UPTEC X09 004

Examensarbete 30 hp
Februari 2009

Arena integration web service provider

Indra Sharma

UPPSALA UNIVERSITET

Bioinformatics Engineering Program

Uppsala University School of Engineering

UPTEC X 09 004	**Date of issue 2009-01**
Author **Indra Sharma**	
Title (English) **Arena Integration Framework Web Service Provider**	
Title (Swedish)	
Abstract In this thesis I have researched and developed a general web service provider that integrates to the Arena Integration Framework (AIF). The web service provider conforms to the existing AIF interface components; Wrapper, Factory and Adapter. A call from the web service generates wrappers that are passed through the FAME architecture. The web service provider can also receive a reply from FAME if needed. The web service provider will be used to simplify communication between different parties.	
Keywords Web Service, Provider, Axis2/c, Wrapper, Factory, Adapter	
Supervisors **Oscar Rotel** **SunGard Front Arena**	
Scientific reviewer **Olle Erikson** **Department of information technology** **Uppsala University**	
Project name	Sponsors
Language **English**	Security
ISSN 1401-2138	Classification
Supplementary bibliographical information	Pages **36**

Biology Education Centre Biomedical Center Husargatan 3 Uppsala
Box 592 S-75124 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 555217

Arena integration framework web service provider

Indra Sharma

Sammanfattning

Front Arena stöder dagligen kommunikation mellan bl.a. börser, banker och deras kunder. Detta sker genom att information skickas mellan olika system genom Front Arena Mapper Engine arkitekturen. För att förenkla och utöka detta kommunikationssystem vore det önskvärt att t.ex. en order kunde läggas genom en webbsida på internet. För att kunna genomföra detta behövs en webbtjänsttillhandahållare.

En tillhandahållare sköter kommunikationen mellan två system. Genom att kombinera en tillhandahållare med en webbtjänst kan meddelanden skickas mellan olika parter från internet till vald destination. För att erhålla en webbtjänsttillhandahållare krävs forskning kring webbtjänster, servrar och designmönstren Fabrik, Adapter och Meddelande.

Den här rapporten beskriver arbetet bakom utvecklandet av en webbtjänsttillhandahållare som ska kunna användas i Front Arenas ramverk. Målet har varit att implementationen ska följa SunGards standard och göras i C++ samt att webbtjänsttillhandahållaren ska vara generell och dess användningsområden skall kunna variera. Rapporten täcker hur detta har åstadkommits och vilka hinder som har uppstått under utvecklingsarbetet.

Examensarbete 30 hp
Civilingenjörsprogrammet Bioinformatik
Uppsala Universitet januari 2009

Contents

Abstract

Sammanfattning

Contents

1 Introduction

Just as the sound of its name web services are services [1, 2] offered through the web. Today there are several thousands of web services on the internet, for example websites where you can buy cloth, books etc. are all web services. Web services have good usability and the customer does not need servers of its own. Support is provided by the company offering the web services. Additionally, web services can be cost-effective because software doesn’t need to be installed.

When one thinks of stock brokers handling orders one often imagine them screaming to each other through telephones. This is one way of communicating but there are other and more advanced solutions. Today orders can be handled by integrated systems where an order can be placed and forwarded to its destination without having to pass through a stock broker [3]. These systems can, in combination with web services, be used to simplify communication between customers and banks.

If there was a possibility to combine the integrated systems with web services it could be possible to log on to a webpage and place an order through the internet. Before this can be done the order needs to pass from the internet into the system. The order should also be understandable to the system. When placing an order through a web service the order will use a specific protocol that the system does not understand. The order needs to be translated into a different protocol before the message can be forwarded to the destination. This is where a Provider is needed.

Providers are located between systems and are used to forward messages between them. A Provider understands two types of protocols and can translate one protocol into another. It translates the incoming message into the protocol that fits the system the message is aimed for and forwards the message. If a reply to the incoming message is sent the Provider translates and forwards it again.

The purpose of this degree project is to (i) investigate how a general web service provider, that handles incoming and outgoing messages, can be implemented, and (ii) implementing the web service provider. Instead of communication between two systems we are now looking at one system communicating with an outside client. When implementing a web service provider there are several aspects that need to be researched. One aspect is how the web service provider should communicate with the outside client. Furthermore, an investigation considering how a connection between the web service provider and the client can be established is essential.
In the end, a web service needs to be implemented and a test application that demonstrates the use of the web service provider should also be built.

2 Background

2.1 SunGard Front Arena and the Front Arena solution

SunGard Front Arena is an operating unit of SunGard Data Systems which is a part of SunGard. SunGard Front Arena is one of the leading companies in risk management and financial software development. The company was originally named Front Capital Systems.

SunGard Front Arena is a world wide provider of financial solutions and has 300 employees. Its headquarters is located in Stockholm and it has offices located in London, Frankfurt, Zürich, Johannesburg, New York, Los Angeles, Chicago, Toronto, Singapore, Tokyo, Hong Kong and Sydney [4].

The product Front Arena was introduced in 1999 and is used by many of today's most advanced financial institutions, a few examples are; Carnegie, SEB, Handelsbanken, Nordea, Swedbank, OMX, MGM and Avanza.

Front Arena is an open integrated solution for risk management, sales, trading, operations and distributions [5]. The system is very flexible and Front Arena can be used for all traded financial products, both liquid and illiquid, OTC and exchange-traded, standardized and structured. Many types of operating systems and protocols such as FIX, SQL, Python, FpML are supported and so are frameworks from Java to .NET.

2.2 Integration Development and Integration Frameworks

The degree project was made in the workgroup Integration Frameworks which is a part of the department Integration Development. Integration Development is responsible for development, maintenance and support. There are three teams in Integration Development named Integration Market, Integration Frameworks and Pune team. Manager of Integration Frameworks is Antonio Pulido and the group has six other members: Oscar Rotel, Christoffer Sabel, Lars Mattsson, Malin Olsson, Inken Koch and

Prasad Balagopalan. Manager of the department Integration Development is Fredrik Söderberg.

2.3 Design Patterns

When developers encounter a problem they tend to search for an already existing solution instead of resolving the problem once again. A Design Pattern is an abstract description of a problem solution and can be applied in many different ways [6]. Adapter, Factory and Wrapper are design patterns.

2.4 FAME architecture

Front Arena Mapper Engine (FAME) is a programming architecture for system integration based on C++ developed by Joakim Hjerpe. FAME architecture is used for connecting two systems to each other and transferring messages between them. The messages passed between the systems are plain text files, mapping files, containing information [7].

Following are the main elements of the FAME architecture:

- Mapping files
- Mapper Engine
- Provider
- Provider Manager
- Adapter
- Wrapper
- Factory
- DB Connections

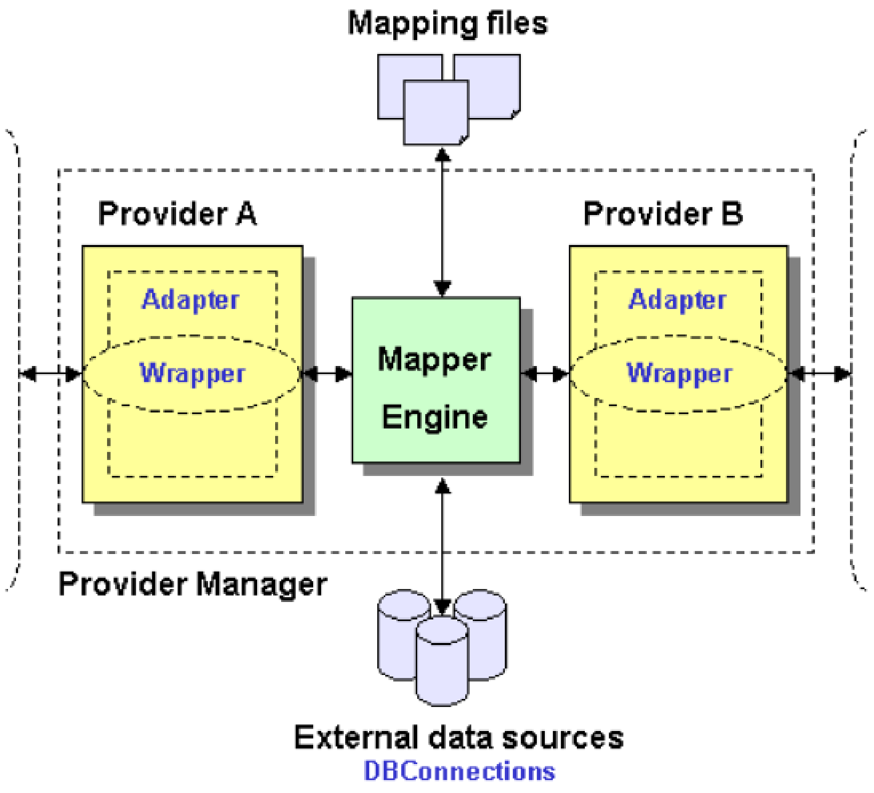


Figure 1. Main elements of the FAME architecture. Source [7], Illustration used with permission from Antonio Pulido.

As shown in figure 1, the heart of the FAME architecture is the Mapper Engine that reads the information inside the mapping files. Depending on the information inside the mapping files the Mapper Engine is able to transfer messages between the Providers. Below is an example that illustrates what a mapping can look like using the web service provider and FIX protocol provider [8] :

TYPE	OrderSingle	WSEnterOrder
	Symbol	WSSymbol
	ClientID	ID
	minQty	Qty

Example 1. *Illustration of a mapping file.*

Example 1 shows what a mapping file might resemble. The mapping file is interpreted by reading from right to left. Example 1 illustrates how an incoming message (from the web service provider) named 'WSEnterOrder' will be mapped to another message 'OrderSingle' made for the FIX provider. The fields of 'WSEnterOrder' will be mapped to the field name to the left. For example 'WSSymbol' will be named 'Symbol' in the 'OrderSingle' message.

Providers are used in the FAME architecture to help systems communicate with each other. A provider is used for communications within the FAME architecture and also for communication with the connected systems on the outside. A provider is defined by the following four classes; Adapter, Factory, Wrapper and Wrapper Descriptor.

The provider manager works as a supervising module and can be seen as the main program or the executable (exe) file. It loads the mapper engine, the mapping files and the providers.

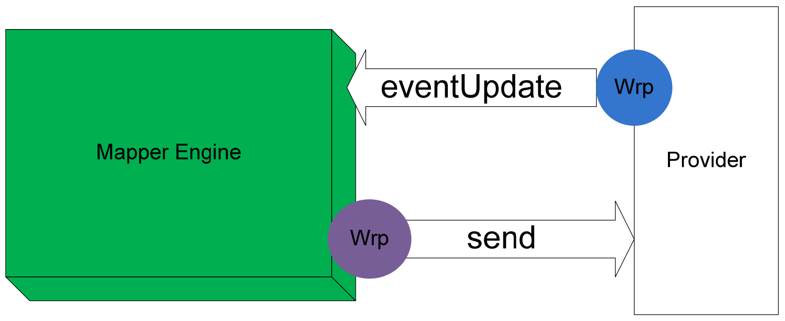


Figure 2. *The method eventUpdate(Wrapper wrp) sends a message to the mapper engine. The mapper engine sends a reply to that message and the provider receives it with the method send(Wrapper wrp).*

The adapter is responsible for the communication; it sets up connections with the FAME architecture and outside systems. The adapter has three methods: open(), close() and send(). open(), establishes a connection, close() closes the connection and send() receives messages (wrappers) from the mapper engine. The adapter has a component named the Adapter Listener which listens for messages from the adapter and forwards them to the mapper engine. The adapter listener has a method called eventUpdate(). eventUpdate() is for sending messages to the mapper engine for mapping (see figure 2).

The wrapper encapsulates information that is being sent between the adapter and the mapper engine.

The factory creates and describes the wrappers and can be seen as “wrapper factories”. All the wrappers that can be created are defined in the factory.

DB Connections (Database Connections) are needed external sources such as PDB files, SQL databases and text files.

Suppose a wrapper from the web service provider is passed on to the mapper engine with the method eventUpdate(). The wrapper type (wrapper name) is WSEnterOrder and the mapper engine will create another wrapper with the wrapper type OrderSingle. OrderSingle is a message for the FIX adapter. The mapper engine will also fill the field Symbol in OrderSingle with the data that is in the field WSSymbol in WSEnterOrder. The same will be done for all the other fields. When the mapping is completed the Wrapper is sent to the FIX Adapter that receives the OrderSingle wrapper with the method send().

2.5 XML and SOAP

XML stands for ‘Extensible Markup Language’ and is designed to exchange information between different systems. An XML file is a pure text document describing the text, meta data and the logical structure of a document [9]. Below is a simple example of what an XML document can look like:

```
<exjobb xmlns=" http://www.ibg.uu.se/">
  <name>Indra Sharma</name>
```

```
   <project> Arena Integration Framework Web Service Provider </project>
   <company>SunGard Front Arena</company>
   <University>Uppsala</University>
</exjobb>
```

SOAP is a protocol for exchanging information and it is based on XML. SOAP can be used with several types of protocols such as HTTP etc. SOAP is one of three building blocks used for web services [10]. Before SOAP was an acronym for ‘Simple Object Access Protocol’ but is no longer an acronym.

2.6 The Apache Software Foundation and Axis2/c

The Apache Software Foundation offers free open source code for software development [11]. Open community developers are the ones maintaining Apache, they have several projects and one of them is Axis2/c, a web service engine implemented in the programming language C [12]. The latest release of Axis2/c is 1.5.0 and one of its functionalities is to provide web services. SOAP is supported in Axis2/c and so is HTTP transport.

2.7 Axis2 HTTP Server

This is a server that can be downloaded from the axis2/c homepage. This project has used version 1.5.0 of axis2/c.

There are two releases, binary and source, of version 1.5.0. The binary distribution is precompiled and ready for installation while the source release needs to be compiled before it can be installed on the computer. The source distribution can be compiled in two ways, by using a makefile and following the instructions on the axis2/c homepage or by using the axis2/c solution (axis2c.sln) and compiling in Visual Studio.

A basic configuration of Axis2/c requires libxml2, iconv and zlib which are external libraries, and also nine different VC projects [13, 14]:

1. *axutil*, contains utility functions that are required by Axis2/c. It generates a dynamic-link library (dll). This library contains logging functions, string operations, error handling etc. It is also linked to all web services and clients written for axis2/c and to all libraries.
2. *axis2_parser*, generates a dll and is needed since it provides a parser abstraction layer.
3. *axiom,* generates a dll and helps to manipulate XML and is the XML model layer.
4. *neethi,* is the axis2/c web service policy and security policy.
5. *axis2_engine*, is the core of axis2/c and encapsulates the main functionalities.
6. *axis2_http_sender*, is the http sender of the engine and generates a dll.
7. *axis2_http_receiver,* is the http receiver of the engine and generates a dll.
8. *axis2_http_ server*, contains a main method that generates the server which can be used for deploying services. It generates an exe.
9. *axis2_mod_addr*, implements web service addressing specification.

2.8 Microsoft IIS and Apache HTTP Server

Apache HTTP Server is the most popular web server in the world [15]. It has played a key role in the development of the World Wide Web and serves over 50% of all web sites. Apache HTTP Server is distributed by the Apache Software Foundation but also IBM distributes a version of Apache HTTP Server. Apache HTTP Server can be used on windows. The web applications used for Apache HTTP Server does not exist for C or C++ web applications.

Microsoft IIS (Internet Information Services) is a windows web server distributed by Microsoft [16]. It is the secondly highest ranked server in the world.

Apache HTTP Server has, in contrast to Microsoft IIS, several features which are free.

2.9 Visual Studio, debug and release mode

Visual Studio is a development environment that can be used when coding in, for example, C and C++ programming language. When compiling in Visual Studio the compilation can be made in debug or release mode.

Debug mode generates executables containing debug information. It helps to create an optimal environment for the Visual Studio debugger.

Release mode generates the smallest possible executable that contains the minimum of information needed to create an executable. Usually the product offered to a customer contains executable compiled in release mode so that only the necessary information is passed on.

3 Materials and methods

3.1 Analysis

The main goal of the web service provider is to provide a simple way of communication between two parties. It is important to keep it general so that it can serve a client in several ways. It should be usable when communicating with a stock exchange but also for simple administrative tasks. It should have good usability and integrate into Front Arena's Integration Framework (AIF). Any functionality of the AIF should be developed, designed, operated and delivered independently of other components [17].

3.2 The server

The first step was to decide how the web service provider would communicate with the outside clients. There where two options to explore:

A) The provider would connect itself to an outside server such as Apache HTTP Server or Microsoft IIS where a web service is stored. The client would then connect and send its request through the server, using the web service, to the provider. The provider would send its reply to the client in the same way.

B) Using open source code distributed on the internet for a mini-server and compiling it. After compiling the server it can be converted into a dynamic-link library (dll) which can be embedded inside the provider.

Solution A is possible but not optimal because the user of the provider needs to install a server which contains superfluous libraries. There is also a question how the connection between the provider and the server can be established. All necessary features of Microsoft IIS are not free of cost, while Apache HTTP Server is completely free. Apache HTTP Server does not support web applications in C or C++ which is the SunGard company standard.

Option B is in theory simpler than A since the above mentioned problems don’t occur. The advantage of option A is that the server does not need to be maintained. In option B the maintenance is the users’ responsibility.

Assuming that all the SunGard providers are implemented in C++ the web service provider should follow the company standard. Investigations showed that Apache Software Foundation offers open source code for servers and that there is a project named Axis2/c that distributes code for a server, Axis HTTP Server. As open source code is not always reliable the best way to test if the Axis HTTP Server works was to install the latest binary release and test it with the provided examples. By following the instructions on the Axis2/c homepage the server source code was successfully compiled and installed after testing the binary solution. The server worked well with the provided examples of web services and clients.

To be capable to embed the server inside the Provider the axis2/c open source project needed to generate a dll instead of an exe. The server needs to be modified in such a way that it can be run just by using a dll. The above mentioned could be accomplished in two ways; by changing the project makefile or by trying to compile the axis2c.sln in Visual Studio (VS). The optimal method is using VS to compile axis2c.sln in debug mode because this environment enables debugging in a simple way. Changing the project and generating a dll would also be easier in VS where this can be achieved by simply changing the project property configuration type. The axis2c.sln was made for VS 2003 and needed to be converted to VS 2008. Furthermore, paths to the included files and libraries were added. In order to use VS for compilation the server needed to be recompiled.

When compiling the solution file many errors were encountered. By turning to the Axis2/c mailing list assistance was provided for these errors by the developers at Apache Software Foundation [18].

3.3 The provider

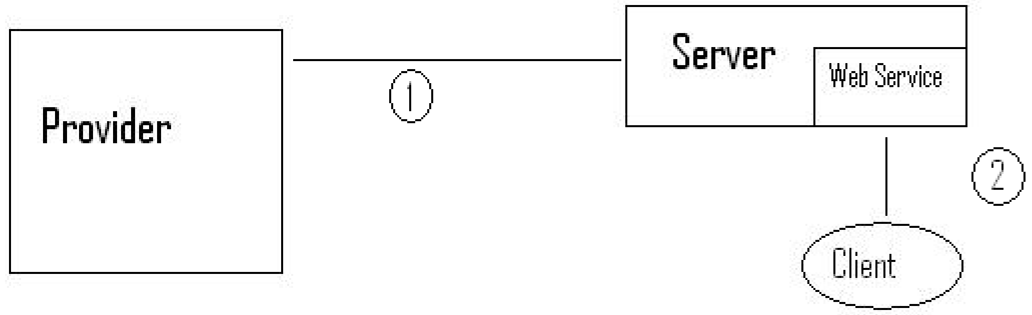


Figure 3.. *The map illustrates the connections and parts that are needed to obtain a web service provider.*

In figure 3, number one illustrates the connection between the provider and the server while number two shows the connection between the embedded web service and the client. The first connection is achieved by taking the main method of the Axis2c HTTP Server and modifying it into a C++ class in the provider. The server can then be started by creating an instance of that class. The instance can access the method that initiates the server.

Next step was the implementation of the web service provider. It was done by first implementing a provider without the web service. The provider generated a Wrapper every 10 seconds and the Wrapper contained two fields, message and counter. Every Wrapper was then filled with the message “Hello” and a number which informed on how many Wrappers were generated.

The implementation of the provider includes implementation of the methods send(), open() and close() in the adapter. The idea was to start a thread, in the method open(), that generates wrappers. The thread class was implemented with the methods; run(), start(), stop() and wait(). The start() method calls the run() method which starts generating wrappers. The stop() method stops the thread and the wait() method which waits until the thread has stopped. The idea behind this is not to start a thread with open() but to start the axis server. In the Axis2/c open source code a main method is used for starting the server. Using the same main method, but making it into a class instead, gave the possibility to generate objects that can start the Axis HTTP Server. Instead of a thread an object is created that starts the server inside the open method. The advantage is that the server can now be initiated from the provider.

3.4 Modifying the server

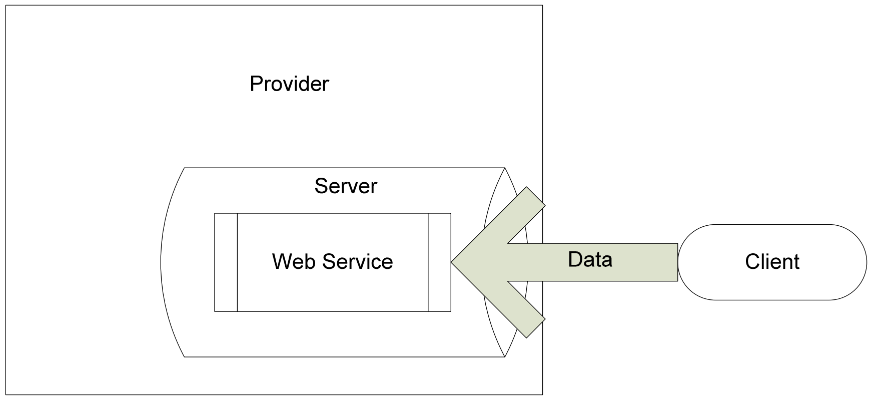


Figure 4. *An Illustration of data transfer from the client to the provider. The server and the web service is embedded inside the provider.*

As illustrated in figure 4, data are sent from the client to the web service. The data should be wrapped inside a wrapper and passed through FAME. This can be done if wrappers are generated and filled inside the web service where the client data is stored. To enable this procedure, the web service should have access to the Adapter Listener and the Factory class. The question is how to give the web service access to these classes?

Another issue is that the web service is written in C programming language and not in C++. Objects do not exist in C programming language which makes it impossible to create wrappers objects in the web service.

There are several possible solutions on how to generate wrappers from the web service. One possibility is to compile the server in C++ instead of C and correct the errors that occur. This could be a suitable solution if the errors are few. However, in this case it is the opposite. Another possible solution is to change the client interface. This would, however, imply that the structure of the Axis HTTP Server would no longer be the same as before. Then axis2/c support might not provide us with help needed in the future.

There is a third solution where a callback function is created in the adapter. If the web service has access to this function it can call it, pass the client data and the callback function can generate wrappers from the adapter. The issue is how to give the web service access to the callback. This is achieved by a variable (env) that is created in the server initiation class and passed on all the way to the web service. By aggregating a pointer to the callback function to env the callback function can be reached from the web service through env.

The callback function was implemented to receive two parameters; the name of the operation (the wrapper name) and an array containing the client data. It creates a wrapper with the same name as the operation name and fills it with the client data.

3.5 The Web Service Provider

Until now one of the web services provided by axis2/c has been used. But the web service provider needs a web service that offers operations that are accurate for SunGard and its customers. Therefore, a new web service was implemented. The client creates a node containing the client data and passes it to the web service. The web service creates an array containing the node data and calls the callback with the operation name and the array. The callback creates a wrapper and calls the method eventUpdate() in the Adapter sending the wrapper to FAME (see figure 5).

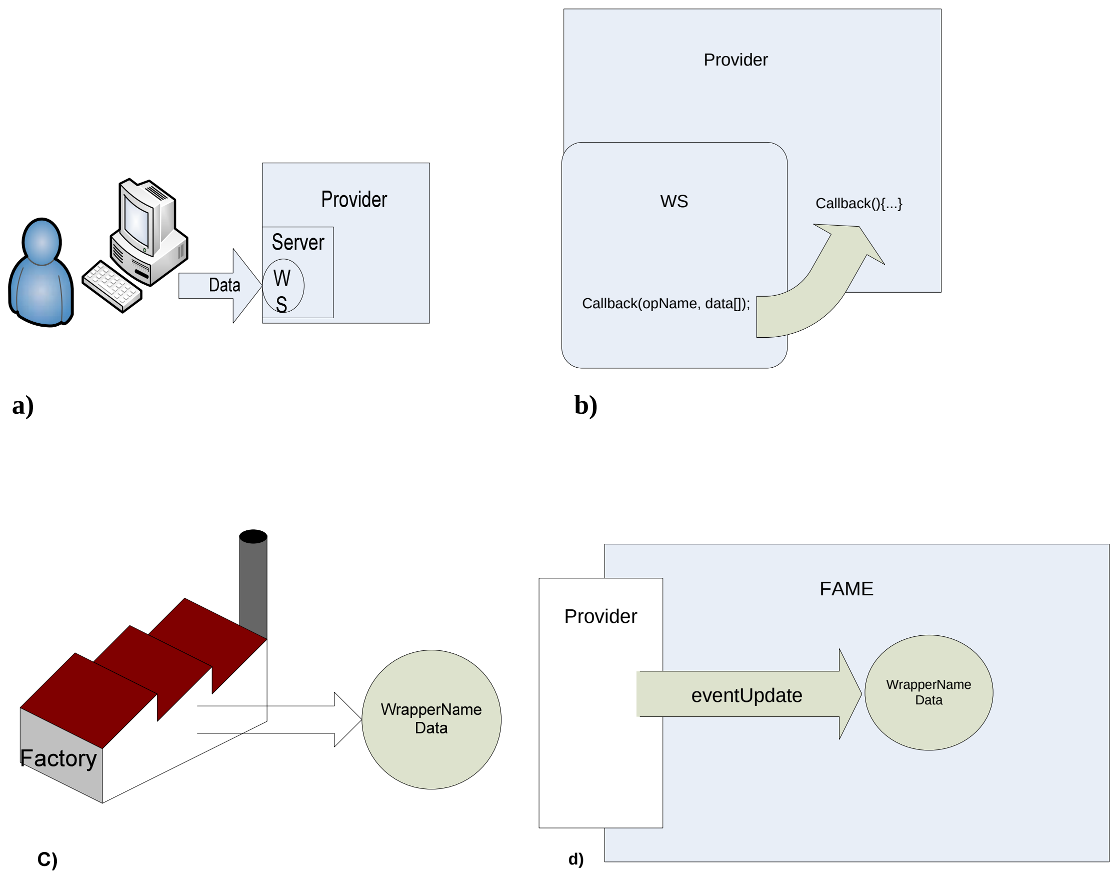


Figure 5.

a) A client enters a message with data over the internet. b) The message reaches the web service and the callback function is called with the operation name and the message data. c) The callback function has access to the Factory and tells it to create a wrapper with the same name as the operation name. The wrapper is filled with the message data. d) The Provider calls eventUpdate() and the wrapper is passed to FAME.

The web service provider sends the message to FAME which forwards the message to its destination. A downside is that the client doesn't know if the wrapper was sent or not. Hence, a reply should be returned to the client and stating that communication was error free. On the other hand, if an error occurs the client should receive a message with the information that there was an error and also what type of error. This was implemented by letting the callback return an array containing the error data. This array is returned to the web service that takes the array data and transfers it to a node. The node is then returned to the client. Basically the communications between the client and the provider works in the same way when sending and receiving data.

The web service provider now works for messages that don't expect a reply from FAME. However, this is not a general web service provider that can handle any type of messages. There are messages that will need a reply from its destination. This reply is passed from the destination through FAME to the web service provider by the method send(). This is illustrated by the yellow and the red arrow in Figure 6. At the moment only the yellow arrow is implemented in the web service provider. The method send() in the adapter needs to be implemented to receive messages (wrappers) from FAME.

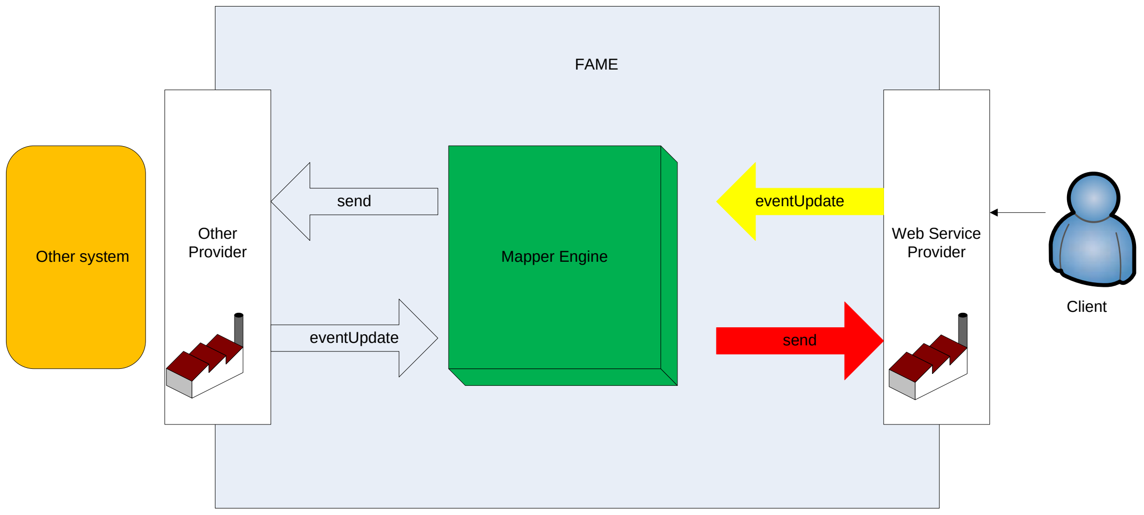


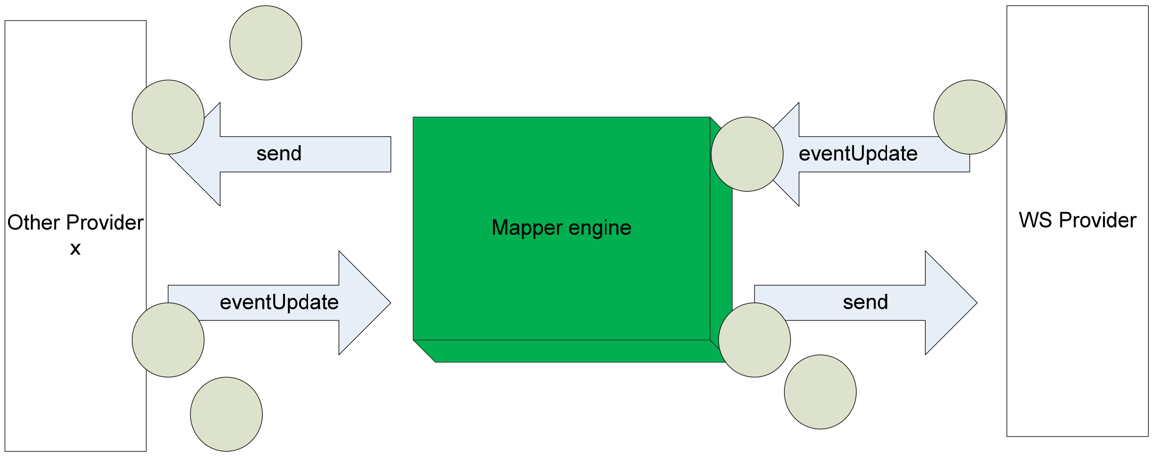
Figure 6. *A representation of the communication between the client and the other system. The arrows illustrate in which direction and with what method the wrappers are passed. eventUpdate() is for sending a Wrapper to be mapped in the mapper engine while send receives a mapped message from the mapper engine.*

When implementing the send() method there is a dilemma. The wrapper that is passed by the mapper engine and then received by the web service provider with the method send() is a reply to a message that was sent with eventUpdate(). The dilemma is how send() will connect the incoming message to the message that was sent (figure 7 a) with eventUpdate() . There could be several clients that sent messages at the same time and then there will be several replies. There system must provide a way to determine which reply goes to which client. There must also be a way for the provider to distinguish when there is a client message that needs a reply and when no reply is needed.

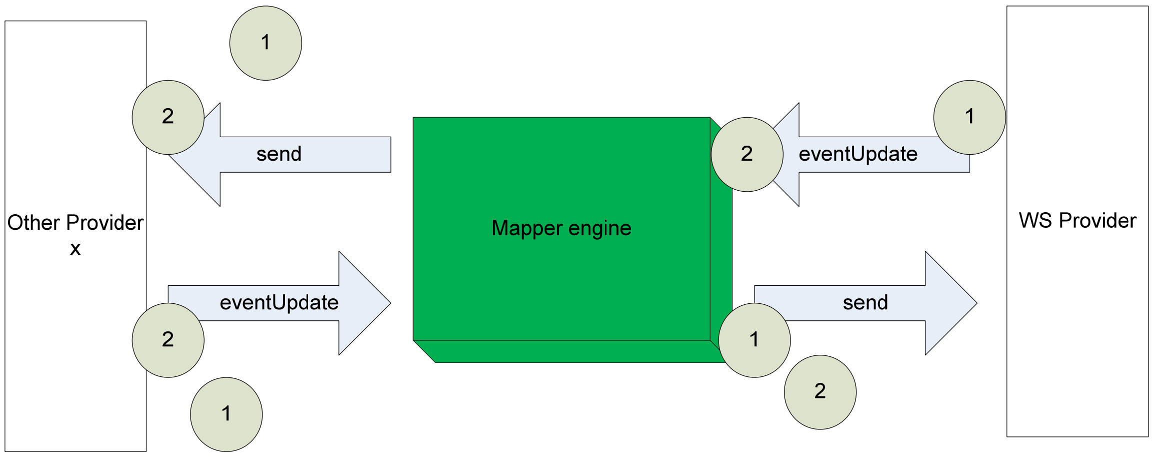
When the provider starts it reads wrapper descriptions from text files. By that means the provider and its factory knows what types of wrappers are defined and can thereby be created. The wrapper descriptions contain the wrapper name and the names of the fields that can be filled with data. Wrapper synchronization can be achieved by adding an extra identification field to the wrapper (figure 7 b). By adding a synchronization identification (SynchId) field a number can be assigned in that field. This number must be unique for every wrapper that is created by the web service provider factory. When the web service provider calls eventUpdate() for that wrapper the mapper engine maps the data and the SynchId field will also be mapped. When the other provider calls eventUpdate() on the reply the same number will be filled in the identification field of that wrapper. When the mapper engine receives the reply it will be mapped again and the id field value will be noted in the SynchId field.
An example of the mapping file for the web service provider and another provider, x:

```
TYPE  xEnterOrder   WSEnterOrder
      xSymbol       WSSymbol
      xID           ID
      xQty          Qty
      xSynchId      SynchID
TYPE  WSEnterOrder        xEnterOrder
```

WSSymbol	xSymbol
ID	xID
Qty	xQty
SynchID	xSynchId



a)



b)

Figure 7. a) *eventUpdate() is called on wrappers by the web service provider. The replies can't be connected to the original message. This is a problem when several messages are sent at the almost exact time.* **b)** *If a synchronization identification field is added it is possible for the web service provider to connect the original message with the belonging reply.*

The SynchId field can also be used to distinguish which messages need a reply. Wrappers that don‘t have the SynchId field will not need a reply and as soon as eventUpdate() is called the client will receive a confirmation message that tells if the wrapper was successfully sent.

4 Result and discussion

4.1 The Server

SunGard Front Arena has several requirements that need to be fulfilled before the web service provider can be released to a customer. This also affects the server because it is a part of the web service provider. The most important features for the server that will be used with the web service provider is that it should be in C or C++ programming language and that it is easy to deploy. The axis2/c HTTP Server fulfills these requirements. On the other hand, there are characteristics that could be improved. Axis2/c HTTP Server expects to have two environment variables to be set before it can run [19]. This is a feature that could be improved because it complicates the deployment for the customers. The ideal solution is one that is as simple as possible to install.

There were some small modifications made in the server code which is not optimal. The changes where made because of the differences between the programming languages C and C++. The optimal solution would not allow any changes made in the server. If a newer version of Axis2/c HTTP Server is released and the web service provided should be upgraded and use this new version the same modifications need to be made again. If the changes are not made the web service provider may not work. The best way would be to have a server that doesn't require any changes made to the code. This can be achieved by recompiling the server in C++ instead of C. I have tried the above mentioned but it caused many compiling errors that would take much too long to solve. Even if the changes mentioned would be made the solution is still not optimal seeing that the same problem would be encountered when it is time for upgrading the server to a newer version. There is another problem; Axis2/c support might not provide help for problems

with the server since it has been modified. The lack of support could become a problem since the maintenance of the web service provider is SunGard Front Arenas responsibility.
Another solution would be using a server written in C++ instead of C. To the best of my knowledge there is no such server. Further investigation could solve this issue.

A third solution is to go around the problem and give the web service access to the callback method in some other way that does not require changes made to the server code. This might be something that the Axis2/c support group could help with and give an answer to.

In the server initiation class there is a method named init(). It is by calling init() that the server is initiated. But there is a possibility that there will occur an error and that the server doesn't start. When that error occurs the error message is not sent to the client and the client will not interpret the cause of the server not starting. Therefore there should be a way to tell the provider if the server has started or not so that the provider can notify the client if an error has occurred. It could be accomplished by using a flag. To set the flag further modifications to the server would be needed which is not optimal. The optimal solution would be if there existed a method in the server that verifies if the server has started or not. I have asked the Axis2/c support for guidance but haven't yet received a reply. The solution would be to contact the Axis2/c support again or to continue investigating the server code.

4.2 The Web Service Provider

The web service provider that has been developed fulfills SunGard's requirements. It is easy to deploy and has good performance. There is no need to install anything else than the web service provider. The web service provider has been developed, designed and can be used independently from the other components in the AIF.

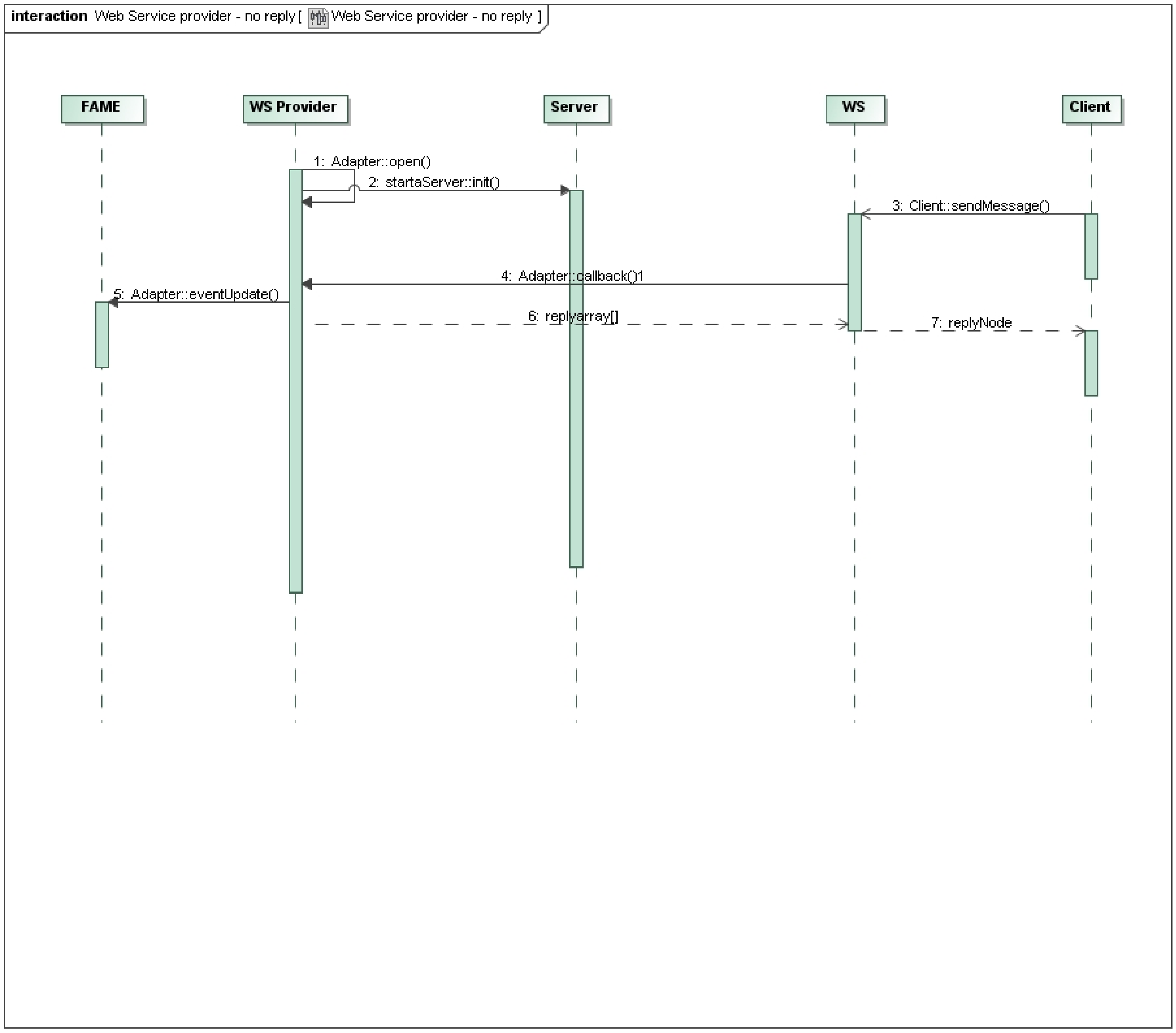


Figure 8. *Sequence diagram that illustrates interaction between the web service provider and other units. The message sent by the client is not expecting a reply from FAME.*

The web service provider is opened (1) and an instance of the server initialization class is created. The instance calls the function that initiates the server (2). Now the server is running and waiting for a client to connect. A client connects and sends a message (3) that is to be forwarded to FAME. The web service receives the client message and calls the callback function in the web service provider (4). The callback function takes an array, containing the client message, as a parameter. The callback method creates a wrapper containing the array values and calls eventUpdate() (5). The callback method checks if the wrapper has a synchronization identification field. In Figure 8 the wrapper does not contain such a field and the web service returns a reply array containing information on if the wrapper was successfully sent to FAME or not (6). The web service receives this reply and returns it to the client (7).

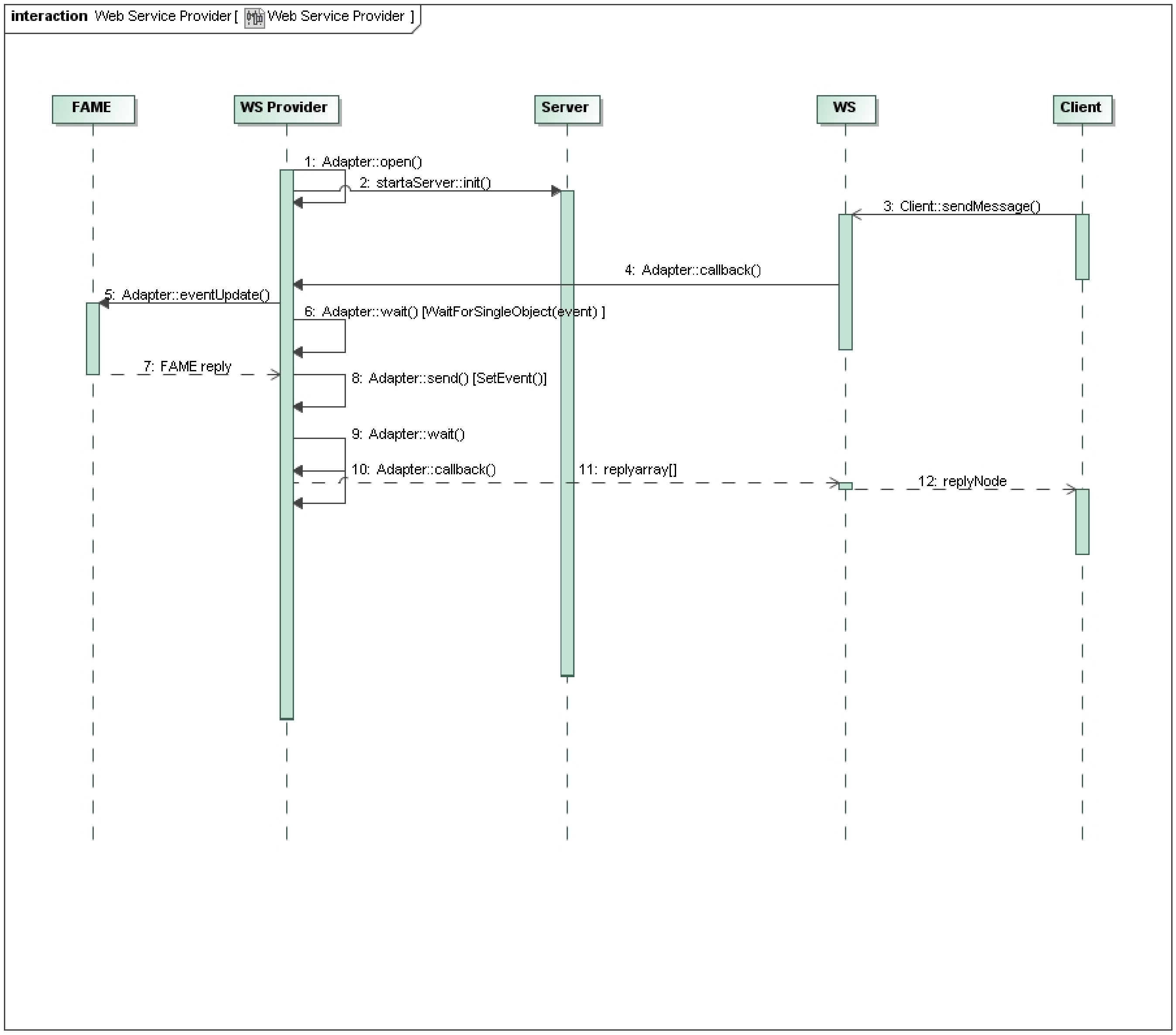


Figure 9. *Sequence diagram that illustrates interactions between the web service provider and other units. The message sent by the client is expecting a reply from FAME.*

Figure 9 shows how the web service provider operates when a reply is expected from FAME. The first five steps are the same as in Figure 8. The difference is that the wrapper created in the callback function contains a synchronization identification field which implies that a reply from FAME is expected. After eventUpdate() has been called the callback function calls another method named wait() (6). The method creates an event and enters the synchronization identification number and the event in a list (here: wrapper list). Then it calls a waiting function (Microsoft's WaitForSingleObject()) that waits for an event. FAME returns a reply (7) and the method send() receives it (8). The wrapper reply contains a synchronization identification number and the wrapper list is searched through until the matching synchronization identification number is found. Then the reply wrapper is added to the wrapper list. An event is set (Microsoft's SetEvent()) and the waiting function starts again (9). The wait() method returns the wrapper in the wrapper list to the callback method (10). An array containing the wrapper data is created and returned to the web service (11). The web service forwards the array information to the client.

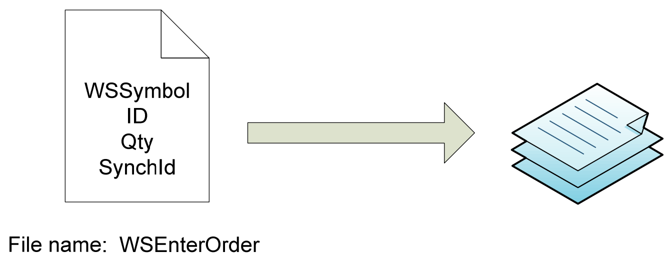


Figure 10. *The web service provider needs text files to describe the wrappers that the factory can create. For every type of wrapper there is one text file needed.*

The web service provider reads the wrapper descriptions from a text file and this could be improved. The web service could read the descriptions from an XML file instead. The difference is not big but it would improve the usability. One text file is needed for every type of wrapper which would make it quite many text files if there are many wrapper types. All wrappers can be described in one XML file. Another advantage using XML files is that more information could be entered for a wrapper. For example the name of the reply wrapper could be defined in the XML file. XML is more structured than a regular text file which increases its usability.

4.3 Test simulation

A small unit test was made to test that the web service provider can be used to send messages from a client to FAME and vice versa. This was achieved by simulating a client call and a response from FAME. The unit test tests the web service provider but not the client, the web service or FAME. The unit test shows that the web service provider is working as it should and that it can be used to forward messages from a client to FAME and also receive replies from FAME by using synchronization.

5 Conclusions

5.1 Usability

In my opinion the overall usability is good but there are still improvements that could be made. The modifications in the server code should be avoided. Further research on how to avoid these modifications is needed. The web service provider could also be modified to read wrapper descriptions from an XML file instead of from a text file. If these changes are made usability will increase.

5.2 Performance

The implementation of the code during this project has not been optimized. Fine tuning of the code could be an option. The performance is still good and the web service provider can be used to simplify communication between two parties. The generated dll:s should be tested together with another component. This has been done for the provider that does not receive a reply from FAME and the performance was good. It has not been tested for the final web service provider. The final web service provider has been tested with a unit test.

5.3 The project plan

The project plan worked out well and was accomplished. It was sometimes difficult to plan my time since some parts of the project plan where diffuse. This could have been avoided by having a more detailed project plan.

5.4 Future

The Arena Integration Framework Web Service Provider can be used for many things. Some of the areas where it could be used are to enter orders, administration and surveillance.

6 Acknowledgements

The last few months I have been working with my degree project at SunGard Front Arena. I have had the opportunity to develop as an engineer and a developer in a very creative environment. First of all, I want to thank my manager Antonio Pulido who has given me this opportunity and helped me feel as a part of his team from day one. I want to give special thanks to my supervisor Oscar Rotel for his guidance, time and support and my scientific reviewer Olle Eriksson for assisting me while writing my report and my thanks to Uppsala University and my classmates for the past years of great science. I also want to thank the rest of my group; Christoffer Sabel and Lars Mattsson, thank you for taking a special interest in me and providing me with so much help and support, Malin Olsson, Inken Koch and Lindsey Marshall, thank you for taking the time to read my report, helping me debug it and providing me with moral support throughout my time at Front Arena.

Several other people have played an important role by helping and supporting me. These persons are: Fredrik Söderberg, Daniel Roos, Prasad Balagopalan, Richard Asplund, Roger Persson, Aries Bucquet, Erik Bergman, Martin Terberyd, Per Isacsson, the rest of the team at SunGard Front Arena and the Axis2/c support group.

Sist men inte minst vill jag tacka mamma, pappa och min bror Aron som stöttat och trott på mig genom hela min utbildning. Sedan vill jag också tacka min pojkvän Anders för hans tålamod som aldrig sinar och hans tro på mig.

7 References

1. McIlraith S. A., Son T. C., Zeng H. (2001). Semantic Web Services, *IEEE Intelligent Systems*, vol. 16, no. 2, pp 46-53.
2. Fensel D., Brussler C. (2002). The Web Service Modeling Framework WSMF, *Electronic Commerce Research and Applications*, vol. 1, no. 2, pp 113 – 137.
3. RIME, D., 2003. New Electronic Trading Systems in Foreign Exchange Markets. *New Economy Handbook*
4. "Front Arena", 2008. http://www.sungard.com/FrontArena (2008-09-12).
5. Exjobb: Arena Integration Framework Web Service Provider, July 2008, Oscar Rotel.
6. Design Patterns, 2nd edition, (December 2004) Gamma, E., Helm, R., Johnson, R., Vlissides, J., Addison-Wesley Publishing Company.
7. FAME Software Development Kit, Document: FCA 1801-2A (internal), March 2003, Toll, Johan., SunGard Front Arena.
8. "FIX.4.2 Message – Order Single", 2008, http://www.fixprotocol.org/FIXimate3.0/?sel_language=en&sel_version=FIX.4.2 (2008-01-02).
9. XML Specification Guide, 1st edition, (1999), Quin, G., John Wiley & Sons, USA.
10. "Introduction to SOAP". http://www.w3schools.com/soap/soap_intro.asp (2008-01-12).
11. "The Apache Software Foundation". July 21st 2008 http://ws.apache.org/axis2/c/index.html (2008-09-17).
12. "Frequently asked questions", 2008, http://www.apache.org/foundation/faq.html#name (2008-12-01).
13. "Axis 2/c VC project". October 4th 2007 http://wsaxc.blogspot.com/2007/10/axis2c-vc-project.html (2008-09-23).
14. "Developers quick start guide for Axis2/c – Windows". June 16th 2008. http://wsaxc.blogspot.com/2008/06/developers-quick-start-guide-for-axis2c.html (2008-09-23).
15. "About the Apache HTTP Server", 2008, http://httpd.apache.org/ABOUT_APACHE.html (2008-12-03).
16. "Internet Information service 7.0" , 2008, http://www.microsoft.com/windowsserver2008/en/us/internet-information-services.aspx , (2008-12-03).
17. Exposing FRONT ARENA functionality as dynamic Web Services, 2008-07-30 (internal), Roos, Daniel., SunGard Front Arena.

18. Kamburugamuva, Supun., Sharma, Indra., “Axis2/c debug mode problem”, October 3rd 2008, http://www.mail-archive.com/axis-c-dev@ws.apache.org/msg15787.html.
19. “Running samples”, July 21st 2008, http://ws.apache.org/axis2/c/docs/installationguide.html#2 (2008-12-04).

8 Appendix

Appendix 1 | implementation of Adapter methods; open, close and send.

```
Wrapper *WSAdapter::open(AdapterListener& al, const Factory& fac)
{

      SendToStatusMgr("Open", "Opening WSAdapter");
      m_factory = (Factory*)&fac;
      m_listener = &al;
      bool test = true;
      DBConnection *providerDbCon = m_conFactory-
>getDBConnection("ProviderDB");
      RegistryDBObject *parametersDbObj = (RegistryDBObject*
)providerDbCon->getObject(NULL, "Parameters", IS_CHAR);
      char *sz_axisRepositoryPath =
FAME::ReadFromDBChar(parametersDbObj,"AxisRepositoryPath");
      char *sz_axisPort =
FAME::ReadFromDBChar(parametersDbObj,"AxisPort");

      m_wThread = new WrapperThread(fac,
al,createWrpCB,sz_axisPort,sz_axisRepositoryPath);    //creates thread
and wrapper
      m_wThread->start();
      while(test)       //makes sure that the thread continues when the
server has stopped
      {
            if(m_wThread->checkServerInit() == 1)
            {
                  printf("Server start Success");
                  test=false;
            }
            if(m_wThread->checkServerInit() == -1)
            {
                  printf("Server start failed");
                  test=false;
            }
      }

      return NULL;
}
```

```
void WSAdapter::close()
{
      m_wThread->stop(); //closes thread
}


Wrapper *WSAdapter::send(Wrapper *wrp, void *user_arg)
{
      //gets SynchId
      const Factory &f = wrp->getFactory();
      const WrapperDescriptor &wd = f.getWrapperDescriptor(wrp-
>name());
      const Field &field = wd.getField("SynchId");
      const void *pField = wrp->extract(field.getFid(),
field.getIndices());
      int WSPId =
SyntaxConverter::toInt(pField,IS_CHAR,field.getLength());

      requestTable->findWrp(WSPId, wrp); // hittar wrp i listan och
lägger till wrp

    return NULL;
}
```

Appendix 2 | implementation of Adapter callback function (createWrpCB) .

```
typedef const char ** returnType;
typedef returnType (* CB)(char*, char *[]);


static returnType createWrpCB(char *opName, char *param[]) //first
array value (array[0]) tells how many parameters they array has.

{
      WSAdapter *wsAdpt = WSAdapter::Instance();
      returnType svar = wsAdpt->WrpCb(opName,param);

      return svar;
}

returnType WSAdapter::WrpCb(char *opName, char *param[])
{

      const WrapperDescriptor &wd = m_factory-
>getWrapperDescriptor(opName);
      Wrapper *wrp = m_factory->createWrapper(wd.id());
      char *num = param[0];
      string digit(num);
      int count = atoi(digit.c_str());
      bool generateWrp = 0;
      int size = 7;
      returnType reply = (returnType)malloc(sizeof(const char *)*size);
      static const char *s_reply = "Reply";
      char *replyN = (char
*)malloc(sizeof(char)*(strlen(opName)+strlen(s_reply)+1));
      strcpy(replyN, opName);
      strcat(replyN, "Reply");
      bool SId=false;
      string errorMessage;
      vector<const char *> ClientReply;

      if(count>0)
      {
            bool eMess = requestTable->fill(count, wrp, wd, param,
errorMessage, SId);

            if(eMess)
            {
                  std::cout<<errorMessage;
                  generateWrp = 1;
            }

      }
      if(generateWrp==0 && !SId) //if everything succeded, all wrapper
fields exist
      {
                  printWrp(wrp);
                  try{
                        m_listener->eventUpdate(wrp,NULL);
                  }
                  catch(...){
                        std::cout<<"EventUpdateError:Wrapper not sent";
                  }
      }
      else if(generateWrp ==1)
      {
                  printf("No wrapper has been sent");
      }

      else if(generateWrp==0 && SId)
      {
                  Wrapper *wrpReply = requestTable->wait(m_SynchId,
m_listener, wrp); //waits for reply (wrp)
                  printWrp(wrpReply);
                  if (wrpReply)
                  {
                        //transforms data from wrp -> vector -> array
                        const Factory &fReply = wrpReply->getFactory();
                        const WrapperDescriptor &wdReply =
fReply.getWrapperDescriptor(wrpReply->name());
                        FieldIterator *FIter = wdReply.getIterator();
                        char *fieldName = NULL;
                        for(FIter->First(); !FIter->IsDone(); FIter-
>getNext())
                        {
                              const Field *fieldR = FIter-
>CurrentItem();
                              const void *pfieldR = wrpReply-
>extract(fieldR->getFid(), fieldR->getIndices());
                              if(fieldR)
                              {
                                    char *value =
SyntaxConverter::toChar(pfieldR,fieldR->getType(), fieldR-
>getLength());
                                    if (value[0] != '\0')
                                    {
```

```
                                                fieldName = (char *) fieldR-
>getName();
                                                if(fieldName !=
std::string("SynchId"))
                                                {
      ClientReply.push_back(fieldName);
                                                      char *v =
strdup(value);
      ClientReply.push_back(v);
                                                }
                                          }
                                          free(value);
                                    }

                              }
                              delete FIter;
                              returnType arrayReply =(returnType)malloc
                              (sizeof(const char *)*((int)
                              ClientReply.size()+2));
                              size= (int)ClientReply.size();
                              char countR[10];
                              _itoa(size,countR,10);
                              arrayReply[0] = _strdup(countR);
                              for (int q = 1; q<=size; q++)
                              {
                                          arrayReply[q] = ClientReply[q-1];
                              }
                              return arrayReply;
                        }
                        else
                        {
                              returnType arrayReply =
                              (returnType)malloc(sizeof(const char *));
                              char *ReplyMessage = (char *)
                              malloc(sizeof(const char *)*(1+1));
                              char *ReplyM = "No reply was found for
                              Wrapper";
                              strcpy(ReplyMessage, ReplyM);
                              arrayReply[0]=ReplyMessage;
                              return arrayReply;
                        }
      }
      else
      {
            reply[0]="6";
            reply[1]="operation";
            reply[2]=replyN;
            reply[3]="resultCode";
            reply[5]="errorText";

            if(generateWrp==0)
            {
```

```
                        reply[4]="0";
                        reply[6]="0";
                }
                else{
                        reply[4]="1";
                }
                return reply;
        }
        return NULL;
}
```