>
 getCreditScoreAsync(Customer customer);
        // Asynchronous operation with callback.
        Future<?>
 getQuoteAsync(Customer customer,
              AsyncHandler<Score>
 handler);
  }
```

Using the callback method The callback method requires a callback handler that is shown in the following example. When using the callback procedure, after a request is made, the callback handler is responsible for handling the response. The response value is a response or possibly an exception. The Future<?> method represents the result of an asynchronous computation and is checked to see if the computation is complete. When you want the application to find out if the request is completed, invoke the Future.isDone() method. Note that the Future.get() method does not provide a meaningful response and is not similar to the Response.get() method.

```
  CreditRatingService svc = ...;
  Future<?>
 invocation = svc.getCreditScoreAsync(customerTom,
        new AsyncHandler<Score>() {
              public void handleResponse (
                    Response<Score> response)
                  {
                    score = response.get();
                    // process the request...
                  }
      }
  );
```

Using the polling method The following example illustrates an asynchronous polling client:

```
  CreditRatingService svc = ...;
  Response<Score> response = svc.getCreditScoreAsync(customerTom);
  while (!response.isDone()
) {
        // Do something while we wait.
  }
  score = response.get()
```

Enabling HTTP session management support for JAX-WS applications:

You can use HTTP session management to maintain user state information on the server, while passing minimal information back to the user to track the session. You can implement HTTP session management on the application server using either session cookies or URL rewriting.

The interaction between the browser, application server, and application is transparent to the user and the application program. The application and the user are typically not aware of the session identifier provided by the server.

Session cookies

The HTTP maintain session feature uses a single cookie, JSESSIONID, and this cookie contains the session identifier. This cookie is used to associate the request with information stored on the server for that session. On subsequent requests from the JAX-WS application, the session ID is transmitted as part of the request header, which enables the application to associate each request for a given session ID with prior requests from that user. The JAX-WS client applications retrieve the session ID from the HTTP response headers and then use those IDs in subsequent requests by setting the session ID in the HTTP request headers.

URL rewriting

URL rewriting works like a redirected URL as it stores the session identifier in the URL. The session identifier is encoded as a parameter on any link or form that is submitted from a Web page. This encoded URL is used for subsequent requests to the same server.

To enable HTTP session management:

Configure the server to enable session tracking.

Enable session management on the client by setting the JAX-WS property, javax.xml.ws.session.maintain, to true on the BindingProvider.

```
Map<String, Object> rc = ((BindingProvider) port).getRequestContext();

...     ...

rc.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);

...     …
```

3. Below is the code of a client who authenticate itself, request the list of services and invoke a service.

```
public class SMClient {
    private static EndpointReference targetEPR =
        new EndpointReference(
                      "http://localhost:8080/axis2/services/SMService");
  private static EndpointReference targetEPR2 =
        new EndpointReference(
                      "http://localhost:8080/axis2/services/WriteService");
public static OMElement readString(String myusr, String mypass, int mytime){
…..........................................................................................................................
}
public static OMElement getList(String myname, String SID){
…..........................................................................................................................
}
public static OMElement writeString(String myusr, String mypass, String myname){
…........................................................................................................................
}
public static OMElement writeStr(String nume, String SID, String cale, String continut){
            OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs = fac.createOMNamespace( "http://quickstart.samples/xsd", "tns");
        OMElement method = fac.createOMElement("WriteStr", omNs);
        OMElement value1 = fac.createOMElement("val1", omNs);
        value1.addChild(fac.createOMText(value1, nume));
        method.addChild(value1);
            OMElement value2 = fac.createOMElement("val2", omNs);
        value2.addChild(fac.createOMText(value2, SID));
        method.addChild(value2);
          OMElement value3 = fac.createOMElement("val3", omNs);
        value3.addChild(fac.createOMText(value3, cale));
        method.addChild(value3);
             OMElement value4 = fac.createOMElement("val4", omNs);
        value4.addChild(fac.createOMText(value4, continut));
        method.addChild(value4);
          return method;
  }
    public static void main(String[] args) {
        try {
          String myuser=new String("eu");
          String mypass=new String("@#");
          int mytime=1;
          String myname="Fat Frumos";
          OMElement writeString = writeString(myuser, mypass,myname);
          OMElement readString=readString(myuser,mypass,mytime);
          Options options = new Options();
          options.setTo(targetEPR);
          options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
          ServiceClient sender = new ServiceClient();
          sender.setOptions(options);
          OMElement result = sender.sendReceive(readString);
```

```
                Iterator it=result.getChildElements();

                String SID="";

                int asdf=0;

                while (it.hasNext()){

                        OMElement str=(OMElement)it.next();

                        if(asdf==0){SID=str.getText();asdf=1;}

                        System.err.println("rasp:   "+str.getText() );

                }

                OMElement myList=getList(myname, SID);

                OMElement resultList = sender.sendReceive(myList);

                it=resultList.getChildElements();

                while (it.hasNext()){

                        OMElement str=(OMElement)it.next();

                        System.err.println("Lista:   "+str.getText() );

                }

        Options options2 = new Options();

        options2.setTo(targetEPR2);

        options2.setTransportInProtocol(Constants.TRANSPORT_HTTP);

        OMElement writeStr=writeStr(myname,SID,"/fis/file","ala bala portocala");

        ServiceClient sender2 = new ServiceClient();

        sender2.setOptions(options2);

                  sender2.fireAndForget(writeStr);

    } catch (Exception e) {

        e.printStackTrace();
        }
    }
}
```

4. The Authentication code:

```
public OMElement Authenticate(OMElement s) throws XMLStreamException
{
            s.build();
            s.detach();
            Iterator it=s.getChildElements();
            OMElement str=(OMElement)it.next();
            String myusr=str.getText();
            str=(OMElement)it.next();
            String mypass=str.getText();
            str=(OMElement)it.next();
            int mytime=Integer.parseInt(str.getText());
            String errmessS="";
              OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs =
            fac.createOMNamespace(namespace, "ns");
        OMElement method = fac.createOMElement("getStringResponse", omNs);
///////////////////Create file and folder, add first client//////////////
            NamespaceClient ns=null;
                        FileHandler fileHandler=null;
```

```
                int res;
                int err_code=0;
                String bd="";
                try {
                        ns=new
NamespaceClient("/home/mistify/Desktop/Licenta/NamespaceManager/conf/test2.cfg");
                } catch(IOException e){        }
                int da_eroare=0;
        Vector<FileMetadata> fmList = ns.listDir("/SMBD");
        if(fmList.size()==0) {
        err_code=666;
        errmessS="Not registred";
        }
        else{
                try {
                        fileHandler = ns.getFileHandler("/SMBD/Users");
                } catch(IOException e){}
        }
                String mylog="username=";
                mylog=mylog.concat(myusr);
                mylog=mylog.concat(" password=");
                mylog=mylog.concat(mypass);
                        int marime=216;
                ByteBuffer result = ByteBuffer.allocateDirect(marime);
                for(int i = 0; i < marime; i++)
                        result.put(i, (byte)0);
                fileHandler.read(0, marime, result);
                int nok=1;
                int offset=0;
                for(int i = 1; i < marime; i++)
                if((char)result.get(i)=='u'         &&         (char)result.get(i+1)=='s'         &&
(char)result.get(i+2)=='e'  &&  (char)result.get(i+3)=='r'  &&  (char)result.get(i+4)=='n'  &&
(char)result.get(i+5)=='a'  &&  (char)result.get(i+6)=='m'  &&  (char)result.get(i+7)=='e'  &&
(char)result.get(i+8)=='=') offset=i;
                for(int i =offset,j=0; j < mylog.length(); i++,j++){
                        char ccc=(char)result.get(i);
                        char cc=mylog.charAt(j);
                        if(ccc!=cc)
                        nok=0;
                }
        err_code=nok;
/////////////////////////////////////////////////////////////////////////////////////
        if(nok==1 && mytime!=0)    {
        int SID=20;
                String mysid=Integer.toString(SID);
         OMElement value1 = fac.createOMElement("return1", omNs);
      value1.addChild(fac.createOMText(value1, mysid));
      method.addChild(value1);
                int errcode=1;
                String errcodeS=Integer.toString(errcode);
         OMElement value6 = fac.createOMElement("return6", omNs);
      value6.addChild(fac.createOMText(value6, errcodeS));
      method.addChild(value6);
        errmessS="ok";
         OMElement value7 = fac.createOMElement("return7", omNs);
      value7.addChild(fac.createOMText(value7, errmessS));
      method.addChild(value7);
////////////////////Write to conf file/////////////////////////////////////
        Vector<FileMetadata> fmList2 = ns.listDir("/Conf");
        if(fmList2.size()==0) {
                res = ns.delete("/Conf/ConfFile");
                res = ns.mkDir("/Conf");
                try {
                        fileHandler = ns.createFile("/Conf/ConfFile", PAGE_SIZE, 1);
                } catch(IOException e){}
        }
        else{
                try {
                        fileHandler = ns.getFileHandler("/Conf/ConfFile");
                } catch(IOException e){        }
        }
                int finset=offset;
                offset=0;
```

```
			for(int i = 0; i < marime; i++)
			if((char)result.get(i)=='n'		&&		(char)result.get(i+1)=='a'&&
(char)result.get(i+2)=='m'   &&   (char)result.get(i+3)=='e'&&   (char)result.get(i+4)=='='   &&
i<finset)
				{offset=i;}
		String fisname="";
		for(int i=offset+5;i<finset;i++){
			char fs=(char) result.get(i);
			String fss="";
			fss=String.valueOf(fs);
			fisname=fisname.concat(fss);
		}
			String numeS="",adresaS="",descriereS="";
			int drAcc=0;
			String confData="<new><Client_name>";
			confData=confData.concat(fisname);
			confData=confData.concat("</Client_name><SID>");
			confData=confData.concat(mysid);
			confData=confData.concat("</SID><valid_from>");
			Calendar cal = Calendar.getInstance();
	SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy kk:mm:ss");
	String valid=formatter.format(cal.getTime()).toString();
			confData=confData.concat(valid);
			confData=confData.concat("</valid_from><exp_date>");
			cal.add(Calendar.DATE, mytime);
```

```
String exp=formatter.format(cal.getTime()).toString();
        confData=confData.concat(exp);
        confData=confData.concat("</exp_date><ACL>");
offset=finset+mylog.length();
        finset=offset+9;
        String fisdr_acc="";
        for(int i=offset+8;i<finset;i++){
                char fs=(char) result.get(i);
                String fss="";
                fss=String.valueOf(fs);
                fisdr_acc=fisdr_acc.concat(fss);
        }
        int dr_acc=Integer.parseInt(fisdr_acc);
        int inc=finset;
        while(dr_acc>0){
                dr_acc--;
                offset=0;
                for(int i=inc;i<marime;i++)
                if((char)result.get(i)=='n'  &&  (char)result.get(i+1)=='u'  &&
(char)result.get(i+2)=='m'  &&  (char)result.get(i+3)=='e'&&  (char)result.get(i+4)=='S'&&
(char)result.get(i+5)=='=')
                {offset=i;break;}
                finset=0;
                for(int i=inc;i<marime;i++)
```

```
            if((char)result.get(i)=='    '    &&(char)result.get(i+1)=='a'        &&
(char)result.get(i+2)=='d' && (char)result.get(i+3)=='d' && (char)result.get(i+4)=='r')
            {finset=i;inc=i+1;break;}
            String fismynameS="";
            for(int i=offset+6;i<finset;i++){
                char fs=(char) result.get(i);
                String fss="";
                fss=String.valueOf(fs);
                fismynameS=fismynameS.concat(fss);
            }
            offset=finset;
            finset=0;
            for(int i=inc;i<marime;i++)
            if((char)result.get(i)=='    '    &&(char)result.get(i+1)=='d'        &&
(char)result.get(i+2)=='e' && (char)result.get(i+3)=='s' && (char)result.get(i+4)=='c')
            {finset=i;inc=i+1;break;}
            String fismyaddrS="";
            for(int i=offset+7;i<finset;i++){
                char fs=(char) result.get(i);
                String fss="";
                fss=String.valueOf(fs);
                fismyaddrS=fismyaddrS.concat(fss);
            }
            offset=finset;
            finset=0;
```

```
        for(int i=inc;i<marime;i++)
        if((char)result.get(i)=='      '      &&(char)result.get(i+1)=='n'             &&
(char)result.get(i+2)=='i' && (char)result.get(i+3)=='v' && (char)result.get(i+4)=='_')
        {finset=i;inc=i+1;break;}
        String fisdescS="";
        for(int i=offset+7;i<finset;i++){
                char fs=(char) result.get(i);
                String fss="";
                fss=String.valueOf(fs);
                fisdescS=fisdescS.concat(fss);
        }
        offset=finset;
        String fisniv="";
        for(int i=offset+9;i<offset+10;i++){
                char fs=(char) result.get(i);
                String fss="";
                fss=String.valueOf(fs);
                fisniv=fisniv.concat(fss);
        }
        confData=confData.concat("<Nume_serviciu>");
        confData=confData.concat(fismynameS);
        confData=confData.concat("</Nume_serviciu>");
        confData=confData.concat("<Adresa_servicu>");
        confData=confData.concat(fismyaddrS);
```

```
                confData=confData.concat("</Adresa_servicu>");
                confData=confData.concat("<Descriere_servicu>");
                confData=confData.concat(fisdescS);
                confData=confData.concat("</Descriere_servicu>");
                confData=confData.concat("<Drepturi_Access_servicu>");
                confData=confData.concat(fisniv);
                confData=confData.concat("</Drepturi_Access_servicu>");
}
        confData=confData.concat("</ACL>");
        confData=confData.concat("</new>");
        confData=confData.concat("  ");
        ByteBuffer buf = ByteBuffer.allocateDirect(confData.getBytes().length);
        for (int i = 0; i < confData.getBytes().length; i++)
                buf.put(i, (byte)0);
        for(int j=0; j<confData.getBytes().length; j++) {
                char c=confData.charAt(j);
                buf.put(j, (byte)c);
        }
        int rest=confData.length()%PAGE_SIZE;
        int rest2=PAGE_SIZE-rest;
        while(rest2>0){
                confData=confData.concat(" ");
                rest2--;
        }
```

```
                boolean ok;

                ok = fileHandler.append(confData.getBytes().length, buf);

        }

        else    {

                err_code=10;

                String errcodeS=Integer.toString(err_code);

         OMElement value6 = fac.createOMElement("return6", omNs);

    value6.addChild(fac.createOMText(value6, errcodeS));

    method.addChild(value6);

        errmessS="invalid username or password";

         OMElement value7 = fac.createOMElement("return7", omNs);

    value7.addChild(fac.createOMText(value7, errmessS));

    method.addChild(value7);

        }

    return method;

}
```