Quality of Service - FFI
IT2901 - Group 7

Bremnes, Jan A. S.
Johanessen, Stig Tore
Kirø, Magnus L.
Nordmoen, Jørgen H.
Støvneng, Ola Martin T.
Tørresen, Håvard

Supervisor:
Duc, Anh Nguyen

FFI represented by:
Johnsen, Frank Trethan
Bloebaum, Trude Hafsøe

April 26, 2012

Abstract

TODO ...

Contents

List of Figures

List of Tables

1 Project Introduction

This is the final report documenting our progress in the course IT2901 - Informatics Project II. It will give a description of the problem we were presented with, how we have planned the work for the project, how we have organized our group and what development methodology we have chosen etc. We will also give a brief discussion about why we have made the decisions we have.

1.1 Project Background

> *Essential to Network Based Defence (NBD) is the concept of end-to-end QoS, which in turn requires employing cross-layer QoS signaling. This means that QoS must be considered at all layers of the OSI model, and that QoS information must traverse these layers...* - Motivation[1]

This was the introduction we got for our motivation to work on this project. In many ways it illustrates the background of the project and why the customer wants us to work on it. As the customer is working with wireless networks with very low bandwidth they need to be able to control the flow of messages. The reason for these strict requirements is the command hierarchy and the risks involved which means that some messages in the network are more important than others. To be able to separate those messages there needs to be a collaboration between all the applications and libraries used. Currently there is no or little support for this cooperation on the application level which is what we were tasked to do. For us this assignment will be a challenge not just because it is some what uncharted waters, but also because of the strict requirement to prioritize.

1.2 Customer

Our customer are two senior researchers at the Norwegian Defence Research Establishment(FFI) [2] working in the SOA division. They had this to say about FFI.

[1] Please see (G) - Quality of Service (QoS) support for Web services in military networks
[2] Forsvarets Forskningsinstitutt (FFI) - [http://www.ffi.no/]

The Norwegian Defence Research Establishment (FFI) is the prime institution responsible for defence-related research in Norway. The establishment is the chief adviser on defence-related science and technology to the Ministry of Defence and the Norwegian Armed Forces' military organization. FFI addresses a broad spectrum of research topics ranging from the assistance of operational units to the support of national security policy via defence planning and technology studies. FFI also has a research unit in Horten focusing on maritime research.

1.3 Course

In this course, students will work in groups to carry out a software project. The department will present a list of available projects. Students are required to work on their project and to attend common activities and supervision meetings. The results from each phase must be clearly documented in the mid-term and final report. [3]

This course is considered to be the Bachelors project, the main project that finishes a bachelor degree. The focus of the course is to give customer interaction and experience in larger development projects with extensive parts of planning and documentation.

1.4 Students

The student education is common among the group members. We're all studying informatics on our third year of our bachelors degree. Most of us started studying straight out of high school and have little or no relevant skills besides the university courses. Although some members have done some work outside of university which counts very positive.

Among the experience in the group some people have experience with Git. We have one person which have done several large scale project which will help us immensely. Some of the people have some experience with NS3 which could come in handy. And everyone have extensive experience with Java.

1.5 Supervisor

The institute(IDI) have assigned us a supervisor. The supervisors role is to give guidance to the group related to matters of group dynamics and project management. The supervisor should also assist in the process of solving conflicts in the group, if any. The supervisor would also step in as a mediator if we would have experienced any problems with the customer which we could not resolve on our own. The supervisor have also given us feedback on our report and progress throughout the project. This has been valuable feedback that we have use to improve our report.

We have had biweekly meetings with the supervisor throughout the project. Every week we sent our weekly report and activity plan to the supervisor to inform him of our progress.

[3] Course description fetched from the course pages at NTNU.no, 29.03.12 - [http://www.ntnu.edu/studies/courses/IT2901]

1.6 Document Structure

This document is structured in a fashion to show you, the reader, our process throughout this project. It should give you a view into each part of the project from the early weeks when we were struggling to understand the initial design right up until our final moments with testing.

We begin with describing the task and the requirements in *Task Description and Requirements*. This is the details of the task we were given by the customer, and the high level requirements that we, together with the customer, agreed upon.

Prestudy continues with our initial thoughts around the project. It includes the architecture we envisioned for the client and the server early on in the project and it will give you some insight into what we initially thought and will serve as a good comparison with our end result.

Next is the chapters about *Project Management* and *Development Methodology*, these two chapters combined should give an impression of our thoughts and plans for collaborations to reach our end goals. It should also give you an idea about how all of our appended documents, such as Activity Plans, have played a role during the course of the project. After reading theses sections, the team structure and the distribution of responsibilities in the group, should be explained.

The chapter about *Design and Implementation* will focus on the design of our two pieces of software in some detail and should give you a good idea about the overall architecture of the system that we started out to design. After that follows a section which should give the necessary details about our code and how the system works deeper down. Last in this chapter is a section about what has changed from the original design.

The next chapter outlines our *Testing*. Here we will explain how to setup the testing suite and will detail our test. These tests will be connected with the requirements and it should detail why we wanted to run each test. The ability to reproduce our whole testing setup should also give you the confidence that we have in our results. We will end this chapter with a look at our results and some thoughts about how these findings relate to our initial problem.

Finally we will wrap up with *Conclusion* and *Project Evaluation* which together will wrap up the report. This will look at the future work for this research and give you our final thoughts about the course, project and process.

After the main part of the report we have some appendixes that elaborate the work process and results of the project. These are:

[TODO: write about the appendixes that will be included in our report.]

At the far end of the report we have added some attachments. These are not directly linked to our report. But they are our notes, plans, and summaries that have been created throughout the project.

[TODO: make sure the right appendixes are included at the end of the report. - weekly reports - schedule - meeting summaries. - activity plans - gantt chart - risk list. - ...]

2 Task Description and Requirements

Our task is to provide a Quality of Service[4](QoS) layer to web services for use in military tactical networks. These networks tend to have severely limited bandwidth, and our QoS-layer must therefore priorities between different messages, of varying importance, that clients and services want to send. Our software will have to recognize the role of clients, and, together with the service they are trying to communicate with, decide the priority of the message.

2.1 Description

Our assignment is to create a Java application which will function as a middleware[5] layer between Web Services[6], and clients trying to use these services. The middleware needs to process SOAP[7] messages, which is the communication protocol for most web services, in order to be able to do its task. On the server side, the middleware needs to process messages and understand SAML[8] in order to deduce the role of the client. This role, together with information about the service the client is trying to communicate with, decides the overall quality of service the messages should receive.

Our software needs to be able to modify the TOS/DiffServ packet header[9] in order for the tactical router[10] to prioritize correctly. Currently NATO has just defined one class, BULK, which is to be used with web services. It is defined in the STANAG 4406 as Military message Handling system. This standard may change in the future and our middleware should handle these upcoming changes gracefully.

In addition to this, the middleware needs to be able to retrieve the available bandwidth[11] in the network, which in the real system will be retrieved from the tactical routers. In our testing this information will come from a dummy layer, but how this information is obtained should also be very modular, so that the customer can change how the bandwidth information is obtained later.

With all this information, the role of the client, the relationship between the client and the service, and the available bandwidth, our middleware layer should be able to prioritize messages. Our product should, as much as possible, use existing web standards, the customer outlined some of their choices and options

[4]Quality of Service refers to several related aspects of telephony and computer networks that allow the transport of traffic with special requirements.[http://en.Wikipedia.org/wiki/Quality_of_service]

[5]In the report middleware will refer to the program we are making. Other distinctions should be made explicitly in the text.

[6]Web Services - A software system designed to support interoperable machine-to-machine interaction over a network.[http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#soapmessage]

[7]SOAP - A lightweight protocol intended for exchanging structured information in the implementation of web services in computer networks.[http://www.w3.org/TR/soap12-part1/#intro]

[8]SAML - Security Assertion Markup Language.[https://secure.wikimedia.org/wikipedia/en/wiki/SAML]

[9]TOS - Type of Service, a field in the IPv4 header, now obsolete and replaced by DiffServ.[http://en.wikipedia.org/wiki/Type_of_Service]

[10]Tactical router - A Multi-topology router used in military networks

[11]Bandwidth - Available or consumed data communication resources.[https://secure.wikimedia.org/wikipedia/en/wiki/Bandwidth_(computing)]

we have for implementation, like SAML, XACML[12], WS-Security[13] and WSO2 ESB[14]. In addition to this, our middleware needs to work with GlassFish[15], as that is the application server the customer uses.

2.2 Requirements

As the customer wanted all documentation written in English, we decided to use this for all written communication and documentation, in order to keep things consistent.

The way the course is structured in terms of deliveries of reports and documentation also creates a fairly natural implicit sprint period to work off of, and using an agile methodology will help in easily producing and maintaining said reports and documentation. In addition to the reports and documentation, we will try to deliver a prototype to the customer before the final delivery in May.

The customer does not require any prototypes along the way, just a working piece of software by the end of the project, so the deadline we have set for the prototype is self-imposed.

The customer has not given us many strict requirements, but instead they have suggested a few things that we could do. Given this freedom, we decided that we should improve on the base requirements by adding most of the things mentioned in this section.

The following is a list of technology requirements. We have a scale from 1 to 4 where we rate the importance of our requirements. 1 is the most important while 4 is the least important. There are requirements that share a priority as they are equally important to the completion of the project.

ID	1
Name	Written in java
Priority	1
Purpose	Java is chosen to ensure that the code can be reused, that it is easily readable for others, and that it is OS independent.
Constraints, assumptions, dependencies	The Java JVM and skills in java programming.
Functional	Working on all platforms that support java. Not OS dependent.
Non-Functional	Ensure good code quality and code conventions
Design constraints	Because we chose to work with WSO2 ESB we decided that we would just use Java version 6. This is because the ESB is hardcoded to use Java version 6, we felt that this was not a big hindrance

⋮

[12] eXtensible Access Control Markup Language. [https://secure.wikimedia.org/wikipedia/en/wiki/Xacml]

[13] WS-Security - An extension to SOAP to apply security to web services

[14] WSO2 ESB - An Enterprise Service Bus built on top of Apache Synapse. [http://wso2.com/products/enterprise-service-bus/]

[15] Application server written in Java. [http://glassfish.java.net/]

ID	2
Name	Message prioritizing
Priority	1
Purpose	Differentiate the messages being sent and make sure that high priority messages is sent before low priority messages.
Constraints, assumptions, dependencies	-
Functional	High priority messages must arrive, even at the cost of dropping lower priority messages.
Non-Functional	-
Design constraints	-

⋮

ID	3
Name	Standards
Priority	1
Purpose	Use standards where they can be used
Constraints, assumptions, dependencies	-
Functional	SAML, Diffserv
Non-Functional	Use web standards were we can and it makes sense
Design constraints	-

⋮

ID	4
Name	Testing
Priority	2
Purpose	Use NS3[16] for testing.
Constraints, assumptions, dependencies	We will be limited in the types of network we can create. Since this is also not real world testing we can only say something about a best case scenario in the simulation.
Functional	The testing framework should be working and we should have test results from it.
Non-Functional	We used unit tests while coding to make sure that the code worked correctly.
Design constraints	The tests have to be designed with the functionality in mind, not the existing code.

⋮

ID	5
Name	Documentation
Priority	2
Purpose	To have extensive documentation on every part of our project. This will ensure that anyone can replicate our results later. This is also important to the customers as they want to replicate our results to see if this type of QoS could be used in an actual network.
Constraints, assumptions, dependencies	-
Functional	The documentation should be so extensive and thoroughly written that anyone can replicate our results. And the use of our library should be documented to help anyone wanting to use it.
Non-Functional	All documentation shall be in English and be written to the best of our abilities to ensure good quality.
Design constraints	There are some constraints that were set by the institute. These constraint dictates sections that has to be present in the report.

⋮

ID	6
Name	Use metadata to determine priority
Priority	3
Purpose	The purpose of this requirement is that our software should use metadata to determine the priority of clients. As the server side has to tell clients which priority they get they have to use metadata to inform the clients.
Constraints, assumptions, dependencies	Since we have to support SOAP messages we are limited in they ways we can express this metadata.
Functional	The metadata has to be presented in a way that a client using SOAP can interpret.
Non-Functional	-
Design constraints	-

⋮

ID	7
Name	GlassFish
Priority	2
Purpose	Make it easy to use Web Services in a production environment.
Constraints, assumptions, dependencies	This puts some constraints on the type of services we can deploy.
Functional	GlassFish must be supported as the application server.
Non-Functional	-
Design constraints	-

:

ID	8
Name	Set package priority
Priority	2
Purpose	Currently there is only one priority class defined by NATO, the BULK class, but this will most likely change in the future, as such our middleware layer needs to be expandable enough to handle this change in the future.
Constraints, assumptions, dependencies	Since we are using Java we are constrained to IPv4 as Java does not support setting the Type of Service field on IPv6[17].
Functional	Must be able to set priority on network layer packets. There must also be an easy way to configure this priority so that future NATO DiffServ classes will be supported.
Non-Functional	-
Design constraints	-

:

ID	9
Name	Network Resources
Priority	3
Purpose	Minimize the usage of network resources. Use the given resources the best way possible.
Constraints, assumptions, dependencies	Since we are to use as little network resources as possible we have some rather large constraints on the messages we can exchange. This would imply among other things that the metadata we want to exchange can not be sent as separate messages, but should be piggybacked on other messages.
Functional	Use as little network resources as possible.
Non-Functional	-
Design constraints	-

:

ID	10
Name	Resource usage
Priority	4
Purpose	Minimize overhead and runtime. The faster it goes, the better. The less resources it uses the better.
Constraints, assumptions, dependencies	-
Functional	The customer has only said that we can expect the product to be used on a standard laptop with full Java support. This means that as long as the program runs on our laptops we should be good to go resource wise.
Non-Functional	-
Design constraints	-

3 Prestudy

This project is one that requires quite a lot of prestudy before we can begin coding or even designing the architecture. Since the customer wanted us to implement existing technologies, such as Glassfish, WSO2, SAML etc. we needed to spend some time researching those technologies to figure out what to use, and how to use it. The following sections will describe the the overall architecture of how we imagined our system to be like.

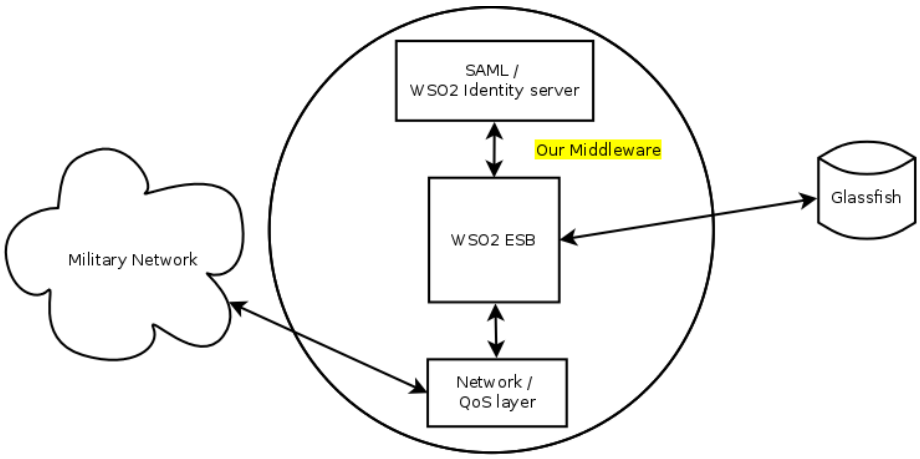


Figure 1: Basic server architecture
This shows our initial thoughts of how the servers side architecture wold look like. This has been changed later in the project.

3.1 Server side Architecture

The server side architecture consists of several components, the WSO2 ESB, the WSO2 Identity Server, the Tactical Router and the GlassFish server. Most of them are visible in the initial architecture shown in figure 1. All of these components are already available, so what we will have to make is mediators[18] in the ESB.

Before the client can request a web service it has to have an identification. To get an ID-token it has to contact the Identity Server using the ESB as a proxy[19] (Fig.2-1). Then the client can request a web service from the ESB. Several things will then happen in the ESB. First the request message is sent to the SAML mediator (Fig.2-2), this mediator contacts the Identity Server to validate the clients ID-token (Fig.2-3). If the token is validated and the client is supposed to have access to the requested service, the message is passed on to the GlassFish proxies (Fig.2-4), otherwise it is dropped. The ESB acting as a proxy will then send the request along to the requested service on the GlassFish server (Fig.2-5).

[18]Mediator - A component in WSO2 ESB which can be used to work on incoming or outgoing messages that passes through the ESB
[19]Proxy - A proxy server acts as an intermediary between clients and servers

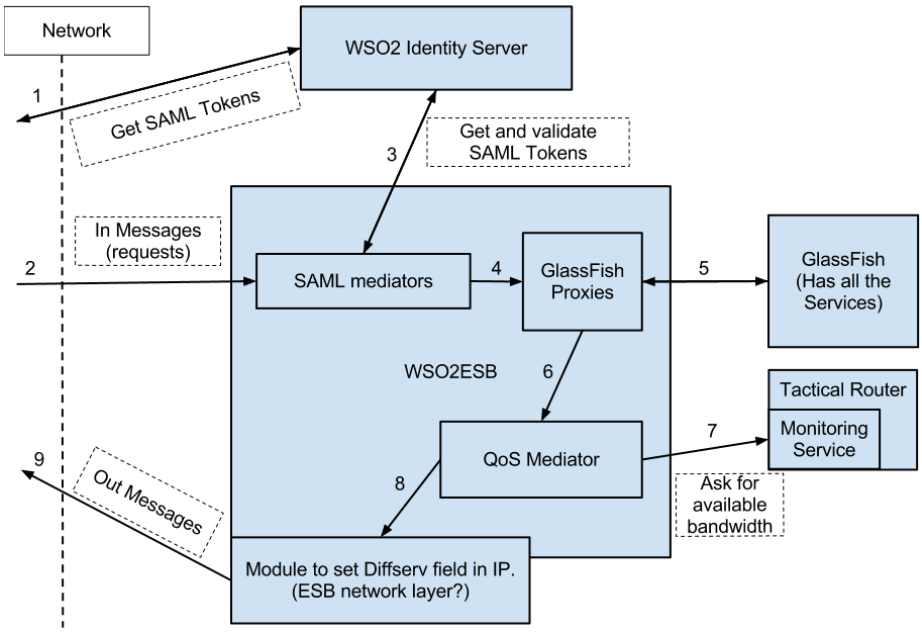


Figure 2: The Server side Architecture
This is the overall design of our implementation of the server side. It shows the modules in the server and the flow in the system.

When the request is received at the service, it will probably start sending some data to the client. This is also done through the ESB. First the message is sent to the QoS mediator (Fig.2-6). This mediator will first look at the role, or identity, of the client and the service requested, and use this information to assign a priority to the connection. Then the Monitoring Service[20] on the Tactical Router is contacted for bandwidth information (Fig.2-7), which is used together with the priority to determine whether the message should be sent right away or held back until some higher priority message is finished sending.

Either in the QoS mediator, in the ESB's network layer, or after that, the Diffserv (ToS) field of the IP header will have to be set (Fig.2-8) before the message is sent to the client (Fig.2-9). This field is used by the routers in the network to prioritize packet sending. This step is quite important to the whole procedure as this is one of the few requirements the customer has given us, as such this step can not be dropped from the final product.

3.2 Client side Architecture

The client-side architecture will be composed of altered (already existing) client software, the OpenSAML[21] library as well as our client library implementation.

[20]Monitoring Service, a service that provides bandwidth monitoring, running on the same server as the Tactical Router

[21]OpenSAML - A set of open source C++ and Java libraries to support developers working with SAML. [https://wiki.shibboleth.net/confluence/display/OpenSAML/Home/]

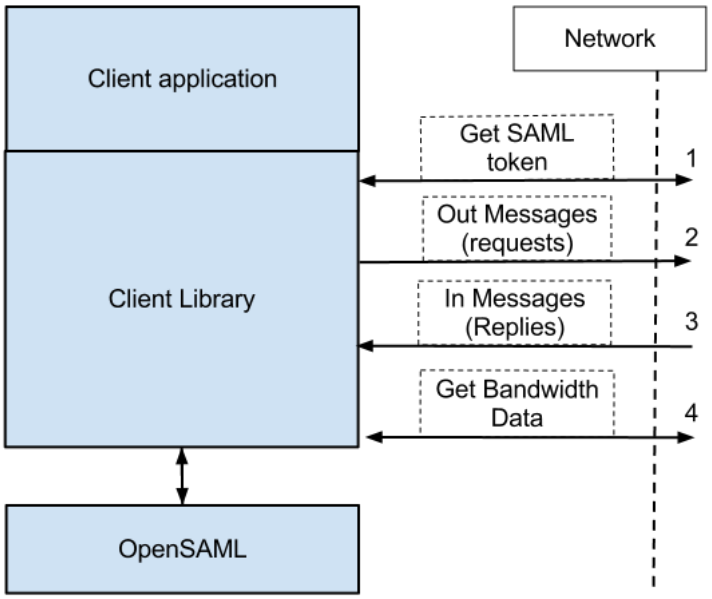


Figure 3: The Client side Architecture
The client architecture shows the basic thought of how the system should look like and the communication to and from the client library.

Before the client library can ask for the data the client needs to get a SAML authentication token from the identity server (Fig.3-1). The communication here will most likely be handled by our library, but the SAML packages will be created and analyzed by the OpenSAML library.

The client library then sends the request from the client to the server (Fig.3-2), appending the SAML token to the package as well as adding some metadata in the SOAP header related to the client role and setting the TOS field of the package to a default value.

The reply from the server is examined by our client library for the metadata the server has embedded in the SOAP header, relevant metadata is stored for future communication and the package is passed to the client application (Fig.3-3).

When new communication is initiated after this first connection is made the client should, if everything went as expected, have the necessary information to prioritize new messages. This means that the client can now take an informed decision about how it should prioritize messages, but in order to do this to the best of it's abilities it also has to take into consideration available bandwidth (Fig.3-4).

3.3 Unit testing

We decided quite early on that we wanted to do unit testing of every piece of code that we would produce, i.e. test driven development. The reason behind this choice is that we think it will result in better code quality. An added bonus

is the simplification of integration testing, due to easier discovery of whether a new code addition will integrate with the old code. Also writing the tests first lets us concentrate more on exactly what the methods should do, instead of the content and how it should do it. One of the problems with test driven development however, is the possible bias that could occur, we could end up only satisfying the test and not the actual requirements. This could be countered to some extent by writing more comprehensive tests. Another positive point in favour of unit testing is the requirement we have, which states that the product has to be written in Java where such test are easy to integrate and write using JUnit.

3.4 Integration testing

For integration testing, we decided that we wanted to do automated system testing every other week in collaboration with code reviews. The procedure we are going to follow will be coding new features in a separate branch. Once every other week the finished branches will have all their unit tests run thoroughly, followed by a code review of at least one person. Then if the automated system tests are fully operational, they will also be run to look for additional errors which the unit tests can not pick up. This point is likely to change in the future as a two week time interval might be to long given the short implementation period. The advantage of doing this integration testing is better overall code quality, since we will test code before it is used by other parts of the system. Since we are also doing code reviews, people will also gain experience with other parts of the system which they previously had not worked on. This will benefit everyone since knowledge about the system is shared, and it will help in the eventuality of someone getting sick. The advantage of developing in separate branches is the reduced risk of polluting code other people are working on, and a better separation of stable and unstable code.

3.5 System testing

When it comes to system testing, the customer was quite insistent that we test the product thoroughly in an emulated network situation. Since we have had some experience with NS3 we decided that we wanted to do the system testing on it. The advantage with this, is that the customer has already set up some testing scenarios and helper-scripts designed for NS3, which they offered for our use. This will greatly reduce the time needed for setting up the test suite, and it will also give us the ability to have automated tests, which we don't have to monitor or interact with. Another added advantage is easy testing, as we only have to start a script in order to run the whole suite, but that comes at the cost of actually setting up the whole thing. As of the midterm report, we have set quite a lot of time aside in order for us to implement the proper NS3 support we need. To monitor what is happening during the test-runs, our applications will output all important information regarding what is going on, in addition to this we will have a packet sniffer on each end which will capture network traffic. Using this information we should be able to tell a whole lot about what is going on in the network and we should be able to decide whether we have met the requirements or not.

Below are some of the detailed test cases which we want to automate on top of NS3. The testing itself will be automated, but in order to get some result from the tests, some human interaction is needed to interpret the output data.

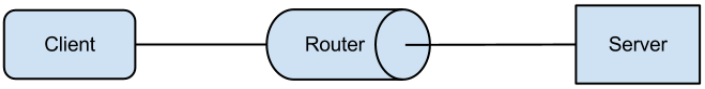


Figure 4: Simple message sending
One client communicates with one server through one router.

Simple message sending:
In this test, shown in figure 4, we want to test the ability of the client and the ESB to communicate. We want to see that the client is able to send messages(message) to the server and get a response back. For monitoring purposes, this test will rely on both applications to log their behavior. In order for us to give this test the green light we must see a message going out from the client then passing through the ESB to GlassFish. Then finally a reply should be sent back from GlassFish to the ESB and then to the client.

Setting Quality of Service:
In this test, we want to test the client and the ESB's ability to set the DiffServ field in the IP header. The first requirement is that the test "Simple message sending" has been passed. For this test to be considered a success, the client has to send a message to the ESB, which is responds with the DiffServ value in the SOAP header. The ESB must at this point have set the DiffServ value in the IP header. The client should then use a service on GlassFish, but this time the IP header must contain the correct DiffServ value. In order to monitor this test, only a packet sniffer located on the client and ESB side needs to be used. The packets must be examined, and the correct DiffServ value must be present in the IP header of all packets, except the first one going out from the client.

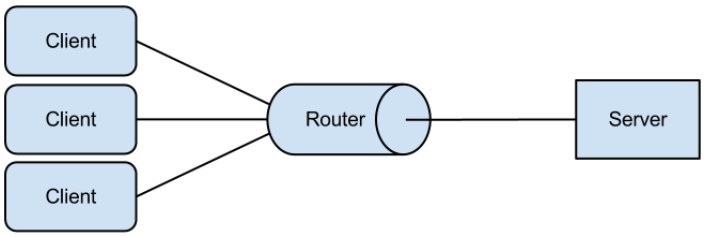


Figure 5: Three clients message sending
Three clients communicate with the same server through the same router. One of the clients will have a higher priority than the two others, in order to test the servers ability to prioritize.

Prioritizing messages:
In this test we want to test the ESB's ability to prioritize messages. The scenario

will be set up as shown in figure 5, with two clients sending lots of messages in an attempt to flood the capacity of the network. All of these messages should have the same priority, but intermittently, a third client with a higher priority will attempt to send some messages. What we are looking for is that these higher priority messages should be sent out from the ESB before the ones with lower priority and, if necessary, it has to stop some already sending messages. For this test to be successful, we must see some lower priority messages being preempted or held back. To do this, the log file of the ESB must be studied, and there should be some clear indications of one of the requirements.

Changing DiffServ value:
In this test we want to check the ESB's ability to change the DiffServ value after a reconfiguration. The test and the result can be performed and examined the same way as "Setting Quality of Service", but this time the test has to be run twice. One where the configuration has one DiffServ value, and a second run where the DiffServ value has a different value. For the test to be successful, one would have to examine the resulting pcap[22] files on the server side, and check each run to see if the two tries have different DiffServ values.

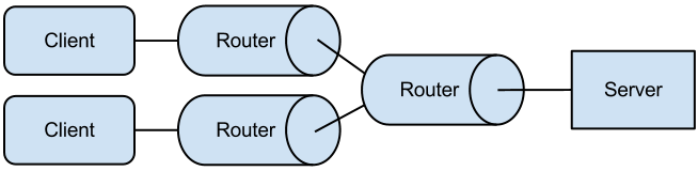


Figure 6: Two Clients with different paths
Two clients communicating with the same server, but with different paths.

Multipath server routing:
In this test we want to look at the ESB's capabilities to talk to the MS and understand the routing result. From the MS the ESB should get some routing information about the topology of the network. As you can see in figure 6, if the link between the server and the first router is not the limiting factor, the two clients should not get in each others way. Therefore, since we get the information about the last router from the MS, the ESB should understand that there is likely no problem and should not preempt any messages. To check if this is actually the case, the ESB will need some time to adjust as it does not get the full picture of the network topology, but after this time, no messages should be dropped from the ESB's side.

Competing clients in a multipath environment:
This test is a compilation of the tests "Multipath server routing" and "Prioritizing messages". For this test we want to make sure that the ESB is smart enough to only preempt the messages going to one of the competing clients. As you can see in figure 7, there is one client which should not affect either of the two others if the link between the server and the first router is not a bottleneck.

[22]Pcap is short for Packet capture, which in our text usually refers to a program which captures the traffic on a given socket

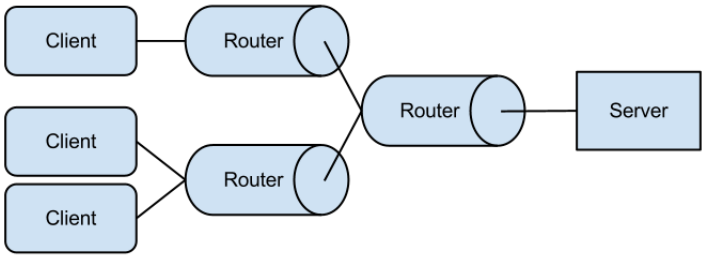


Figure 7: Tree Clients where two are competing
Three clients communicate with the same server, but only two of them share the path.

This should allow this client to receive messages even though the two other clients are competing for scarce resources. To check that this test is successful, a combination of the clients log files and the server log files will have to be used. If most(over 96%) of the messages arrive at the higher priority client and the third client is not affected then this test is successful.

Competing clients in a low bandwidth scenario:
In this test we want to test that the ESB can manage to prioritize messages in a network with a joint bottleneck, but with different endpoints. In figure 7, if the link between the server and the first router is the bottleneck, the ESB should after a small initialization understand that it has to preempt messages going out to all clients, in order to let a higher priority client get the service it is supposed to get. The scenario will be set up in such a way that one of the two competing clients will have higher priority than the two others, the two lower priority clients should then send a steady stream of messages, which should fill the bottleneck link. The third client should then start sending some messages which must now fill the entire bottleneck link, and create a situation where the ESB has to hold back or preempt messages going to either of the two other clients. As before, a combination of the ESB and the clients log files have to be examined.

3.6 Alternative solutions

The customer also gave us a paper[1] which described a previous project they had worked on which tried to solve something quite similar to what we were tasked with. The paper described a system which were used in conjunction with Tactical routers to retrieve bandwidth information and to control sending of messages into this network. As the customer explained this work was not something we could directly copy as the project had not used a lot of web standards and had focused more on the tactical routers as opposed to web services. What we could take out of it however was how they throttled messages. The paper contained five methods which we could easily implement and use their result as an indication of what methods we should use to throttle or hold back messages.

One architecture, which our customer suggested for the project, was to have a proxy in between nodes and creating a custom QoS layer which would sit in

front of both the client and the services. This layer would then communicate with a SAML server for authentication, and would have to do all the message prioritization based on the same criteria as our architecture. There are several points about this architecture which would make it a good fit for us. Since the QoS layer would be identical on both client and server side it would mean less work, and more code that could be shared among components, but this freedom comes with some downsides. The first and most glaring problem encountered would be that services on the server would have to be altered to be able to communicate with this front end. Even though we were free to choose architecture ourselves, the client expressed a wish that we would not choose this model because the customer wants to use COTS[23] services which would not be compatible with the new front end.

Even though the above mentioned architecture is not the best fit for us we wanted to take some aspects of the architecture further. Since clients can easily be altered, the above mentioned solution is not applicable for server side, the solution could however be used for the clients. Having a proxy on the client side could be quite good, but because of the work involved and probable time constraints we chose not to go with this solution. On the server side however a front end is not the best solution for us. What we instead are looking into, is to use an ESB which would be configured together with the services and work as a proxy. Because many ESBs have integrated SAML processing we could easily take advantage of such facilities along with custom message processing, with which we would then extend the ESB to support our needs. The clients would have to point to the ESB, but this should both be trivial to do and the customer has expressed their agreement that this is satisfactory. We could eventually expand the functionality with service discovery, which then would be a good solution to the problem.

So far we have outlined major alternative architectures which could be alternatives to our project, but there are also alternatives within our proposed solution. One such alternative is not to use a premade ESB, but rather build one ourselves. This solution was thoroughly investigated, but was eventually turned down because of the massive amount of work that had to be done, the quality of an already made ESB is much higher than we could ever achieve during this project, and lastly, the open source tools available to implement the functionality needed for SAML was not very well documented, and would take considerable time to get familiar with.

On the client side we also have the choice of having either a HTTP[24] proxy or writing our own custom library. Both have some advantages and disadvantages, a proxy would be better for integration with client programs, but creating this proxy or configuring and customizing an already existing solution is not trivial. On the same note, creating a library for use in client programs is easier, but this would mean that client programs would need to be altered to be usable with our middleware, which isn't that desirable. We chose to go down the road of least resistance, as we see it currently we would have do quite a lot of research into proxies which could in the worst case scenario result in just wasted time as far as our product goes. A client library would from our perspective be easier as we would have more control, the overall design should be easier and we know that

[23]COTS - Commercially available Off-The-Shelf
[24]Hypertext Transfer Protocol

with this sort of library we can integrate OpenSAML which is a huge advantage.

3.7 Process model

Before we started this project we were quite unsure how we were going to work on the project we had heard lots about Scrum, but few of us had ever used it in a project of this scale. For us this meant that we had lots of options, but we did not know a lot about them.

After we talked to the customer about what the task was, and understood what we were going to do, we decided that we wanted to a Waterfall model of software development Waterfall model[25]. Because our assignment is quite research focused we need to start with a planning stage in order to completely comprehend the task ahead and how we are going to solve it. This method of working lends itself very well to the Waterfall model, but we feel that it would not work so great with Scrum or a more agile method. Secondly the customer does not have the resources to meet with us every week to have scrum meetings and be a part of our team. This does not make Scrum completely impossible, but it would be harder than the Waterfall model with little extra reward. Our choice of process model is described in section 5.2.

To practice some agile development we decided that we want to do the implementation phase as agile as possible. This would mean weekly sprints with code review and rapid delivery. This would be something that we could do after the planning phase is done to hopefully produce higher quality code. The reason behind this decision was our interest in Scrum or agile development and the thought that weekly sprints with code review will improve the overall code quality and help us keep our eyes on the prize.

3.8 Tools

We had no clear outline of what tools we were going to use in the prestudy phase. The tools we ended up using is described in *System Technology 5.3*

4 Project Management

In this section, we'll take look at how we organized the team, a brief risk assessment, and an evaluation of the work process.

4.1 Team Organization

This section describes in detail how we organize ourselves and how we split roles and tasks among the team members. We have a flat team structure[26] and have shifted our focus accordingly over to team communication.

[25] Waterfall model - A sequential design process. [http://en.wikipedia.org/wiki/Waterfall_development]

[26] Flat organization structure is a structure with few or no levels of intervening management. The idea is that well-trained workers will be more productive when they are directly involved with decision making. [http://en.wikipedia.org/wiki/Flat_organization]

4.1.1 Team Structure

We already know each other coming into the project so we have chosen a flat organizational structure (Fig:8), with no intervening levels of management, since all decisions within the team will more or less be made by all the members together either way. Relying on the entire group for decisions will both involve and invest everyone in the project and will work well with our already existing group dynamic.

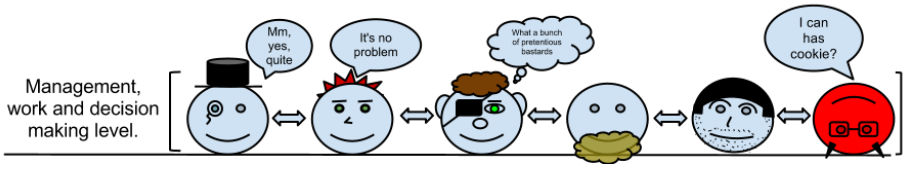


Figure 8: Team Organization chart
It was made during lunch, but the general principle still remains, that the structure is flat.

As the structure shows in the chart (Fig:8), there is no difference in what responsibility level anyone have, or what role one has. The concept of changing roles weekly is good for a learning situation, but very inefficient where knowledge and research are key components in a limited timed project. We anticipate that the time for this project probably won't be sufficient for any role changes, and therefore we have to keep people focused on the task they have been assigned. The efficiency of the current task relies on having the current research fresh in mind. If we were to change the roles every week, the newly assigned person would spend a lot of time getting up to date at the beginning of every week, which in turn wouldn't yield any measurable gains.

Rather than focus on responsibilities within the group, we've chosen to focus on tasks. The task will to some degree still represent areas of responsibilities, and since tasks will be spread across several group members, we don't run the risk of a single missing member crippling the entire group. Instead the remaining member(assigned) to a task will be able to pick of the slack. This, together with thorough documentation of a members knowledge, will just about eliminate the problems associated with an absent group member.

Further, the team structure and the distribution of responsibility gives us the chance to define how we want to deal with task and their priority. The work flow that we have makes us prioritize tasks continuously and get the most pressing task done at the correct time. It's similar to a max heap. We put tasks in to the heap, heapify(prioritise tasks) and choose(pop) the maximized task, the task that has the highest priority.

When we choose a task we consider the person's interest, experience and existing knowledge. Most times the tasks fall naturally to one person that has worked with similar tasks earlier in the project. Other times there is more of lottery, where the task has no prerequisites. Often we rely on a person's initiative to take a task or we easily delegate them with a question, "Can someone do that?". Task delegation and sharing the work load has not been a problem so far in the project.

4.1.2 Team communication

We decided that we will work together from 10 to 16, Monday through Thursday every week, with allowed exceptions for lectures and such. Group members can also work in their free time to make up for missed collaboration hours or to just put in some extra work. This means more work than the course requires, but we decided that we want to do it this way so we can either take some time off now and then, or have more time for the exams in May.

We will not be able to have frequent face to face meetings with the customer, because the customer is located in Oslo. We decided to have weekly meetings using Skype instead, as well as e-mail communication as needed. Since we have seen what happens in projects where there is little to no communication, we decided, in agreement with the costumer, that we at least wanted to have weekly meetings in order to keep a good dialog with the customer, and also give them the opportunity to take part in the development of the project.

4.1.3 Roles

[TODO: 4.1 Roles. Expand and explain our roles. Ola is the room contact, Jørgen knew FFI prior to the project. Comment on the fact that we did not change roles in the process(This has been moved to the project evaluation chapter!).]

Our team structure is discussed in *Team Structure* 4.1.1. It describes the general structure and the ideal situation and delegation. But to make this work in practice roles are unconsciously delegated to different people. A person then ends up with a role loosely based on the first delegation of a task. The first person to do that task has more experience then other people, so next time that task, or a similar one has to be delegated it is delegated to the same person as last time, since that will be more efficient.

Jan - IS, Jørgen - NS3, Ola - Server side, Magnus - Report, Stig Tore and Håvard - Client side.

4.2 Risk Assessment

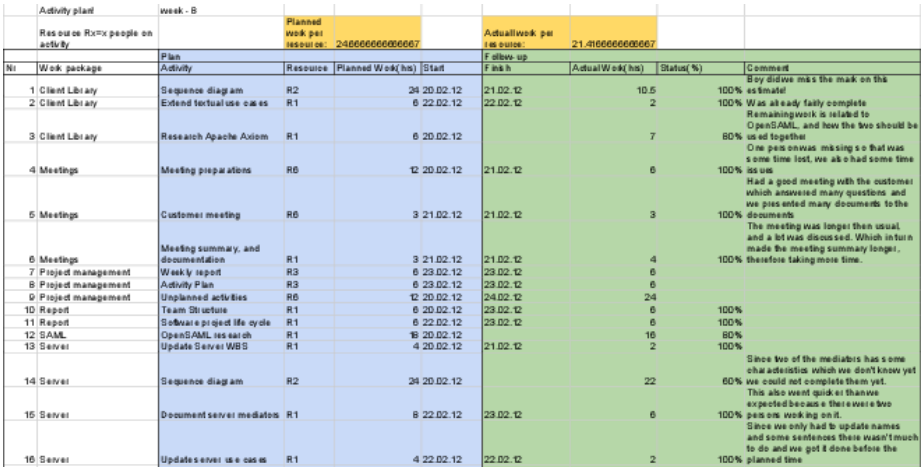
To help us avoid most problems we created a risk list which should contain most of the problems that we can encounter during this project. A risk list can never cover all the cases that could occur, but we do think that our risk list contains most of them. To cover the last cases which we might have overlooked we decided that we would add some *Risk Management* strategies to this section to explain how we will handle unforeseen risk as they appear.

To handle most of the risks that we have not written down in our risk list we will try to have a close dialog with the customer so that if an unforeseen risk should occur they will know about this. That way we can discus the problem together and come up with a solution that is satisfactory for everyone. Having this open dialog with the customer also ensures that they won't be surprised by any choices we make.

4.3 Progress tracking and Documentation

In the beginning we had a summary every day where we wrote what we were working on and what had to be done. We stopped doing this after we got good

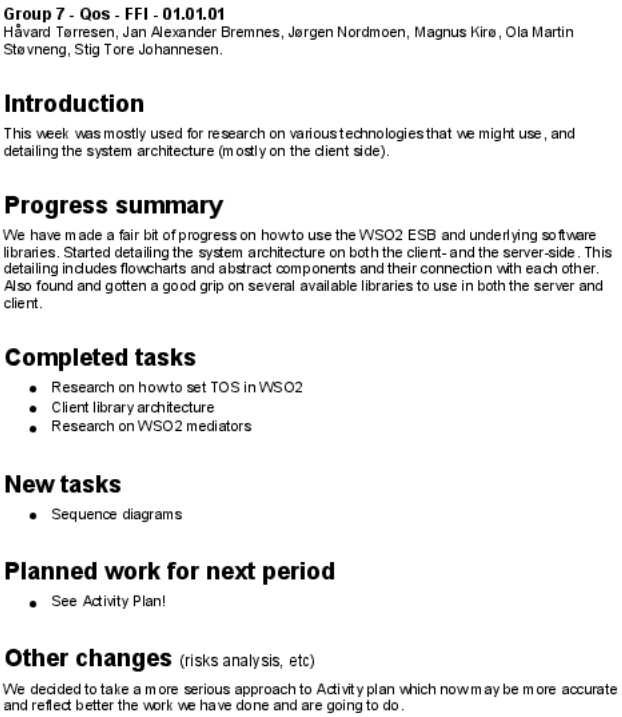
activity plans because the daily summaries became unnecessary.



	Activity plan	week - 8							
	Resource Rx=x people on activity		Planned work per resource:	24666666666667		Actualiwork per resource:	21,416666666667		
		Plan				Follow-up			
Nr	Work package	Activity	Resource	Planned Work(hrs)	Start	Finish	ActualWork(hrs)	Status(%)	Comment
1	Client Library	Sequence diagram	R2	24	20.02.12	21.02.12	10.5	100%	Boy did we miss the mark on this estimate!
2	Client Library	Extend textual use cases	R1	6	22.02.12	22.02.12	2	100%	Was already fairly complete
3	Client Library	Research Apache Axiom	R1	6	20.02.12		7	80%	Remainingwork is related to OpenSAML, and how the two should be used together
4	Meetings	Meeting preparations	R6	12	20.02.12	21.02.12	6	100%	One person was missing so that was some time lost, we also had some time issues
5	Meetings	Customer meeting	R6	3	21.02.12	21.02.12	3	100%	Had a good meeting with the customer which answered many questions and we presented many documents to the documents
6	Meetings	Meeting summary, and documentation	R1	3	21.02.12	21.02.12	4	100%	The meeting was longer then usual, and a lot was discussed. Which in turn made the meeting summary longer, therefore taking more time.
7	Project management	Weekly report	R3	6	23.02.12	23.02.12	6		
8	Project management	Activity Plan	R3	6	23.02.12	23.02.12	6		
9	Project management	Unplanned activities	R6	12	20.02.12	24.02.12	24		
10	Report	Team Structure	R1	6	20.02.12	23.02.12	6	100%	
11	Report	Software project life cycle	R1	6	22.02.12	23.02.12	6	100%	
12	SAML	OpenSAML research	R1	18	20.02.12		16	80%	
13	Server	Update Server WBS	R1	4	20.02.12	21.02.12	2	100%	
14	Server	Sequence diagram	R2	24	20.02.12		22	60%	Since two of the mediators has some characteristics which we don't know yet we could not complete them yet.
15	Server	Document server mediators	R1	8	22.02.12	23.02.12	6	100%	This also went quicker than we expected because there were two persons working on it.
16	Server	Update server use cases	R1	4	22.02.12	22.02.12	2	100%	Since we only had to update names and some sentences there wasn't much to do and we got it done before the planned time

Figure 9: Example Activity plan
This figure displays the structure of our activity plans. It's meant as an overview. See the appendix for full weekly reports. H.3

The activity plans(Fig:9) now have the role of our day to day summaries and work progress. We update the activity plan as we go along. This way we have a complete overview of tasks and work hours that are planned this week. As we update the activity plan we have an overview of the work done this week and where we have missed with our time estimation.



Group 7 - Qos - FFI - 01.01.01
Håvard Tørresen, Jan Alexander Bremnes, Jørgen Nordmoen, Magnus Kirø, Ola Martin Støvneng, Stig Tore Johannesen.

Introduction

This week was mostly used for research on various technologies that we might use, and detailing the system architecture (mostly on the client side).

Progress summary

We have made a fair bit of progress on howto use the WSO2 ESB and underlying software libraries. Started detailing the system architecture on both the client- and the server-side. This detailing includes flowcharts and abstract components and their connection with each other. Also found and gotten a good grip on several available libraries to use in both the server and client.

Completed tasks

- Research on howto set TOS in WSO2
- Client library architecture
- Research on WSO2 mediators

New tasks

- Sequence diagrams

Planned work for next period

- See Activity Plan!

Other changes (risks analysis, etc)

We decided to take a more serious approach to Activity plan which now may be more accurate and reflect better the work we have done and are going to do.

Figure 10: Status report example
This is an example of one of our status reports. We create them every week. All the weekly reports can be found in appendix H.2

As we can see in the weekly report (Fig:10) the status report has a standard setup. We created a template early in the project so that we could reuse it later and reduce work. In the process of creating the template we put some thought into it so that we would get a template that would work throughout the project without further changes. Despite the thought process of creating the templates, we had to make some small changes throughout the project.

4.4 Process Evaluation

[TODO: Write something about the fact that we use one day a week to work with planning and paperwork.]

Project management documented: Possible deviations and how they have been handled.

5 Development Methodology

We did not follow any established development methodology, such as Scrum[27] or XP[28], as this project required more planning and configuration of existing

[27]Scrum - An agile software development methodology. [http://en.wikipedia.org/wiki/Scrum_(development)]

[28]XP - A type of agile software development. [http://en.wikipedia.org/wiki/Extreme_programming_practices]

solutions, than actual coding. In the process of choosing a development methodology we considered scrum, waterfall, agile, xp, and some others in addition to combination of these. In the end we chose a mix of the Waterfall model[29] and Agile methods[30], we discuss these decisions in the sections below. You will also find a list of the tools we chose to work with, and why we decided to use them.

Because this is a research project, the customer will act more as an advisor than a customer, and will have more suggestions and advice than demands and requirements. We have been given a clear understanding of what the final product should be, and we have a list of requirements that should be met. Other than that, we are relatively free regarding how we go about solving the problem. Because of this, a single methodology, like Scrum, won't work for us, as it requires us to be in close and frequent contact with the customer, presenting a prototype every other week and continue development based on the customers feedback and demands.

As mentioned, this is a project that requires quite a lot of planning before any programming can be done. This necessitates that we start the development according to a waterfall model in terms of the architecture planning as well as the requirements specification. By using the waterfall model in these first phases, we ensure that the planning is done thoroughly to minimize the amount of trial and error during the later implementation phase.

As the project progresses we'll be switching to a more agile development method, in order to allow iterative development and facilitate any necessary changes that may turn up as code is produced, as opposed to waterfall-coding, where we have to strictly follow our plans. Agile also lets us use the flat organizational structure we have chosen, which we believe will greatly help cooperation within the team.

5.1 Project Organization

We have divided the project tasks in to work packages. These packages are represented in a WBS[31] (F). The schedule for the project is represented in a Gantt Chart[32] (Fig.11). The figure is part of our full Gantt chart. As the full diagram cannot be included nicely in the report we have attached it as an HTML document (G).

5.2 Software project life cycle

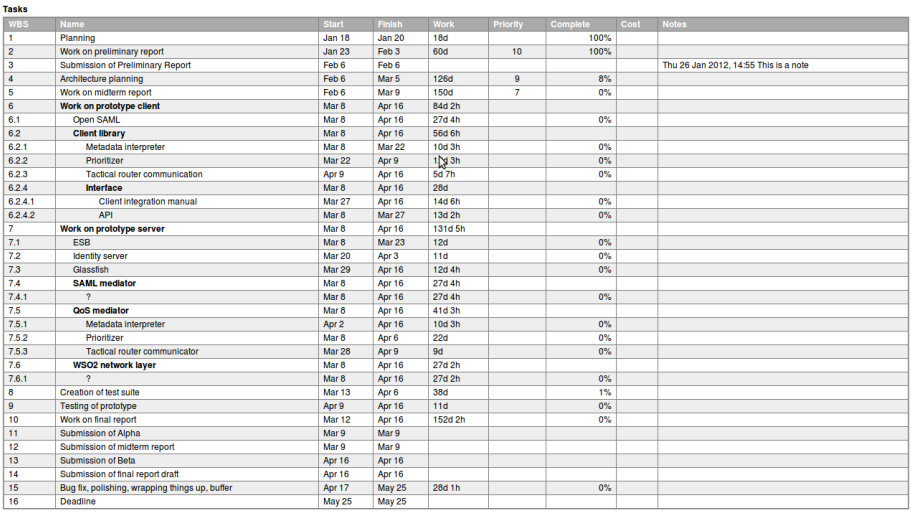
For our project life cycle we chose agile. Originally we started out with the intention of using Scrum and Scrum only. That idea was quickly scrapped as we found out that our task was very research heavy. This made us rethink our approach to the development cycle and turn in the direction of agile software development.

[29] Waterfall model - A sequential design process. [http://en.wikipedia.org/wiki/Waterfall_development]

[30] Agile methods - A group of software development methodologies based on iterative and incremental development. [http://en.wikipedia.org/wiki/Agile_software_development]

[31] WBS - An oriented decomposition of a project into smaller components. [http://en.wikipedia.org/wiki/Work_breakdown_structure]

[32] Gantt Chart - A type of bar chart that illustrates a project schedule. [http://en.wikipedia.org/wiki/Gantt]



Tasks

WBS	Name	Start	Finish	Work	Priority	Complete	Cost	Notes
1	Planning	Jan 18	Jan 20	18d		100%		
2	Work on preliminary report	Jan 23	Feb 3	60d	10	100%		
3	Submission of Preliminary Report	Feb 6	Feb 6					Thu 26 Jan 2012, 14:55 This is a note
4	Architecture planning	Feb 6	Mar 5	126d	9	8%		
5	Work on midterm report	Feb 6	Mar 9	150d	7	0%		
6	**Work on prototype client**	Mar 8	Apr 16	84d 2h				
6.1	Open SAML	Mar 8	Apr 16	27d 4h		0%		
6.2	**Client library**	Mar 8	Apr 16	56d 6h				
6.2.1	Metadata interpreter	Mar 8	Mar 22	10d 3h		0%		
6.2.2	Prioritizer	Mar 22	Apr 9	10d 3h		0%		
6.2.3	Tactical router communication	Apr 9	Apr 16	5d 7h		0%		
6.2.4	**Interface**	Mar 8	Apr 16	28d				
6.2.4.1	Client integration manual	Mar 27	Apr 16	14d 6h		0%		
6.2.4.2	API	Mar 8	Mar 27	13d 2h		0%		
7	**Work on prototype server**	Mar 8	Apr 16	131d 5h				
7.1	ESB	Mar 8	Mar 23	12d		0%		
7.2	Identity server	Mar 20	Apr 3	11d		0%		
7.3	Glassfish	Mar 29	Apr 16	12d 4h		0%		
7.4	**SAML mediator**	Mar 8	Apr 16	27d 4h				
7.4.1	?	Mar 8	Apr 16	27d 4h		0%		
7.5	**QoS mediator**	Mar 8	Apr 16	41d 3h				
7.5.1	Metadata interpreter	Apr 2	Apr 16	10d 3h		0%		
7.5.2	Prioritizer	Mar 8	Apr 6	22d		0%		
7.5.3	Tactical router communicator	Mar 28	Apr 9	9d		0%		
7.6	**WSO2 network layer**	Mar 8	Apr 16	27d 2h				
7.6.1	?	Mar 8	Apr 16	27d 2h		0%		
8	Creation of test suite	Mar 13	Apr 6	38d		1%		
9	Testing of prototype	Apr 9	Apr 16	11d		0%		
10	Work on final report	Mar 12	Apr 16	152d 2h		0%		
11	Submission of Alpha	Mar 9	Mar 9					
12	Submission of midterm report	Mar 9	Mar 9					
13	Submission of Beta	Apr 16	Apr 16					
14	Submission of final report draft	Apr 16	Apr 16					
15	Bug fix, polishing, wrapping things up, buffer	Apr 17	May 25	28d 1h		0%		
16	Deadline	May 25	May 25					

Figure 11: Part of our Gantt diagram
This is an example to show that the gantt diagram exists and what it looks like. Se full diagram in attachments: (G)

Early in the project we expected that we could begin coding and prototyping before too long. This proved to be wrong as there was a lot of research to be done. Scrum was originally a tactic to improve product flexibility and production speed. This works very well in software development when you already know what you are supposed to do and the major part of the task is to implement the required functionality. When the functionality has to be designed and researched extensively, scrum becomes unsuitable.

With the agile method there is elements that suits us better then others. "Individuals and Interaction" and "Customer Collaboration" are two important elements that we use. The full description of the agile method can be found in the Agile Manifesto[33].

Individuals and interactions are strongly connected with the organization of our team (4.1). The flat team structure force us to have a good dialog among the group members. This increase the team members interaction and strengthens the team communication. The strengthened communication promotes the individuals of the group and the team members confidence, which in turn increases the total productivity of the group. The frequent interactions with the customer are also a part of our adaption to the agile development method.

Customer Collaboration is the aspect of the group contacting the customer and keeping a good dialog with them. This is to make sure that we produce a product that the customer wants. To achieve this part of the agile manifesto we have meetings with the customer every week and have frequent email correspondence to iron out the bumps of our product. The frequent communication

[33]Agile Manifesto, the key elements of the agile software development method. [AgileManifesto.org]

with the customer helps us to create a more precise and consistent system with better documentation. The main part of the communication with the customer is for the benefit of the project and constant improvement. The constant improvement and iterative work flow is a central part of the agile method.

Waterfall vs Scrum, to be continued.

5.3 System Technology

We intend to do a test driven development in order to achieve high quality code. This will give us something to test while we are working, and it will also give us a great way to tell if some new piece of code gets in the way of previously written code. For this purpose we will use JUnit[34] as the testing framework. We will also be doing periodical code reviews approximately every two weeks of development, synchronized with a code/feature freeze where we make sure everything works. As the customer wanted extensive testing of the middleware, we agreed to do testing on the network emulator NS3, as we have someone in the group already familiar with it. The advantage of using NS3 will be extensive testing, but also a great deal of empirical and verifiable data, which the customer can also use to evaluate the product.

We will use Git[35] and GitHub[36] to handle our file repository, although Google Docs will be used for easy sharing and collaboration of schedules, meeting minutes, and reports. Even though the course set us up to use Subversion (SVN)[37], we decided against this as Git gives us more options to develop code which will not greatly affect other parts of the code base before we decide to integrate it. To this extent we have decided that we should take advantage of Git's built in support for 'branches' as much as possible. The argument for using Google Docs is that we have the possibility of editing a document together and easy sharing of documents. Delivered reports will be created with LaTeX[38], which we prefer over standard word processors. Our GitHub repository is open to the public, and the software will be released as open source, most likely under the Apache License.[39] In some places we are forced to use the ASF, like the changes we have done to Apache HTTPComponents and Apache Synapse, but as the licenses allows for derived works to be licensed under a different license we are free to chose a different one. The Apache License version 2.0 is also compatible with the GPLv3 so there is no problem for us in using either. We agreed to use the Apache2 license. FFI could not find any negativities for them in the license so we decided to use it.

Each of us is free to choose his own IDE[40] for programming. Because we are using Git, there should be no problem in using the IDE of our choice, and this gives us the added advantage that each person can use the tool which he is most comfortable with. We will stick to the standard Java Coding Conventions.

[34] JUnit - A testing framework for the Java programming language. [http://junit.org/]

[35] Git - A free and open source, distributed version control system. [http://www.git-scm.com]

[36] GitHub - A web-based hosting service for software development projects that use the Git version control system. [http://www.github.com]

[37] Subversion - A version control system

[38] LaTeX - A document preparation system for the TeX-typesetting program

[39] http://www.apache.org/licenses/LICENSE-2.0.html

[40] IDE - Integrated Development Environment

Since we were so free to choose which tools we wanted to use we decided that this list should be quite lightweight. However the list compiled should be an indication of what is needed for the project. Some of the tools were chosen by us as is and other were demanded by needs of other components. All tools used can be upgraded, downgraded or dropped during the course of this project. The final report will contain the official list as such this list is not in any way final. Our final report will also contain a list with supported tools tested with the final product.

- Git version 1.7.x
- Java version 1.6.x
- Free choice of IDE
- JUnit version 4.x
- NS3 version 3.13
- WSO2 ESB 4.0.3
- Axiom version 1.2.11 - Note that our server code must use the same version of axiom as the ESB.
- HTTPCore version 4.1.4
- Commons-logging version 1.1.1
- MobiEmu

  MobiEmu was installed with all dependencies, for more see https://code.google.com/p/mobiemu/ for all of MobiEmu requirements.

6 Design and Implementation

In this section we will discuss the design and implementation of the whole system. First of is the section about the client side of the equation and then follows the server side.

Below is a first taste of how our system works. In figure 12 you can see a cloud of clients which can be on the same local network or be on different networks. The *ESB* talks to *GlassFish* where services are placed and clients talk to the ESB which mediates the message into GlassFish. In order for the ESB to know how much bandwidth each client can utilize, it is dependent on the *Monitoring Service* which relays that sort of network information.

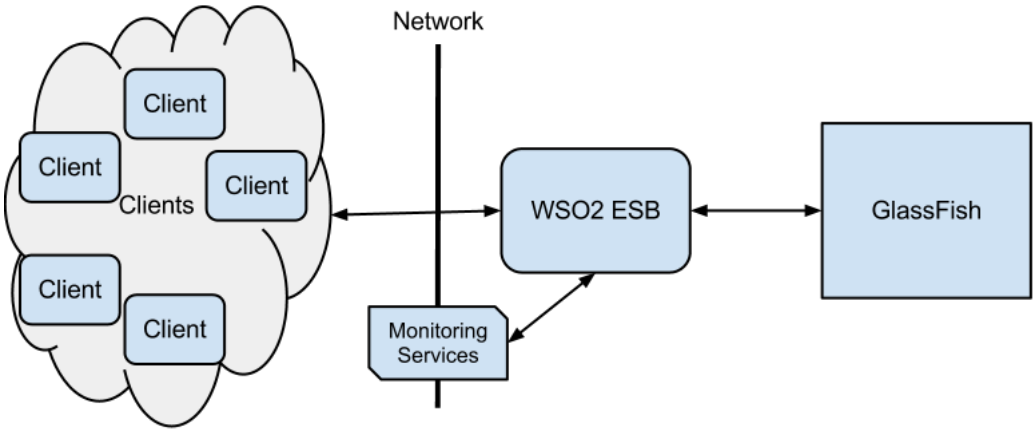


Figure 12: Total Overview
This figure shows roughly how our project looks from a bird's-eye view.

6.1 Client Side

This chapter will introduce the client side design, and architecture of our system. Section 6.1.1 will introduce the different components that make up the entire client, and includes a description of the different components that make up the client library. Next, section 6.1.4 will describe the use cases, and section 6.1.5 will take care of the data flow, followed by a detailed architecture description 6.1.6. Finally, section 6.1.7 will go through the sequence diagrams.

The class diagram, found in appendix G, is a useful addition when understanding the architecture of the client library and its functionality. The descriptions and diagrams in this section might become clearer when looking at the class diagram and see the connections between classes and the more specific contents of the classes.

6.1.1 Introduction

The client architecture consists of the following two components: The client application and the client library. Additionally, the library makes use of some external components to do some of its work.

6.1.2 Component description:

Client application:
The user-controlled applications that utilize web services. These must be modified to send all communication through the client library in order to get the priority it should get, and to interact properly with our server setup.

Client library:
This component will handle all communication with the service providers, as well as authenticating users and prioritizing their messages, based on who they are, and what their current role is. The authorization will involve a component

from the server side of the project, the identity server, which returns a token if the client is authorized. Client applications connect through a simple interface to provide credentials and data.

6.1.3 External libraries:

Axiom:
This component will be used to parse and manipulate XML[41] data in the form of SOAP and SAML. These are fairly extensive and complex data structures so an easy to use external library is essential here.

Apache HTTPComponents:
A lightweight component for easily setting up and using HTTP connections. While not strictly necessary this component will allow us to connect and communicate across networks far more easily than the standard java components.

6.1.4 Use Cases

Title: Accept client info
Actors: Client software, Client Library Interface
Main:

1. Client software connects to the library interface
2. Client delivers its credentials
3. Credentials are passed from the interface to the sequencer.
4. Credentials are checked by the sanity checker and passes.
5. Credentials are passed from the sequencer to the token manger.
6. Credentials are stored in the credential store.
7. Buffer, for previous tokens, is flushed

Extension:

4a. Credentials are clearly invalid

5a. Return error

Precondition: None
See: Requirement 6 (Section 2.2)

Title: Accept data to be sent
Actors: Client software, Client Library Interface
Main:

1. Client delivers data to be sent.

[41] XML - eXtensible Markup Language

2. Data is passed to the sequencer.
3. Sequencer creates DataObject and ReceiveObject.
4. ReceiveObject is returned to client.

Precondition: Client has established connection to the Library interface and it's credentials are accepted.
See: Accept client info

Title: Connect to server
Actors: Client Library, Server
Main:

1. Connection manager connects to the server
2. Set priority on socket based on SAML-token and related metadata

Extension:

1a. Unable to connect to server

2a. Return error

Precondition: DataObject has been created, and contains both bandwidth info and a token
See: Accept client credentials, Accept data to be sent and Fetch bandwidth info, requirement 8 (Section 2.2)

Title: Get SAML token
Actors: Client library, Server
Main:

1. Client library sends client credentials to server
2. Server verifies the credentials
3. Server returns SAML-token
4. Token is parsed into a token object
5. Token object is put into DataObject.

Extension:

2a. Client credentials not valid

3a. Server returns error

4a. Client library throws error

Precondition: Client has given library credentials and data to send, and a SAML token for the destination doesn't already exist. Connection to server has been established.
See: Accept client credentials, Accept data to be sent, Fetch bandwidth info and Connect to server, requirement 2, 3 and 6 (Section 2.2)

Title: Transaction towards server
Actors: Client lib, server, client
Main:

1. MessageHandler sends buffered data to server
2. Server returns reply to data.
3. The ReceiveObject in the DataObject gets the data from the server.
4. MessageHandler send data to sequencer.
5. Sequencer sends data to interface (QosClient)
6. Client fires a data received event to all listeners.

Extension:

2a. Server unavailable, reply doesn't arrive within timeout, etc.

3a. Throw error.

Precondition: Data to send exists, SAML token is in cache, connection to server active.
See: Accept client credentials, Accept data to be sent, Fetch bandwidth info, Connect to server and Get SAML token

6.1.5 Data Flow

Client credentials (Visualized in Fig.13)

1. The Client application sends credentials to the library interface
2. The interface passes the credentials on to the token manager, through the sequencer
3. The token manager stores the credentials in the credential storage
4. The token manager sends the credentials to the SAML communicator in order to fetch a token
5. The SAML communicator requests a token from the identity server
6. The identity server sends a token to the SAML communicator
7. The token is returned to the token manager
8. The token manager stores the token in the credential storage

Client Data (Visualized in Fig.14)

1. The client generates data and sends it to the library via API/Interface
2. The data moves on to the sequencer and down to the message hander
3. The message handler tells httpCore to establish a connection to the server, and the data is sent
4. The server sends a reply that is picked up by httpCore and forwarded to the messageHandler
5. The interface is notified of the reply
6. The interface notifies the client that the reply is ready

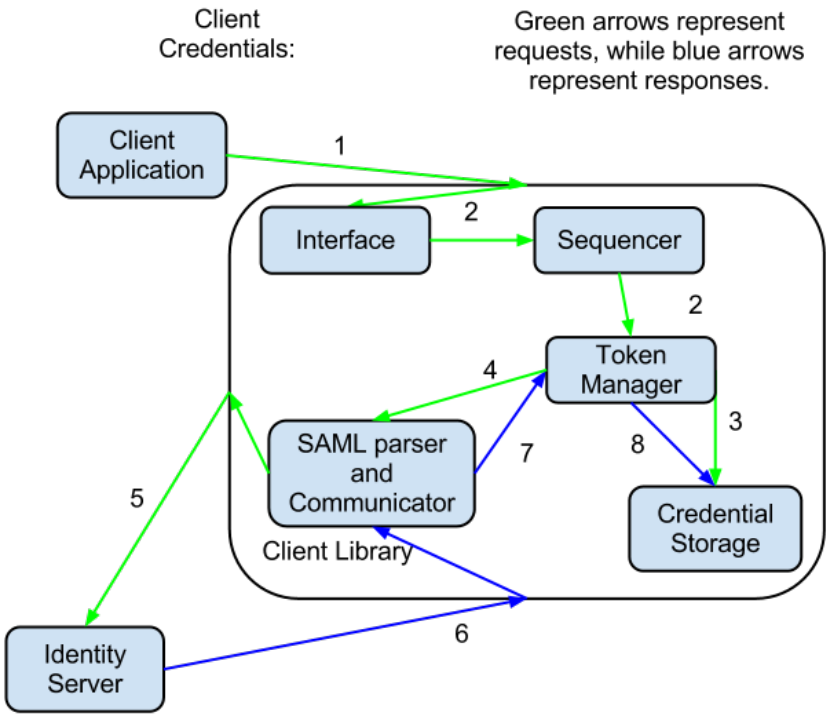


Figure 13: Client Credentials Flow
This figure describes how the client credentials are sent through the client library and through the system in general.

6.1.6 Architecture

All the following sub paragraphs in this subsection are parts of the client library which is shown in figure 15.

Interface
Known in the class diagram as "QosClient", responsible for providing a clean and easy to use interface for the clients.

Sequencer
The central piece of the client library. Responsible for keeping a record of all other modules in the system and communicate between them as well as making sure everything happens in the right order.

Sanity checker
This modules single purpose is to check the credentials provided by the client for sanity, in other words make sure they're conform to the rules for the credentials. It does however not verify that they are valid.

Token Manager

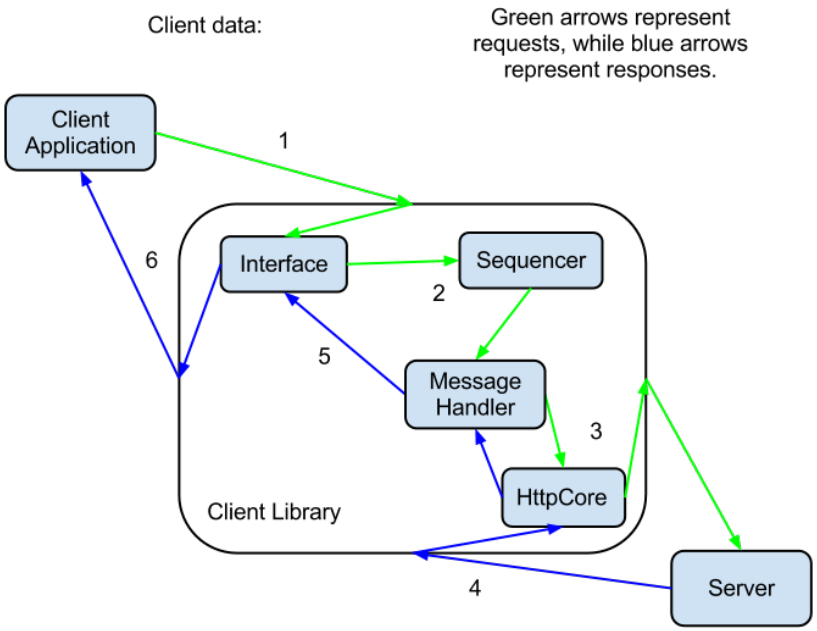


Figure 14: Client Data Flow
This is the message sequence. It describes the route the individual messages takes through the system.

This provides a nice and clean interface for the sequencer to store credentials and fetch tokens for data transmissions.

Saml Communicator
This module will take care of the communication between the client library and the identity server.

Saml Parser
This takes the reply from the identity server and parses it into a token object so that it can be easily used and stored.

Credential Storage
Responsible for storing token objects as well as user supplied credentials. Also makes sure that no token objects are returned if they are invalid or expired.

6.1.7 Sequence Diagrams

This section contains all the sequence diagrams for the client, with descriptions for each one.

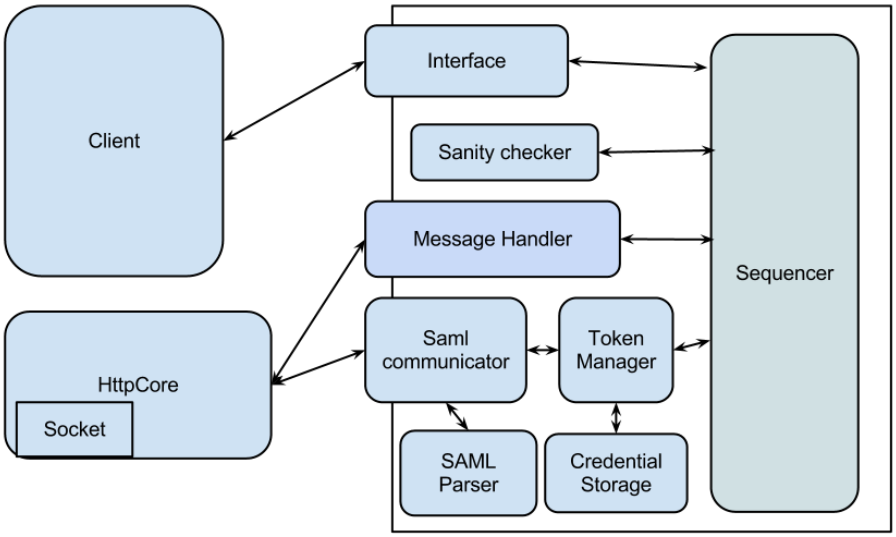


Figure 15: Detailed Client Architecture
This describes in detail the structure of the client library.

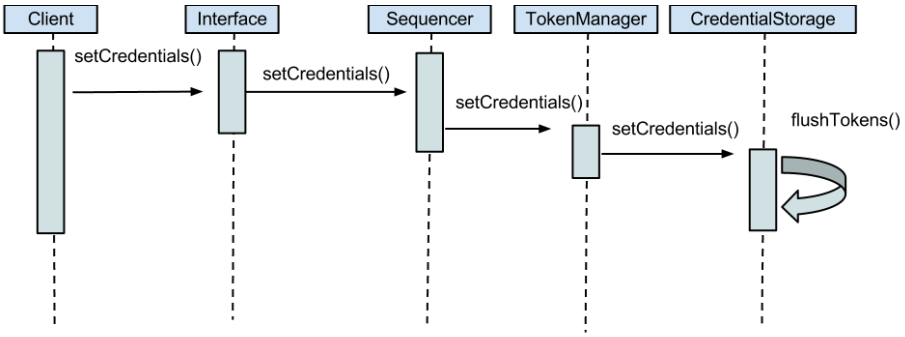


Figure 16: Accept client info
This describes how the client credentials are passed from the client, through the interface and into the credential storage.

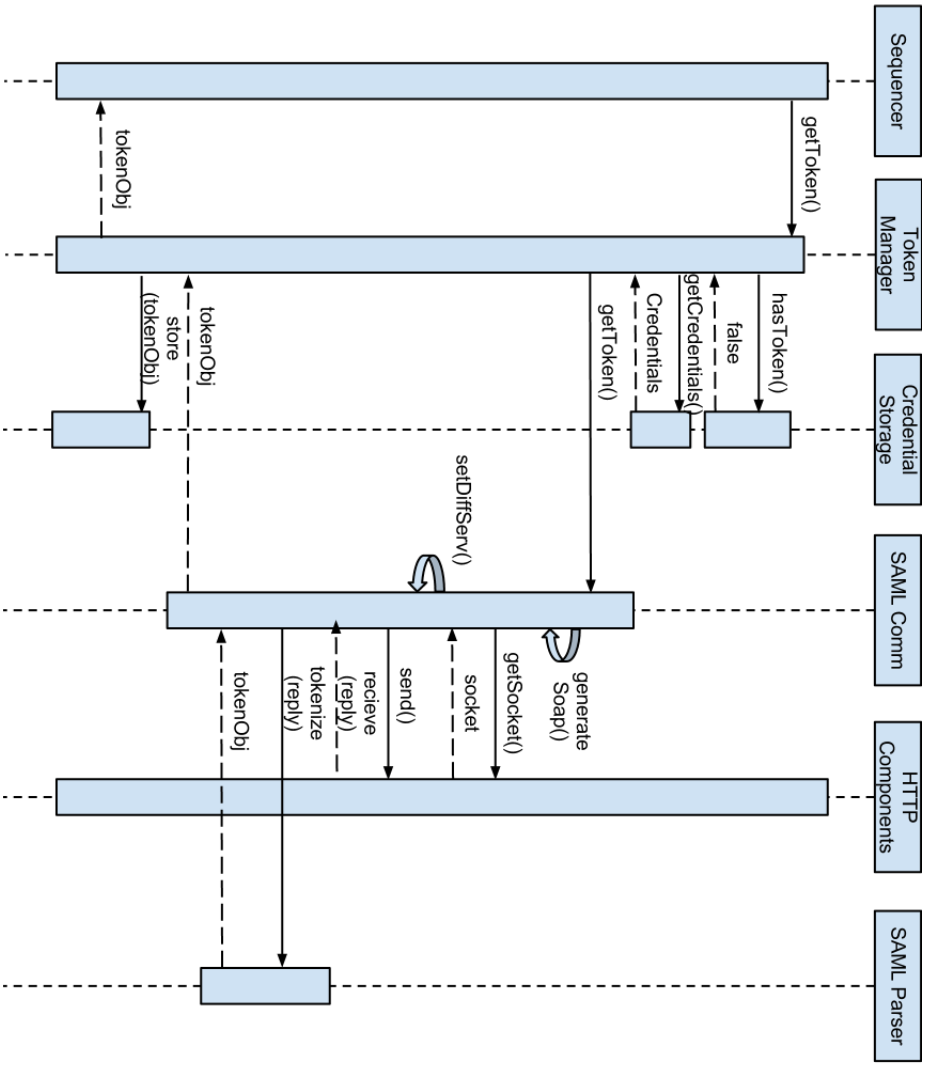


Figure 17: Getting non-stored token
This describes how the client library acquires a token from an external source, and stores it, when can't find the desired token in the storage, or if the token is invalid.

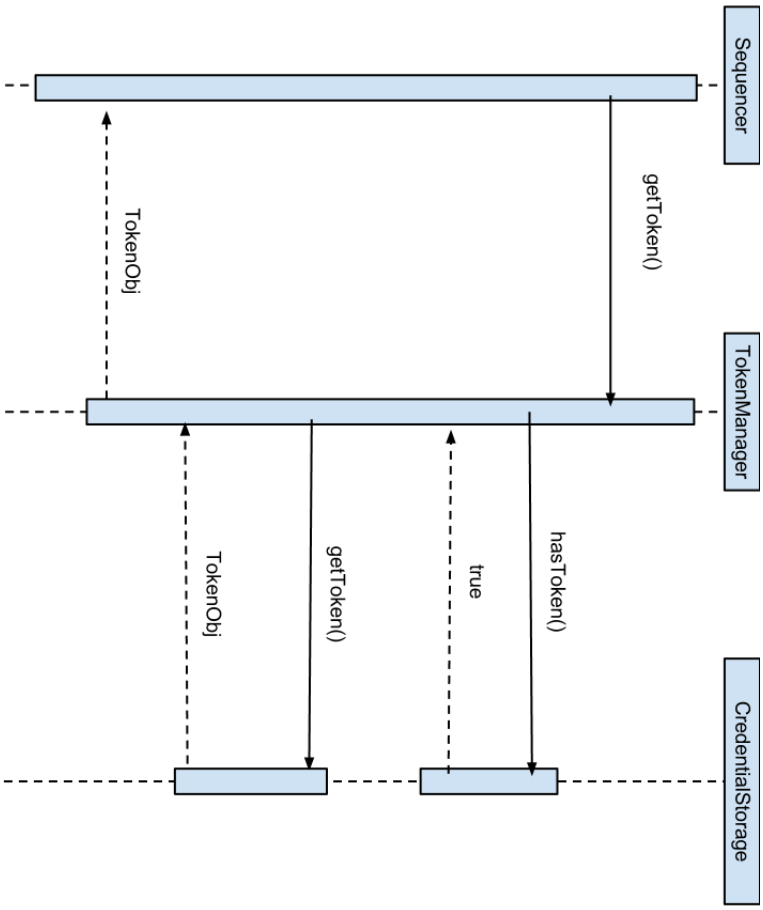


Figure 18: Getting stored token
When a token exists, we return that token instead of creating a new one.

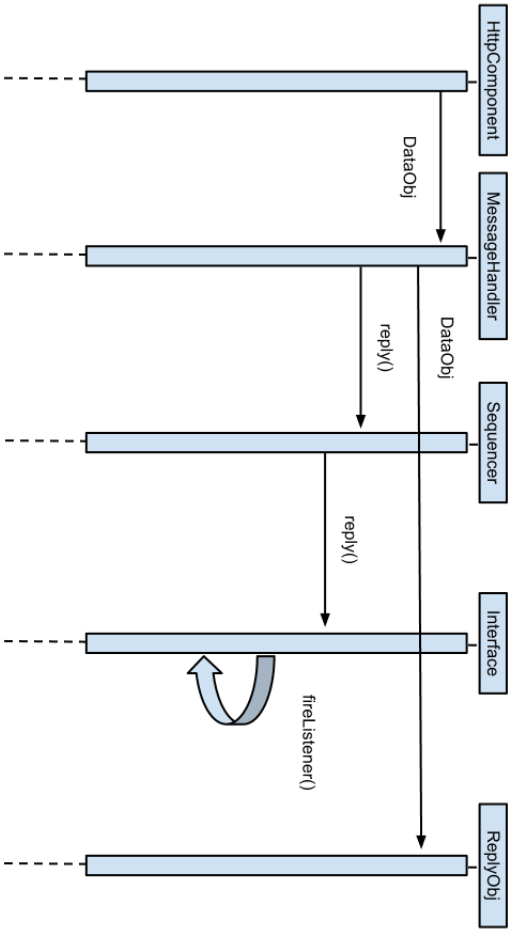


Figure 19: Receive reply
This diagram shows how the client library receives a reply and sends it to the client.

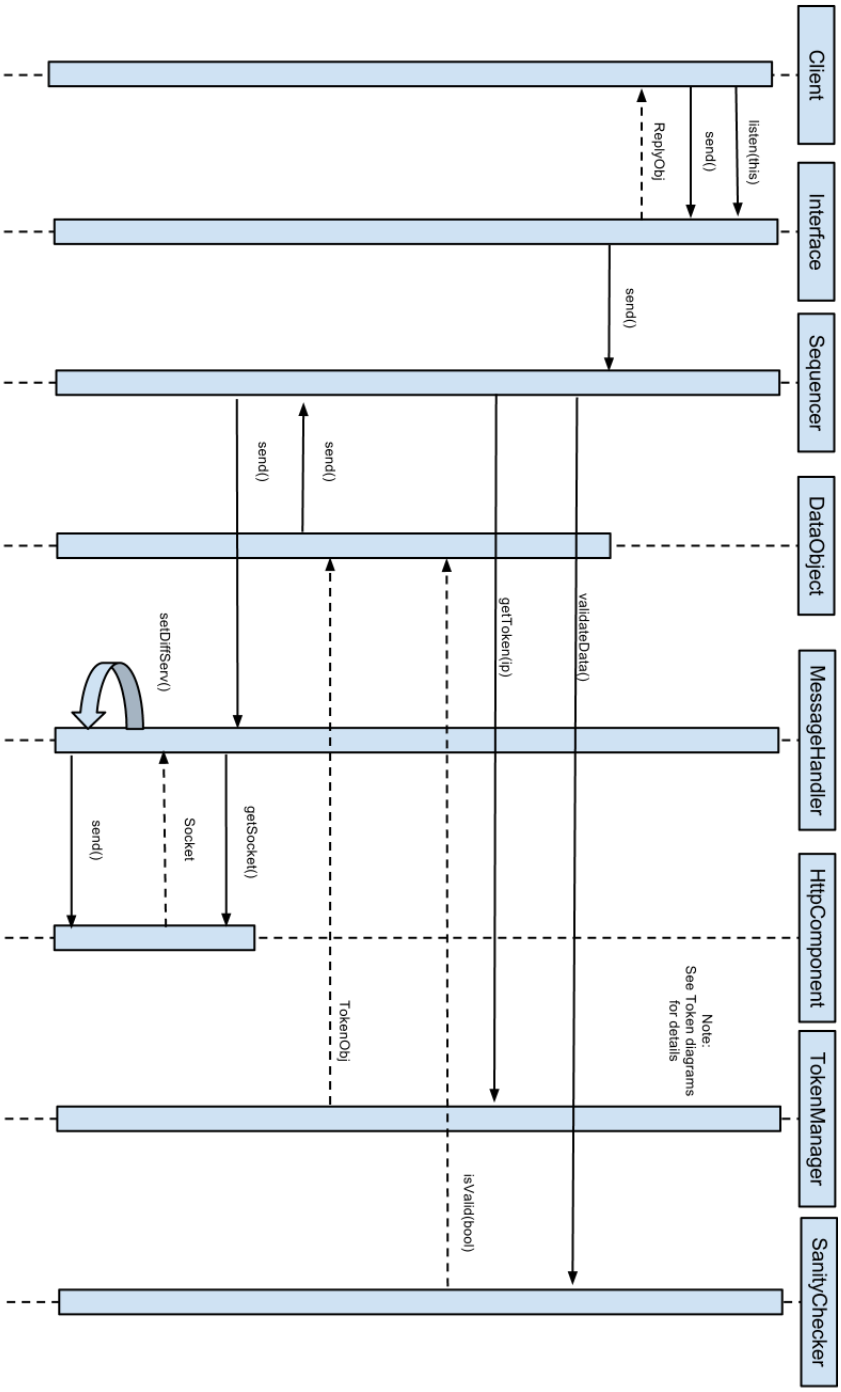


Figure 20: Send data
This diagram describes how data moves through the client library, receives a token, and eventually gets sent to the server.

6.2 Server Side

In this chapter we are going to introduce the design, implementation and configuration that we have done on the server side. In section 6.2.1 we will introduce the framework that we have built upon and what we are going to do with it. Next follows use cases, 6.2.2. Section 6.2.3 will go into more detail about what the framework consists of. The section will also guide you through the basic processing units which is used in the framework. The next section, 6.2.4, contains the dataflow through the server side, which will help you get a good overview of our design. 6.2.5 goes into detail in describing our custom components in the framework, and together with the dataflow should give you a good understanding of the whole server side. Together with section 6.2.6 you should get good system overview. Section 6.2.7 will give you the details about how we have configured the framework, it will not contain description of how we have set variables during testing, but using this description should make it possible to get the framework up and running.

6.2.1 Introduction

The server side architecture consists of several components, the WSO2 ESB, the Monitoring Service and the GlassFish server. The GlassFish server is not necessary to modify and the MS is something we must assume exist in the network. The ESB is what we have to modify, configure and extend to meet our requirements.

The ESB will be used to implement QoS for the web services. To do this, it will have to communicate with the MS, in addition to the clients and the services. The ESB must be configured to work as a proxy for the services on the GlassFish server. It will also be configured to use certain mediation sequences for incoming requests and outgoing responses. The extensions to the ESB consists mainly of custom mediators used in the mediation sequences. These mediators will have the tasks of determining priority of messages, contacting the MS for bandwidth data, and enforcing the priority. There will also be made modifications to one of ESB's library source code to allow setting the DiffServ field in the IP header.

6.2.2 Use Cases

This section will outline the use cases that we have thought of in relation to the server side. With the help of these you should get a rough idea of what we want the server side to be able to do.

Title: Request mediation
Requirements: 3, 7
Actors: Client, ESB, GlassFish
Main

1. Client sends SOAP message with SAML Token to ESB proxy
2. ESB extracts SAML token to get the client role
3. ESB removes SAML metadata from message
4. ESB adds metadata to message context.

5. ESB sends message to GlassFish endpoint

Extensions:

2a. SAML Token is invalid

2b. ESB sends error message to client

Precondition:

- Client is connected to ESB

Title: Response mediation
Requirements: 2, 3, 7
Actors: Client, ESB, GlassFish
Main

1. GlassFish sends message to ESB
2. ESB sets priority metadata in message context and SOAP header.
3. ESB retrieves bandwidth information (See Monitoring Service communication use case)
4. ESB prioritizes message (See Prioritize message use case)
5. ESB sends message to Client

Extensions:
Precondition:

- Request mediation

Title: Monitoring Service communication
Actors: Monitoring Service(MS), ESB
Main

1. ESB requests bandwidth information from MS to a specific address
2. MS returns bottleneck bandwidth to the ESB, as well as the address of the last Tactical Router before the endpoint.

Extensions:

1a. ESB specifies an invalid address

2a. MS returns no information

2b. Address is in the same sub net as the ESB

Precondition:

- Response mediation

Title: Prioritize messages
Requirements: 2, 6, 8
Actors: ESB
Main

1. ESB acquires QoS information through settings
2. ESB adds QoS information to the SOAP header of the message
3. ESB sets DiffServ field in IP header

Extensions:
Precondition:

- Response mediation
- Monitoring Service communication

6.2.3 Description of ESB concepts

In this section we will shortly describe some important concepts of the ESB and message mediation.

A mediator is the basic processing unit in Apache Synapse[42]. Each message going through the ESB gets mediated through a sequence of mediators, which can be configured through either XML or WSO2's graphical user interface. As long as the mediator inherit from a Synapse interface, any custom mediator can be used in the same manner as the built in mediators. To control the flow of messages through the ESB, there are two paths that can be controlled, the "in sequence" and the "out sequence", which can also be configured to only apply for certain endpoints.

The ESB is built around the notion of a message context, this object contains all the information regarding the message and the context around it. In the message context we can add properties, manipulate the message itself and manipulate the sending streams of the message. All the properties added during the receiving of a message is also added to the outgoing message, which we can use to our advantage.

Each mediator in the sequence get access to the message context of the incoming or the outgoing message and can thus manipulate the context to its liking. When the mediator is done with the work it is supposed to do, it either calls the next mediator, sending it the possibly altered message context or returning true to indicate that the work is done.

As you will soon see, we have taken full advantage of the modularity in the ESB. This means that even though much of the functionality in our mediators could be moved into one or two mediators we have decided to make many. For us this means much easier testing of each component and it gives each mediator a better defined role. For our customer this means an easier setup where they can mix and match each mediator and easily create custom sequences with just the functionality they need.

[42] Apache Synapse - An enterprise service bus

6.2.4 Dataflow

This section describes the data flow through the ESB with the help of two diagrams. As a bonus, these diagrams show the general architecture of the server side very well.

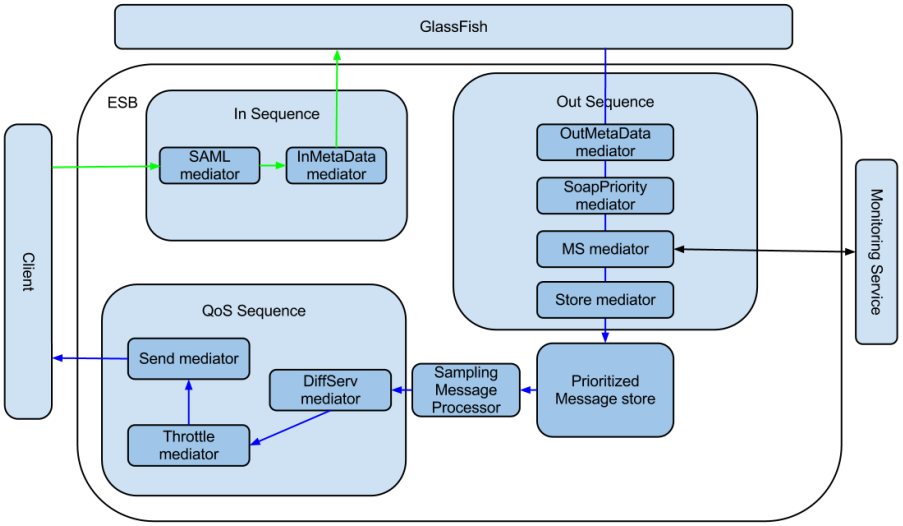


Figure 21: Server Data Flow
This diagram displays how the date flows through the server side.

Service Request :
To follow this flow, follow the green arrows in figure 21. The ESB receives a request message from a Client, it is then sent to the SAML mediator, and then to the InMetadata mediator which when done sends it to the service endpoint on the GlassFish server, and the flow is over.

Service Response:
To follow this flow, follow the blue arrows in figure 21. The ESB receives a response message from the Service, it is then sent through a sequence of mediators, first the OutMetadata mediator, SoapPriority mediator, MS mediator and then the Store mediator. The Store mediator stores the message in the Prioritized Message Store. The message is stored until the Sampling Message Processor picks it out before sending it on to another sequence of mediators. First in the sequence is the DiffServ mediator, then the Throttle Mediator and finally the Send mediator. The send mediator sends the message back to the client and the flow is completed.

SAML Authentication Request:
This flow is shown in figure 22.The ESB receives a request (from a Client) directed at the dummy Identity Server, the ESB then uses the SendBack mediator to send the same message it got in back. The message then travels to the Out

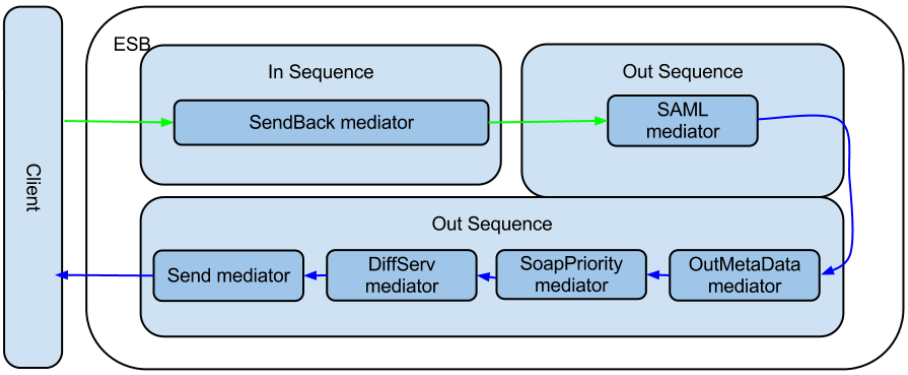


Figure 22: SAML Authentication Flow
This describes the flow of an authentication request.

sequence where it gets a priority, a DiffServ value and some metadata gets added to the header of the message. The reason why this is done is explained in section 6.3.4.

6.2.5 Extensions to the ESB

This section will contain a textual description of all the mediators used in the ESB. First we will describe all the custom mediators and then a short description of the built-in mediators we will use. All of our custom mediators have an accompanying sequence diagram to give a detailed overview of their inner working which is referenced in the title.

Custom mediators:

SAML mediator(24):
This mediator retrieves the user role from the SAML authentication and set this as a property in the message context. The service is retrieved from the 'recipient' field also found in the SAML authentication and added as another property. Depending on the configuration of the ESB this mediator can also detach the SAML authentication if this is no longer needed.

InMetadata mediator(25):
This mediator adds the IP of the client to the message context, which is done in order for the MS mediator to do its work. It will also set the Time-to-Live values in the message context if this is present in the SOAP header.

OutMetadata mediator(26):
This mediator retrieves the client role and service properties from the message context. These properties are then used along with a persistent registry to infer a priority for the message, and what the DiffServ field in the IP header should be set as. The priority and DiffServ values are then set as new properties in the message context. The DiffServ property in the message context will be used in

the synapse core to set the DiffServ field before sending the message (See B).

SoapPriority mediator(27):
This mediator adds the DiffServ value and the priority as two custom SOAP header items. We use these fields on the client side in order for the clients to use the same DiffServ value.

MS mediator(28):
This mediator retrieves the IP address[43] of the receiving client from the endpoint reference in the message context. It sends this IP address to the Monitoring Service and gets the IP address of the last Tactical Router on the path to the client, as well as the limiting bandwidth on the path. The mediator then sets this information as properties in the message context before sending the message to the next mediator.

Prioritized Message store:
This is not a mediator, but it is an important part of the response mediation sequence. This is a message store that stores messages in a priority queue. The queue is mainly ordered by the priority property of the message context, and secondly by the time when added. When retrieving messages from this store, the message on the top of the queue is returned. This ensures that high priority messages are processed before lower priority messages.

DiffServ mediator(29):
The DiffServ mediator sets the correct DiffServ value on the Socket. The DiffServ value is retrieved from the same value as the OutMetadata mediator put in earlier. Since correct use of DiffServ was very important to the client this mediator also does extensive logging which is important to look at when debuging.

Throttle mediator(30):
This mediator is used to ensure that high priority messages are sent first, by disrupting already sending messages, and it tries to ensure that the network is not being overflowed by this server by holding back messages. To determine what to disrupt and what to hold back, and for how long, several properties are used; the priority of the message, the available bandwidth, the IP address of the client side Tactical Router, and the real time demand of the request. In order to do this, the mediator must keep a list of sending messages and where those messages are going.

SendBack mediator:
This mediator sends the message back to the client, but before it is sent it is mediated through the out sequence of the ESB.

Built in Mediators:

Send mediator:
This is a built in mediator that sends the message to an endpoint (the requested

[43]IP address - A numerical label assigned to each device connected to the Internet

service).

Store mediator:
This is a build in mediator that stores the message context in a message store, here this is the Prioritized Message store.

Sampling Message Processor:
This is not a mediator. It is a built in class that takes messages out of the Prioritized Message Store at a defined interval. And then sends them to a mediator sequence, here starting with the DiffServ mediator.

6.2.6 Sequence Diagrams

This section contains some sequence diagrams which you can use to get a more in depth look into the code and methods used in the mediators above. The diagrams may not reflect the actual method names or display the full complexity of the code, but they should be sufficiently detailed that it should be possible to recognize them in the code.

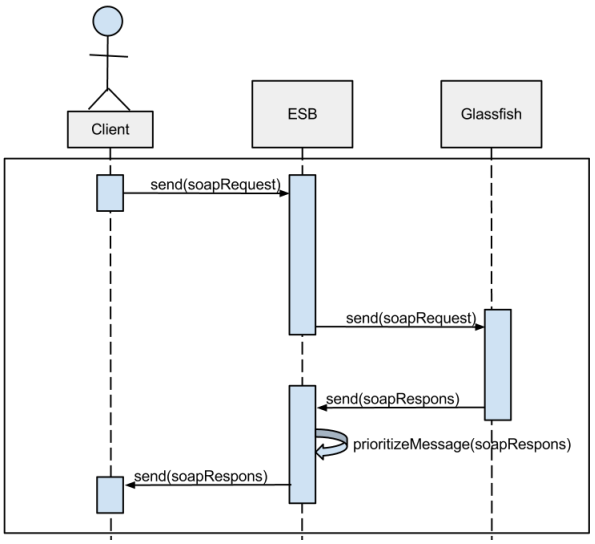


Figure 23: System-level sequence diagram
This high level diagram shows how the client communicates with web services through the ESB.

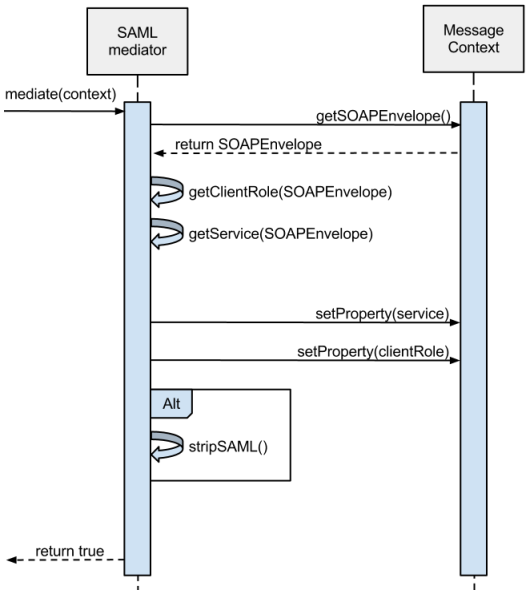


Figure 24: SAML mediator sequence diagram
This diagram describes how the SAML mediator will get data from the message, and set it in the message context so it can be used later in the response sequence

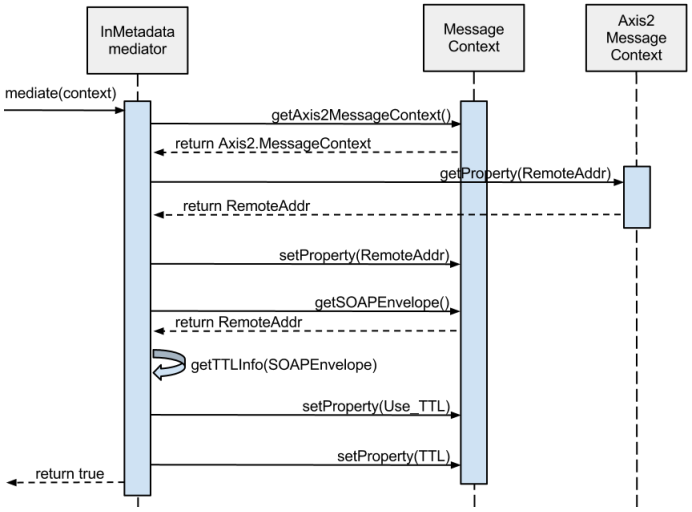


Figure 25: InMetadata mediator sequence diagram
This diagram shows how the InMetadata mediator works when it adds the IP address and Time-to-Live.

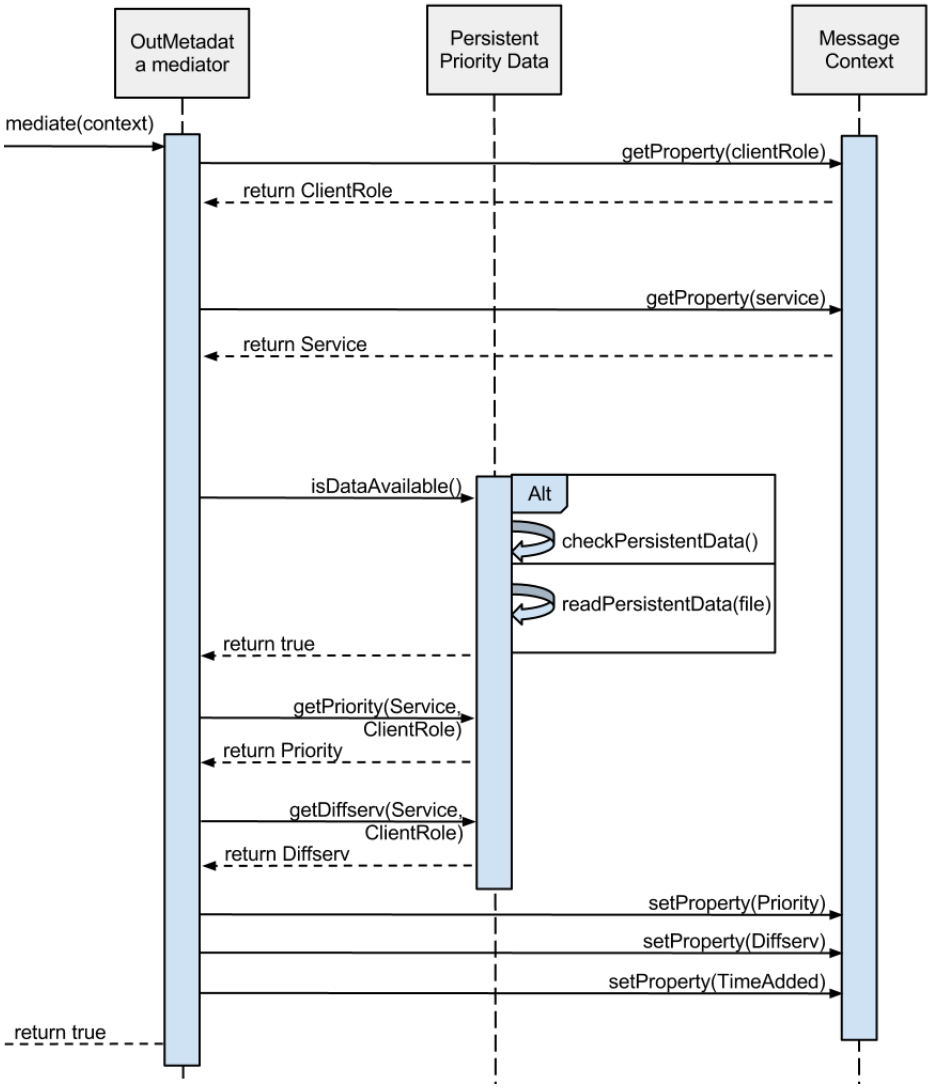


Figure 26: OutMetadata mediator sequence diagram
The diagram shows how OutMetadata mediator works.

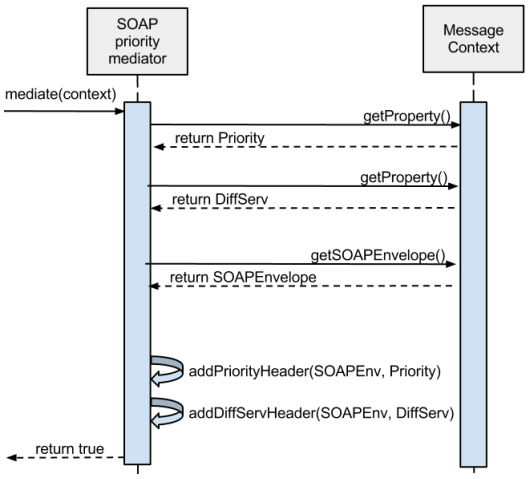


Figure 27: SOAP Priority mediator sequence diagram
This diagram shows the inner working of the SOAP priority mediator.

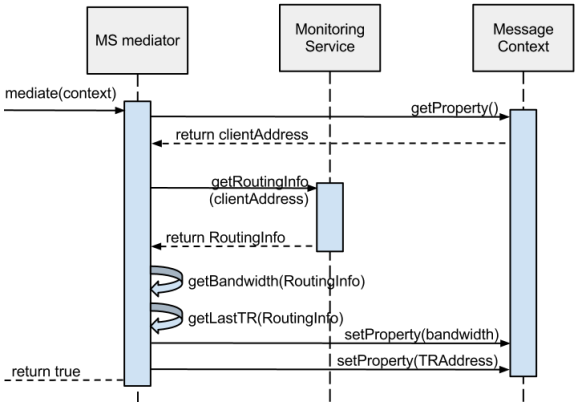


Figure 28: Metadata mediator sequence
This diagram describes how the MS mediator retrieves the routing information from the MSCommunicator and add it to the Message Context.

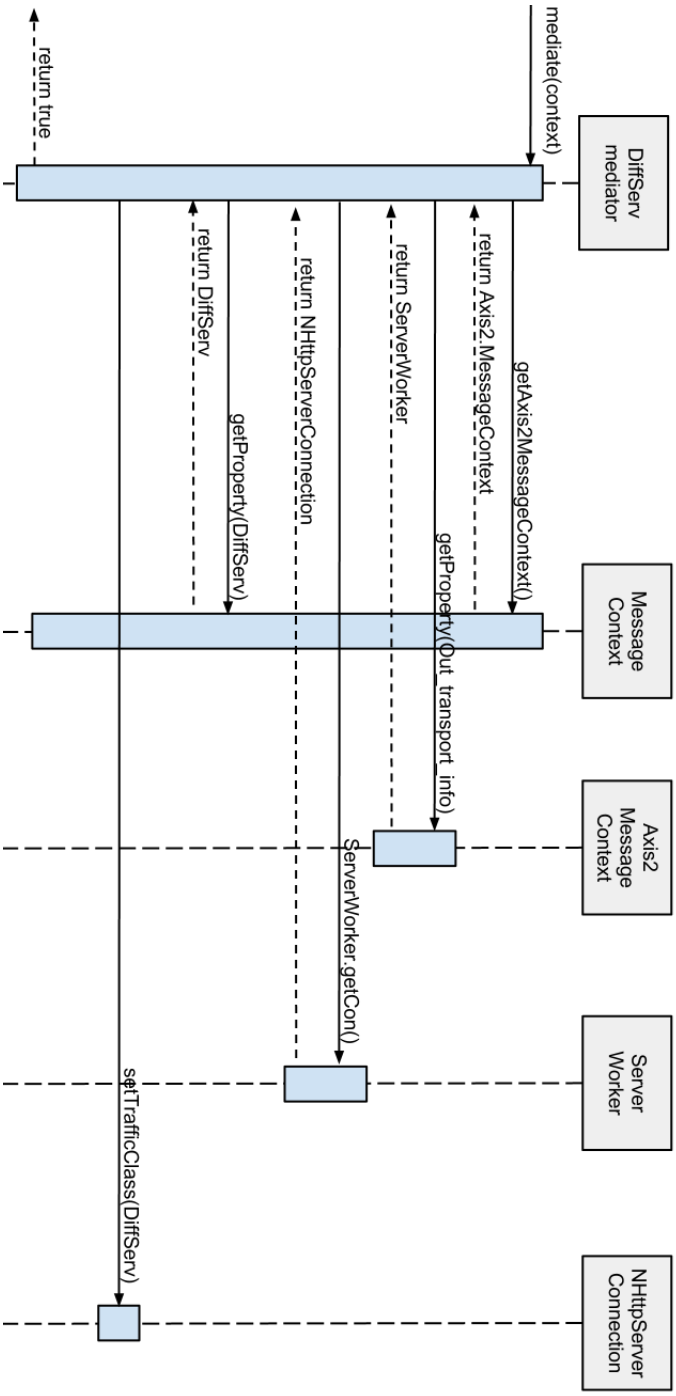


Figure 29: DiffServ mediator sequence diagram
How we set DiffServ priority on the underlying Socket. In diagram form.

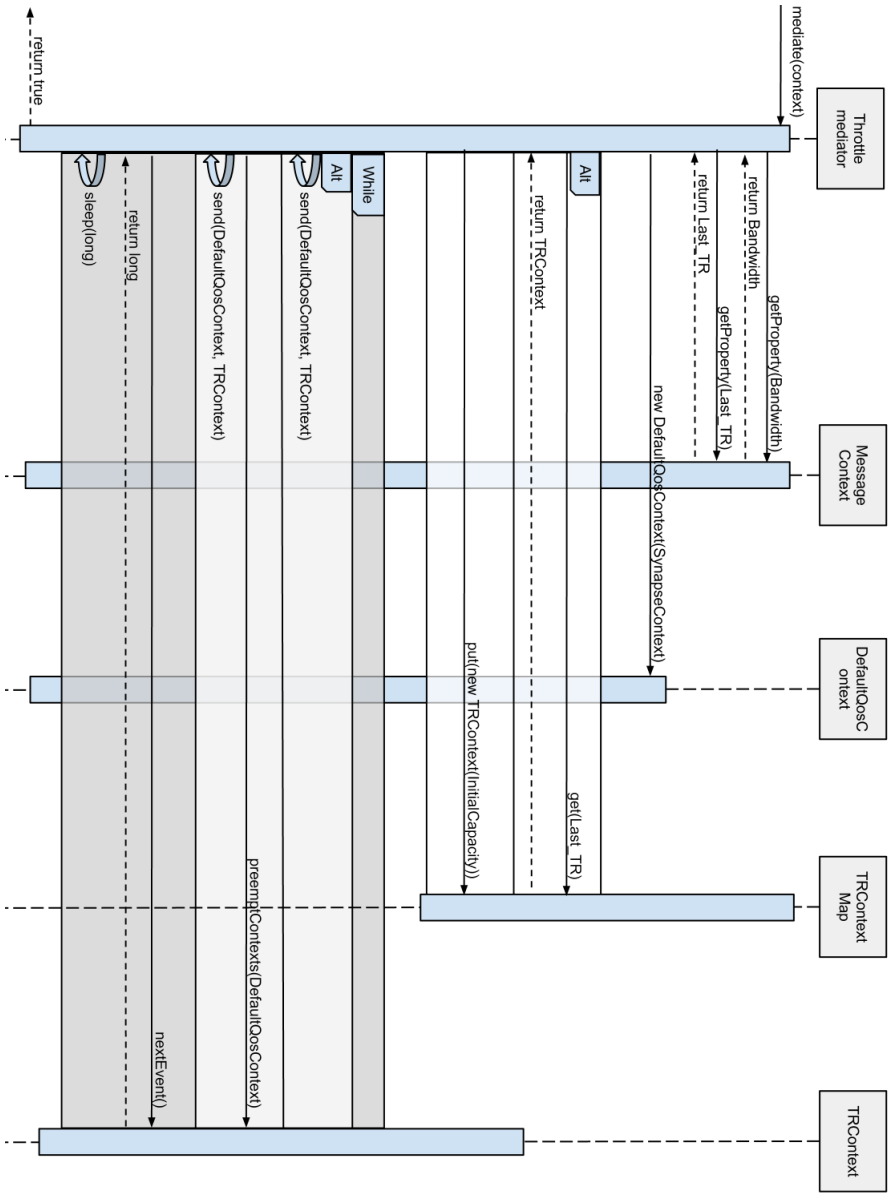


Figure 30: Throttle mediator sequence diagram
The meat of the server side mediators.

6.2.7 Configuration of the ESB

In this section we will explain how to configure the ESB. For general configuration of the ESB, f.ex. configuring new services or WS-Discovery, please refer to the official WSO2 ESB or Apache Synapse documentation.

It is highly recommended that you follow appendix B for an in depth guide on how to setup the ESB with the needed modifications before you read this

section.

First we will look at the file “ppd.xml”, which should now be in “/path/-to/wso2esb/”, ppd is short for Persistent Priority Data. This file contains maps from “service name” and “client role” to “priority” and “diffserv”. Here “service name” should be the path to the service on the ESB, “client role” should be the name of the client’s role, “priority” is the internal priority we use in the ESB (higher is better) and “diffserv” is the value that will be set in the IP header on communications with the client. Both DiffServ and priority must be integers. The service element also has the useDefault property, which when set to true lets roles not configured in this file use values in the default role. When useDefault is set to false unconfigured clients will get a priority and DiffServ of 0. If useDefault is set to false there is no need to configure a default client for the service. Below is an example setup of a service in ppd.xml.

```
1  <config>
2     <services>
3              <service name="/services/EchoService"
4              useDefault="true">
5                    <client role="clientRole1">
6                      <priority>100</priority>
7                      <diffserv>10</diffserv>
8                    </client>
9                  <client role="Default">
10                     <priority>321</priority>
11                     <diffserv>8</diffserv>
12                 </client>
13             </service>
14    </services>
15 </config>
```

Next we look at a file that is specific to our implementation of the MSCommunicator (Monitoring Service Communicator). Since our implementation does not actually have a monitoring service to contact we use the file “ms.xml” in “/path/to/wso2esb/” to configure data groups of destination IP, name of last Tactical Router before client and the bandwidth capacity of the ‘weakest’ link on the path measured in KBps. Below is an example configuration.

```
16 <config>
17    <RoutingInfos>
18             <RoutingInfo>
19                 <destIP>127.0.0.1</destIP>
20                 <lastTR>bob</lastTR>
21                 <bandwidth>0.2</bandwidth>
22             </RoutingInfo>
23    </RoutingInfos>
24 </config>
```

If the MSCommunicator is modified to communicate with a Monitoring Service this file will not be needed anymore.

The last file we will look at is synapse.xml (ref:G - synapse-configs.zip), which should now be in "/path/to/wso2esb/repository/deployment/server/synapse-configs/default/". This file contains configuration for proxies, endpoints, message stores, mediation sequences, and more. The important things here are:

- The sequence qos where we can find the configuration for the throttle mediator. Here we can set the property minBandwidthPerMessage (integer, measured in Bps) and timeout(integer, measured in ms). Timeout is the longest time a message will be allowed to try sending before it is discarded.
- The messageProcessor. here the parameter interval can be set. This determines how often a message should be taken out of the message store. This is measured in ms.

Other things in this file should mostly be untouched, as they define what the ESB does with messages, most of which is needed to do the prioritizing and throttling.

6.3 Changes

There are a lot of changes from the initial prestudy design to the final design described in this chapter. The prestudy design did not go into a lot of detail because of limited knowledge about the systems we were to be using. Some assumptions made proved to be wrong, and a lot of difficulties popped up along the way. We had a very long planning phase before the implementation in this project. Before we started the implementation we were much closer to the final design than we were in the prestudy, with only a few notable changes made during implementation. These changes are what we would like to discuss in this chapter. This discussion should help you get a better understanding of some of our discoveries, and maybe why we ended up with the product we now have.

6.3.1 Client Specific Changes

Sanity checker:
The design of the client library initially called for a sanity checker to validate the messages the client was trying to send and the credentials the client supplied. During implementation we discovered that in order to sanity check the soap message it would either have to be an excessively simplistic sanity check, or we would have to parse the entire message. Parsing the entire message would sanity check it far more thoroughly than we would have been able to, so the sanity checking of soap messages was removed from the sanity checker module, and we ended up relying on the sanity checking built into the parser in Axiom.

MSCommunicator:
Initially the plan was to integrate the MSCommunicator module in the client library. But we realized that bandwidth data is not necessary for deciding on the DiffServ-value, as that is decided by the Identity Server and returned through SAML. Additionally, due to the fact that the data sent by the client will be smaller than the data returned from the web service most of the time, and the network will usually consist of several clients connected to one service we saw no need to prioritize and hold back messages on the client side based on the bandwidth of the network.

6.3.2 Server Specific Changes

On the server side there was not a lot of big changes during the implementation.

The biggest change from early design to final design is probably the ThrottleMediator. Before we started the implementation, we were very unsure of what we would be able to do in this mediator. So we wrote down a few things we wanted to try, most of which we managed to implement. We wanted to make it more dynamic, and maybe learn something from how long data took to be sent to the different endpoints. Time was a limited resource, but we ended up making more out of it than what was initially anticipated.

Initially OutMetadataMediator did the work of SoapPriorityMediator as well, but we split them up because we figured several more specialized mediators would be more modular and easier to modify later.

During implementation we found out that we had to get the IP address of the client in the 'in' sequence for use later in the 'out sequence, so we made InMetadataMediator take care of this as well as getting potential time to live data.

Because of the lack of the Identity Server we needed an alternative to send the diffserv value back to the client. So we made the SAML-sequence described in 6.2.3 under SAML Authentication Request.

6.3.3 Changes that affect both sides

OpenSAML:
OpenSAML is not used at all. If we had succeeded in implementing the Identity Server, the IS would take care of all SAML generation. Since we use a dummy layer in the ESB to "simulate" the IS, the client needs to generate SAML, but it takes far too much time to initialize the OpenSAML libraries for it to be useful in our case, so we decided not to use it, and instead use Apache Axiom to build a proper SOAP-wrapped SAML-message based on some hardcoded strings, and variable roles and timestamps.

Tokens:
The biggest change since the initial design finalization is how tokens are fetched and used in our implementation. The initial idea was that the server side would contain an identity server, which our client would identify itself towards. As it turned out, setting up and using the identity server was a near impossible task (see details on why below) that would have taken us far beyond our project deadline, so together with the customer it was decided that since this wasn't a part of the requirements for the project, it could be dropped. What we ended up with was a far simpler system where the client itself creates a token, which is then sent to a simple echo service on the server to get the SOAP headers we needed from it.

6.3.4 Regarding the Identity Server

Our original design called for the implementation of the WSO2 Identity Server, but our final product does not include it as we ran into some problems while trying to set it up. After discussing our situation with the customer, they agreed

that we could drop it. It was, after all, not a functional requirement from their side, though it would be preferable to have it included.

There are several reasons why we failed to implement the Identity Server, one of which is that we started our research of the IS too late, only two weeks before our prototype demonstration, and we only had one group member working on it. After seeing how well documented and fairly easy to use the WSO2 ESB was, we figured that the IS couldn't be much worse, but we figured wrong. Which leads us to the next source of our problems; the WSO2 Identity Server product page at http://wso2.com/products/identity-server is severely lacking in documentation. The user guide and administration manual contains barely no information about how to configure it and set it up to be usable in different usage scenarios. They provide links to some blog posts that employees had written back in 2009, and even though the blogs contained some useful information, and sometimes provided example configuration files and client code, they did not state which versions of the different products they were basing their examples on, so we don't know if there were compatibility problems between different versions of the IS and ESB.

We used this blog post http://blog.facilelogin.com/2009/05/accessing-proxy-services-in-wso2-esb.html to try and set up communication between the IS and ESB. It was not entirely similar to our use case, as it uses X509 signing and encryption with HTTP transport, instead of using HTTPS and let the transport layer take care of security. When we tried to use the supplied client code and configuration files, we got some problems with the latter, as the client code would not accept them, throwing exceptions stating that they were not of the correct format, though the IS and the ESB accepted them. If we changed the configuration files so that the client would accept them (the only adjustment needed was to change the name of a tag in the WS-security policy XML-file, from <sp:Policy> to <wsp:policy>) then the IS and ESB would not accept them. Since it was just a minor adjustment, and the rest of the policy stayed identical, we don't believe this caused any problems, but it is an example of how frustrating it could be trying to use the code provided, as it was poorly commented and assumed you had previous knowledge of how Apache Axis2, Axiom, Rampart and Tomcat worked, since the WSO2 IS builds upon these products.

After much trial and error, a lot of exceptions and googling for solutions, we managed to get the IS to issue security tokens and send them to the ESB, but the ESB failed in decrypting and verifying them. We were unable to figure out exactly why it failed, as the error message we got was that the ESB could not find the public key of the Identity Server, but using the Java keytool we could verify that the key was in fact present in the ESB key store. After spending quite a few hours trying different solutions, exporting the IS public key and importing it to the ESB key store under a new alias, importing the ESB public key into the IS key store etc. we gave up, as this specific use case was not the one we were after, and we had at least succeeded in getting the ESB and IS to talk to each other.

Next, we tried configuring the IS to issue username tokens and send them over HTTPS, which would remove the need for endpoint encryption and was after all the use case we had in mind. We could find no specific examples for this scenario, so we tried creating our own security, policy file, since the one from the previous example specified endpoint encryption, and adjusted the settings in the

ESB and IS. In this way we managed to get the IS to create username tokens, but nothing more, as it crashed when trying to send it to the ESB, stating that the SOAP header did not include a security element. Monitoring the SOAP messages we could see that the header actually did contain this element, so we don't know why the IS couldn't find it.

This was as far as we got before our prototype demonstration, and as already mentioned, our customer agreed that we could drop the Identity Server and instead use a dummy layer in the ESB. This means that our final product does not use the IS, and users cannot log on to the system because there is no user store, so the client will create static SAML-tokens which it sends to the dummy layer in the ESB, which then returns the same SAML-token, but wraps in in a SOAP message containing information about the clients priority in the system and its diffserv value.

Had any of us had some experience with WS-security from before, and been familiar with the WS-security policy language, and the Axis2, Axiom, Rampart and Tomcat from Apache, which the WSO2 Identity Server builds upon, we might have been more successful. We were quite frankly stumbling around in the dark, without knowing exactly where to begin.

7 Testing

This chapter will introduce our testing setup together with the result of our testing. We will start by introducing the testing suite, how the testing client works and we will also have a discussion about strengths and weaknesses of the system as a whole. Then we will introduce the actual tests and present our results. After you have read this chapter it should be clear to you how to setup the testing suite, how to replicate our tests and you should have an in depth view of our results. The result section will also contain a discussion on the weaknesses which have had an effect on the tests and how we have interpreted our results despite this.

7.1 About the testing setup

7.1.1 Suite

Since we chose to do most of our testing on NS3 using the MobiEmu[44] framework most of our tests are fully automated. As we have not manage to integrate everything into the testing framework some variables are still static, but these will be highlighted where applicable.

Please refer to appendix C for instruction on how to set up the testing framework. And for a description of the different variables in the test.

As we alluded to in section 3.5 we had quite high hopes for how we were going to test the whole system. Unfortunately for us that did not pan out the way we wanted it to. We had some major problems regarding NS3 and how it connects

[44] A framework for emulating mobile ad-hoc networks with Linux containers and ns-3. https://code.google.com/p/mobiemu/

its nodes to the Tap-Bridges created. This led us to having to rethink our whole test setup. We decided, after talking to the customer, that we would change the test and create a simpler network layout which would work with NS3. In figure: 31 you can see this new altered network layout. What this means for the project is that we can not test all the functionality that we wanted to, but we are still quietly confident that we can draw some conclusion about the results. In appendix E we have outlined the problems we faced and a possible solution that we did not have time to implement.

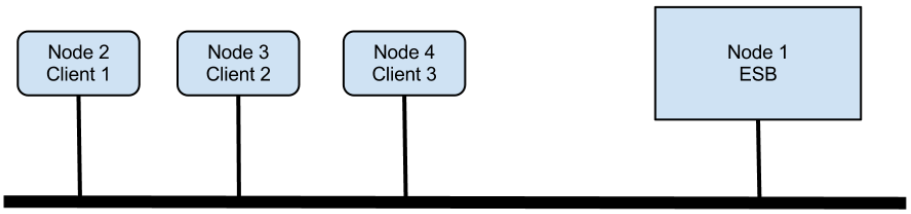


Figure 31: The layout of our network during testing
In this figure we have illustrated the layout of the network during NS3 testing.

This new layout does limit the scope of the test and also what we can approve of functionality, but still we think we have a strong end product with results that will absolutely be relevant for the customer.

The MobiEmu testing framework operates by creating LXC and connecting these to tap bridges created by NS3. This then emulates any network possible to create in NS3. From the point of view of programs running inside the LXC, they are full Linux machines connected to a real network. This means that any program able to run on Linux should run properly inside the LXC and any messages they send out is sent through NS3. This means that we have full control over how our network behaves and we can emulate quite a lot of scenarios.

When MobiEmu start up it creates a number of LXCs, it then starts up NS3 outside of any LXC and connects each of the LXCs to a corresponding tap bridge in NS3. Inside each of the LXCs it then starts the experiment and waits for the whole thing to finish before it completes nicely. Before each run MobiEmu stores all files and folders in the whole folder in order to easily recreate an experiment. When the experiment is done the result files are moved into the result folder.

Our tests then are set up as follows. We have three clients, two clients with low priority and one client with high priority. They send messages to the ESB which is connected to the same LAN as all the clients. The specific test client we use is describe in the following section. In the section 7.2 we have a detailed description of the reason behind each case. We test with several different bandwidths in order to test our setup and see how it handles a verity of different bandwidths. The last thing that we test is with different "Timeout" on the ESB, this is done in order to test what setting works better for the different bandwidths. We do it this way because of the static nature of configuration on

the ESB, detailed in the section 6.2.7.

7.1.2 Test Client

In this section we will shortly describe the test client used for testing, "EchoClient-Client.jar"(ref:G).

The client starts by reading its configuration file. Its filename can be provided as the only command line argument, otherwise it defaults to "client.config'. This configuration file contains variables such as username, password, role, service to contact, what the request should be, how many requests to send and with what interval it should send them at, and some describing what to log.

The client expects the request to contain 'REQID' as part of the request message, and before sending it, the client will replace it with 'REQID=XX' where 'XX' is the number of the request. This way the client can check that it gets the right response by checking whether 'REQID=XX' is present in the response, and whether the ID is correct. This makes validating the response from the service easier, but it restricts the service to put the request message somewhere in the response. Another side effect of this being the only validation is that other errors in the response are not easily detected as long as the 'REQID=XX' is there. If for example the response is cut short, by the stream being cut, as long as the ID is present it will be treated as valid. The client also prints the length of the response, so scripts that parse the results can pick up on responses with abnormal lengths.

After reading the configuration file the client makes an instance of the client library described in section 6.1, using the username, password and role, and itself as an ExceptionHandler. By implementing itself as the ExceptionHandler the client will be notified about all the exceptions occurring in the client library, the client does not act upon these exceptions, but it does log them. A normal client might want to send a request again here.

A timer is used to start the sending/receiving in a new thread at the interval specified and the number of times specified. Starting new threads for every request and sending at a small interval is a good way to test how the client library handles concurrent requests.

The actual sending and receiving of data is rather simple, as you can see in this code snippet:

```
1 /* Uses connection.sendData(data, destination) to send a
      request. Here {REQID} in the request found in
      clientConfig is replaced with {REQID=reqID}. The client
      library will put the response in the returned
      ReceiveObject when it is received. */
2 ReceiveObject ro = connection.sendData(
3 config.get(DATA).replace("{"+REQID+"}", "{"+REQID+"="+reqID+
      "}"), destination);
4 try {
5     logLine("Waiting for response "+reqID);
6
7 /* Calls the blocking method ReceiveObject.receive() to get
      the response. An alternative to this would be to make a
      DataListener and add it to connection This way, every
      listener would receive all the responses, and would be
      unsuited for this test */
```

```
8       String response = ro.receive();
```

For more details about configuring the test client read the example configuration provided in appendix G. For more detailed information on how the test client works, the java, TestClient.java, file with javadoc is also provided in appendix G.

7.1.3 Test Service

In this section we will shortly describe the test service used for testing, "EchoServiceLargeReply.war"(ref:G).

The test service is very simple and deployable with GlassFish. It expects a request like this:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
       envelope/">
3      <S:Header/>
4      <S:Body>
5          <ns2:hello xmlns:ns2="http://me.test.org">
6              <name>PAYLOAD</name>
7          </ns2:hello>
8      </S:Body>
9  </S:Envelope>
```

And responds like this:

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
       envelope/">
3      <S:Body>
4          <ns2:helloResponse xmlns:ns2="http://me.test.org">
5              <return>
6                  PAYLOAD!
7                  Lorem ipsum dolor sit amet, consectetur
                       adipiscing elit. Etiam sodales magna at
                       est iaculis vel fermentum velit tristique
                       . Cras nulla urna, ultrices vitae posuere
                        a, iaculis sit amet lectus. Aliquam
                       mattis sapien et elit commodo ...
8              </return>
9          </ns2:helloResponse>
10     </S:Body>
11 </S:Envelope>
```

With PAYLOAD intact, and 10KB of Lorem Ipsum[45]. We add these 10KB of text to ensure that the response is considerably larger than the request, which should get us more realistic results when testing. We also send the original payload back so the test client can easily identify which request was responded to. Those 10KB of text also has a large impact on testing, it means that on lower than 10KBps bandwidth all the messages can't be sent in time. We chose to have it this way because it would give us a predictable test and also some static configuration on the ESB could be configured with this in mind.

[45] http://www.lipsum.com/ - simply dummy text

7.1.4 Weaknesses

There are some weaknesses connected with our testing suite and how we do the testing. Chief among them is the limited scope made necessary by limitations encountered in NS3. However there are other areas where the testing suite could be expanded which could be done without butting heads with NS3.

One limitation that was self imposed is the fact that we only have one test network layout. This should have been expanded, but because of limited time at the end of the project we chose to focus more on the one test. With a more expanded network layout, which is quite feasible despite the problems encountered with NS3, one could add more clients and introduce several priorities to test how the ESB would behave. We theorize that the ESB should behave in the same way and we have created it in such a way to prioritize the highest priority messages no matter what the other messages does.

Another limitation to the test setup is that we have no easy way to communicate between the LXCs. What this means is that we can not coordinate when to terminate the whole test. What this means is that we have to enforce a cutoff time which does skew some test. Especially bad is this when we run the test without our Throttle mediator. Because without our Throttle mediator even on the lower bandwidths no messages should be lost and the percentage of successful messages should be one. This will effect the test, but we decided to keep it this way because we mean the general trend on the results are still clear.

The test client also has some problems. For one it does not try to retransmit any messages. This has a profound effect on the results regarding successful message percentage which will be quite different with and without our Throttle mediator. With our Throttle mediator we will perceive a lower percentage compared to without the mediator, but this is just a result of us dropping messages which retransmitting would to some degree correct. Another thing worth mentioning is that we have scheduled all the clients to start about the same time, there is a slight delay between them, but this scheduling means that on lower bandwidths there will be a distinct sending period where all the clients sends messages and there will be a distinct receiving period.

For the lower bandwidth test there is also the fact that the client library uses HTTPS which we have observed in the lowest bandwidths do sometimes timeout because of the size of the handshakes which do somewhat interfere with the results. This should however be quite insignificant as the results sent back from the service is so large and would guaranteed timeout if the smaller handshake messages timeout.

On the server side the biggest limitation is the static nature of the setup. We have tried to make the test so that we could test as much as possible, but there is one variable which we have not gotten to tweak. On the ESB we can configure the interval in which messages are taken out of the message store, but because of limited time to perform the tests we could not test this. For the results this means that the time taken for the lower priority clients will be a bit skewed, but again the trend should be clear.

7.2 Test Cases

As most of the tests below are quite alike the reasoning behind them are also quite like. The main difference between them are the “Timeout” which refer to

the timeout the ESB uses. For more about the "Timeout" see 6.2.7.

Since this project had somewhat of a research focus from the customers side we did not perform these test in order for us to validate our system. Instead we have run these tests to try and say something about the feasibility of the original question asked when we started. We will come back to this topic in the result section.

We used the same client setups for all of the tests. The low priority clients, Client 1 and 2, was configured like this:

```
1  %This is a comment
2  %variables are written with NAME:VALUE
3  %line without ':' or '%' is treated as end of file.
4  %doLog:boolean, whether to log or not, default is true.
5  doLog:true
6  %logToFile:boolean, whether or not client library should log
       to file, default is false.
7  logToFile:true
8  %username:String, Must be configured.
9  username:testname
10 %password:String, Must be configured.
11 password:testpassword
12 %role:String, Must be configured.
13 role:clientRole1
14 %service:URI, Must be configured.
15 service:https://10.0.0.1:8243/services/EchoService
16 %interval:long, between requests in milliseconds, only one
       request if not configured
17 interval:1000
18 %nofreqs: int, number of requests to send, send forever if
       not configured
19 nofreqs:100
20 %delay:long, wait before first request in milliseconds,
       default is 0
21 delay:10
22 %request:SOAP, Must be configured.
23 request:<?xml version="1.0" encoding="UTF-8"?><S:Envelope
       xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:
       Header/><S:Body><ns2:hello xmlns:ns2="http://me.test.org
       "><name>{REQID}</name></ns2:hello></S:Body></S:Envelope>
```

Client 2 had 0 as delay. While the high priority client, Client 3, had interval=3000, nofreqs=30 and delay=15.

ID	1
Description	In this test what we are looking at is how our system behaves with a very low timeout, since we have full control over the message sizes sent in the test we know that this timeout will be too low on the lower bandwidths, but should perform much better on high bandwidths.
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: 500**, interval: 500, minimum bandwidth per message: 10240
Automated	Yes
Expected Result	We expect to see that the ESB will preempt even higher priority messages in the lower bandwidth tests because of the low timeout, but on higher bandwidths the sending time of the all the messages should be lower than on the later tests. To put that in the same setting as our results, we expect the percentage of successfully received messages to be lower than in the tests below, but we expect the time to also be lower across the board.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

ID	2
Description	In this test we have increased the timeout substantially, we expect to see some improvements on 10KBps and still retain some of the benefits of a lower timeout on 20- and 40KBps
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: 1000**, interval: 500, minimum bandwidth per message: 10240
Automated	Yes
Expected Result	We expect to have a higher percentage of completed messages on 10KBps than with a timeout of 500 and we expect the results on 1-, 20- and 40KBps to be relatively unchanged.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

ID	3
Description	Again we have increased the timeout and expect to see some improvements on percentage, but we the total time taken should start to drop on higher bandwidths.
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: 2000**, interval: 500, minimum bandwidth per message: 10240
Automated	Yes
Expected Result	We expect all the messages on 20 and 40KBps to arrive, we expect that on 10KBps more messages should arrive, but not all. The time taken should again increase.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

ID	4
Description	In this test we want to see how the ESB copes with a much larger timeout.
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: 5000**, interval: 500, minimum bandwidth per message: 10240
Automated	Yes
Expected Result	We expect that 10-, 20- and 40KBps should be enough to get most of the messages for the high priority client through, the time taken should again increase and this should be noticeable on 40KBps compared to Test 1.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

ID	5
Description	In this test we have gone all out. The timeout is massivly increased to see how the ESB behaves on the lowest bandwidths, 1- and 5KBps respectively.
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: 100 000**, interval: 500, minimum bandwidth per message: 10240
Automated	Yes
Expected Result	We expect the same percentage on 10-, 20- and 40KBps as Test 4. What we want to see is that on 5KBps the percentage is increased quite substantially compared to the previous tests.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

ID	6
Description	In this test what we have done is to remove our Throttle mediator which should mean that our ESB setup will no longer be doing any throttling of messages. The message queue is still there so there will be some priority in the sending and receiving. We want to test this because this should give us some idea about how our setup will do against no QoS at all.
Repetition(s)	10
NS3 variables	Datarate: 1KBps,5KBps,10KBps,20KBps,40KBps
ESB variables	**Timeout: N/A**, interval: 500, minimum bandwidth per message: N/A
Automated	Yes
Expected Result	We expect that the average percentage of all the clients will be slightly above what our test can do, we refer to section 7.1.4 for some elaboration about this. What we want to see is that on lower bandwidths the average time taken for messages to arrive at Client 4 will be substantially higher than when we use the Throttle mediator. This will indicate that our Throttle mediator actually does some useful work and also indicate that this could be a viable strategy for our customer to continue researching.
Folder with compressed test	In appendix G you can find the appropriate compressed test case.

...

7.3 Results

8 Conclusion

[The summarised findings of the project and presentation of the key findings.]

8.1 Project accomplishments

[Did we reach the goal?]

8.2 Future Work

In this chapter we will discuss what can be worked on further and improved in the future. Some things we did not have time to do as well as we might have wanted, and some things that we knew would have to be made/modified later.

8.2.1 Server

In our MS Mediator we use a dummy implementation of the MS Communicator, which just reads an xml-file with the needed data. A future work would be to make an implementation that acutally contacts a Monitoring Service and use this in the mediator.

The throttling done in the ESB is relatively static, one configuration will work very well for some scenarios, but might not work as well for other ones. In the future, making it more dynamic might be desireable. The first thing to do might be to make and use a new implementation of the Message Processor instead of the built in SamplingMessageProcessor, retrieving a messages from the Prioritiezed Message Store dynamically instead of just once every predefined interval milliseconds could get you a long way. Also the Throttle mediator could be made more dynamic, for example by varying the now static variables based on percieved network load.

If propper SAML authentication is implemented the IdentityServer proxys sequence would have to be modified as well.

8.2.2 Client

As mentioned in the changes secton (6.3), the components related to authentication on the client side were not implemented as desired due to the problems with the Identity Server. Naturally, this is something that can be adressed in the future, when a proper IS is working.

8.3 Prototype demonstration

Discuss the prototype presentation on monday and the results of it.

9 Project Evaluation

[This is the section where we evaluate the project and the past process.]

9.1 Task evaluation

9.2 Team organization

The problem we had with the roles in the group is that we did not manage to change up the roles during the project. This has resulted in some instances where we could not proceed with something because someone was ill. This was something that we were quite wary of in the beginning of the project, but we did not follow it up with the same care. Some of the reason behind this is that is was sometimes just easier to ask the person in charge to tell you about the parts that you did not know about. Another reason was that we worked together every day during the project which probably gave us the false confidence that we would not need to share the information. The few early roadblocks that we did encounter during the start up when we could have done something about it were probably so small that we just carried on without thinking about it.

9.3 Planning

9.4 Methodology

9.5 Meetings

9.6 Communication

Group communication
The communication inside the team worked good, because we were working in the same room most of the time. This contributed to the prevention of conflicts. We also had some "Team building" that helped us not get on each others nerves.

Supervisor communication
The overall communication with the supervisor has been as expected. Some times there would have been nice with an answer to some of the emails about meetings. Typically we ask the supervisor for a meeting, decide time and place, and the supervisor doesn't confirm the meeting time, but he shows up so there is not really a problem. The communication with the supervisor has been as expected.

Customer communication
The communication with the customer went well. We had a lot of communication over email, and we had weekly meetings over Skype.

9.7 Design phase

- went smoothly, but did the design work out as we planned it?

9.8 Implementation phase

9.9 Overall Summary

A Client User Guide

A.1 Intro

This is the user manual for the QoS client library. Here we will cover how to use the client library to communicate with our server implementation, in a way that properly set the diffserv value on the packages sent from the client to the server. This manual is intended to be as simplistic and understandable as possible.

First we will cover the steps you need to take in order to be able to use our library. Then move on to the easiest way to use it, as well as how to use it with a listener pattern. This will be followed by a section covering the available settings in the library, and finally we will bring up some caveats of our implementation that is worth keeping in mind.

A.2 Required interfaces

One of the most important parts of all network communication is being able to catch and handle exceptions gracefully. In order for a client, the user of our

library, to be notified of exceptions that occur they will have to implement the ExceptionHandler interface found in no.ntnu.qos.client.ExceptionHandler. This interface contains several methods, which are:

Listing 1: ExceptionHandler interface

```
1  /**
2   * URI is malformed/invalid
3   * @param e UnknownHostException
4   */
5  public void unknownHostExceptionThrown(UnknownHostException
       e);
6  /**
7   * Problem reading variables, input, streams or strings
8   * @param e IOException
9   */
10 public void ioExceptionThrown(IOException e);
11 /**
12  * Problem with the HTTP connection in the form of timeouts,
        too many retries, etc.
13  * @param e org.apache.httpcomponent.HttpException, cast to
       generic Exception for convenience.
14  */
15 public void httpExceptionThrown(Exception e);
16 /**
17  * Problems with the socket, invalid SSL port or socket
       closed from service due to capacity problems.
18  * @param e SocketException
19  */
20 public void socketExceptionThrown(SocketException e);
21 /**
22  * Input message is invalid or malformed.
23  * @param e UnsupportedEncodingException
24  */
25 public void unsupportedEncodingExceptionThrown(
       UnsupportedEncodingException e);
```

The most important of these is the IOException method. This is the exception that will be thrown (As a straightforward IOException, or as a subclass of it called NoHttpResponseException) whenever the connection is closed without receiving the full reply from the server. In addition to the normal occurrences of this exception it will also happen whenever the server decides to cut a connection for priority reasons.

It is worth noting that whenever an exception is thrown for a specific request, the response for that request is set to the name of the exception, so as to enable the client to find out which request has been terminated by an exception.

A.3 Using the library

Once you've implemented the ExceptionHandler interface the client library can be easily constructed using:

Listing 2: Constructing the library

```
QoSClient client = New QoSClientImpl(String username, String
    userrole, String password, ExceptionHandler this);
```

Once you have a valid instance of QoSClient sending any data will be as simple as:

Listing 3: Sending data

```
ReceiveObject replyObject = client.send(String soap, URI
    destination);
String reply = replyObject.receive();
```

The string you send has to be a valid soap message. The soap message has to contain a valid envelope with a body element, more on this in the caveats section.

The ReceiveObject you get back from the send method is a blocking string, which means that calling receive on it will block execution until a reply is available.

A.4 Using the library with listeners

After constructing the library as shown above, you are able to add listeners to it using:

Listing 4: Add listener

```
    client.addListener(DataListener listener);
```

And remove them again using:

Listing 5: Remove listener

```
    client.removeListener(DataListener listener);
```

The DataListener referenced here is the interface DataListener in the qos client library, it requires that you implement a single method:

Listing 6: The DataListener interface

```
/**
 * Default receive method
 * @param recObj SOAP data
 */
public void newData(ReceiveObject recObj);
```

Which will be called whenever a ReceiveObject gets some reply data.

A.5 Change Credentials

If you for some reason wish to change the user credentials during execution, this can easily be done by calling:

Listing 7: Changing user credentials

```
    client.setCredentials(String username, String role,
        String password);
```

A.6 Logging

The client library supports logging both to file and console, by default it only logs to console, and only warnings and above. To set whether to log to console you can call:

Listing 8: Turn logging to console on or off

```
1    client.setLogToConsole(boolean on);
```

To set logging to file:

Listing 9: Turn logging to file on or off

```
1    client.setLogToFile(boolean on);
```

To change the level of the logging between warning and above, or everything:

Listing 10: Switch between the two logging scopes

```
1    client.setFineLogging(boolean on);
```

A.7 Caveats

A.7.1 URI from client

It is assumed by the client library that the port of the URI is a valid SSL port that is capable of communicating using TLS. It does however nothing to validate the authority of the servers certificate and will accept any certificate.

A.7.2 SOAP from client

The client library requires that this is a valid SOAP envelope, with an element with the local name of "Body" one level inside the envelope. Note that all XML is case sensitive.

A.7.3 User credentials

User credentials are not checked for validity (beyond very basic sanity checking) until the library attempts to get a token.

For our implementation of the client server set, this doesn't matter much since no credential validation is ever done, but it might matter if the client library is extended to interact with an actual identity server. If this is done, both the SAML communicator as well as the SAML parser should be reimplemented.

A.7.4 Redundant token fetching

If two or more requests are sent to the same service-set at the same time and the token has not already been fetched for that service-set there is a high probability that the token will be fetched over the network several times before being stored in the credential storage. Meaning you will waste network bandwidth getting the same information several times.

A.8 Example code

A simple client that will send a soap containing “Hello World” to the service at “https://localhost:443/service/myService” and retry 4 times

Listing 11: A simple example client

```
1  public class ExampleClient implements ExceptionHandler {
2     final static int retry = 4;
3     public static void main(String[] args) {
4         URI destination = new URI("https://localhost:443/
              services/myService");
5         QoSClient client = new QoSClient("John", "Anon", "
              DoeIsMe", this);
6         String soap = "<?xml version=\"1.0\" ?><S:Envelope
              xmlns:S=\"http://schemas.xmlsoap.org/soap/envelope
              /\">" +
7                 "<S:Body>Hello World</S:Body></S:Envelope>";
8         ReceiveObject recObj = null;
9         int send = 0;
10        do{
11            recObj = client.send(soap, destination);
12            send++;
13        } while (recObj.receive().endsWith("Exception") &&
              send<=retry);
14        System.out.println(recObj.receive());
15    }
16
17    @Override
18    public void unknownHostExceptionThrown(
          UnknownHostException e) {
19        //Do nothing
20    }
21    @Override
22    public void ioExceptionThrown(IOException e) {
23        //Do nothing
24    }
25    @Override
26    public void httpExceptionThrown(Exception e) {
27        //Do nothing
28    }
29    @Override
30    public void socketExceptionThrown(SocketException e) {
31        //Do nothing
32    }
33    @Override
34    public void unsupportedEncodingExceptionThrown(
35            UnsupportedEncodingException e) {
36        //Do nothing
37    }
38 }
```

B Server Setup Guide

We have altered some source code which our project is dependent upon. That is why it is quite important that anyone wanting to use our setup follow the steps below. Our ESB mediators will not work unless this is done properly.

First we have to download the ESB from wso2.org, and extract it, for this project we used version 4.0.3 found here: http://wso2.org/products/download/esb/java/4.0.3/wso2esb-4.0.3.zip

This version of the ESB uses httpcore-nio version 4.1.3, which does not support setting DiffServ in the IP header. To fix this we have to download its source code and make some modifications to it.

Now we need to retrieve the source of HTTPCore, this can be done by pasting the command in Listing:12 into a bash prompt. This will download the source compatible with WSO2 into a folder named "hc", this may take some time depending on your Internet connection.

Listing 12: Checkout HttpCore source

```
1 $ svn checkout http://svn.apache.org/repos/asf/
    httpcomponents/httpcore/tags/4.1.3/ hc
```

After the download has finished we have a working directory, of the version of HTTPCore which WSO2 ESB uses, and it is now time to apply our patch to enable support for traffic class. The instruction in Listing:13 describes how. Here "hc.diff" is a file containing our changes which svn can use to alter the working directory. The file "hc.diff" can be found in appendix G.

Listing 13: Apply HC patch

```
1 $ cd hc/
2 $ patch -p0 -i /path/to/hc.diff
```

Since we have now applied our patch we just need to compile HTTPCore in order to use it in Synapse later on. To build it we just copy-paste the commands from Listing:14 and let it compile.

Listing 14: Build HttpCore-NIO

```
1 $ cd hc/httpcore-nio/
2 $ mvn clean install
```

When the building is complete we can add the relevant jar, "hc/httpcore-nio/target/httpcore-nio-4.1.3.jar" to the WSO2 ESB. For the ESB to recognize it correctly it must have the correct name, and it must also have the correct data in the META-INF folder inside the jar. To fix this we follow the steps in Listing:15.

Listing 15: Create WSO2 compatible jars

```
1 $ cd /path/to/wso2esb/repository/components/plugins/
2 $ mkdir backup
3 $ cp httpcore-nio-4.1.3.wso2v2.jar backup/httpcore-nio
    -4.1.3.wso2v2.jar
```

```
$ mkdir build
$ cp /path/to/hc/httpcore-nio/target/httpcore-nio-4.1.3.jar
    build/httpcore-nio-4.1.3.jar
$ cp httpcore-nio-4.1.3.wso2v2.jar build/httpcore-nio-4.1.3.
    wso2v2.jar
$ cd build
$ unzip httpcore-nio-4.1.3.wso2v2.jar
$ zip -r httpcore-nio-4.1.3.jar META-INF
$ cd ..
$ cp build/httpcore-nio-4.1.3.jar httpcore-nio-4.1.3.wso2v2.
    jar
```

Java 1.6 only support setting DiffServ on IPv4, so to make java actually set the DiffServ header we must set IPv4 as the preferred IP stack. This is done by adding the command line option -Djava.net.preferIPv4Stack=true when starting the java program. This should be done with all clients as well as ESB. The ESB is usually started by running /path/to/wso2esb/bin/wso2server.sh, so we edit this file adding the line containing "-Djava.net.preferIPv4Stack=true" in Listing:16.

Listing 16: Changes made to wso2server.sh

```
while [ "$status" = "$START_EXIT_STATUS" ]
do
    $JAVACMD \
    -Xbootclasspath/a:"$CARBON_XBOOTCLASSPATH" \
    -Xms256m -Xmx512m -XX:MaxPermSize=256m \
    $JAVA_OPTS \
    -Dimpl.prefix=Carbon \
    -Dcom.sun.management.jmxremote \
    -classpath "$CARBON_CLASSPATH" \
    -Djava.endorsed.dirs="$JAVA_ENDORSED_DIRS" \
    -Djava.io.tmpdir="$CARBON_HOME/tmp" \
    -Djava.net.preferIPv4Stack=true \
    -Dwso2.server.standalone=true \
    -Dcarbon.registry.root=/ \
    -Dcarbon.xbootclasspath="$CARBON_XBOOTCLASSPATH" \
    -Djava.command="$JAVACMD" \
    -Dcarbon.home="$CARBON_HOME" \
    -Dwso2.transports.xml="$CARBON_HOME/repository/conf/mgt-
        transports.xml" \
    -Djava.util.logging.config.file="$CARBON_HOME/lib/log4j.
        properties" \
    -Dcarbon.config.dir.path="$CARBON_HOME/repository/conf"
        \
    -Dcomponents.repo="$CARBON_HOME/repository/components/
        plugins" \
    -Dcom.atomikos.icatch.file="$CARBON_HOME/lib/
        transactions.properties" \
    -Dcom.atomikos.icatch.hide_init_file_path=true \
    -Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER
        =true \
    -Dcom.sun.jndi.ldap.connect.pool.authentication=simple
        \
    -Dcom.sun.jndi.ldap.connect.pool.timeout=3000   \
```

```
    org.wso2.carbon.bootstrap.Bootstrap $*

    status=$?
done
```

Now that the ESB is set up, and supporting DiffServ, we can add our own components to it. The first thing to add is a jar-file containing all the custom mediators and the custom message store. Copy the file "no.ntnu.qos.jar" found in appendix G to the folder /path/to/wso2esb/repository/components/lib/.

Next we add a base configuration for the ESB to use, extract "synapse-configs.zip" found in G into /path/to/wso2esb/repository/deployment/server/synapse-configs/. This configuration contains setup of mediator sequences to be used, and a simple sample service endpoint/proxy configuration.

Finally, before starting the ESB, we should add the template xml-files found in "mediator-configuration.zip" in appendix G to the folder /path/to/wso2esb/.

The ESB should now be ready and can be started by running the script:

```
$ ./path/to/wso2esb/bin/wso2server.sh
```

C MobiEmu Setup Guide

In this appendix we will show you how to setup the MobiEmu framework to make it ready to run the tests from Chapter 7.

We will start by downloading NS3 which is required in order to run our tests. Just download NS3 from http://www.nsnam.org/release/ns-allinone-3.13.tar.bz2 in order to get the same version as we are using and then follow the instructions at http://www.nsnam.org/docs/release/3.13/tutorial/html/getting-started.html#building-ns-3 in order to build and let NS3 configure itself.

Since this framework is quite nice and packages up everything before a system test all we need to do is unpack one of our run folders and we should be good to go. We have included a compressed folder which contains five compressed folders where each of these contains a test which we need to unpack in order to run.

After you have unpacked one of these tests we only have to add the ESB and GlassFish to be able to run the tests.

We will assume that you have unpacked one of our tests and created a folder named "system-test" which contains the whole experiment.

In order to do the next part you will need to have set up the ESB with our changes, if you have not done so, please see appendix B

Listing 17: Copy ESB and GlassFish into System test

```
$ cd /path/to/system-test/experiments/esb/
$ cp /path/to/esb/bin/ bin
$ cp /path/to/esb/dbscripts/ dbscripts
```

```
$ cp /path/to/esb/lib/ lib
$ cp /path/to/esb/repository/ repository
$ cp /path/to/esb/samples/ samples
$ cp /path/to/esb/tmp/ tmp

$ cd ..
$ cp /path/to/glassfish/ glassfish
```

Now we just have to add the testing client.

Listing 18: Adding the test client

```
$ cd /path/to/system-test/experiments/
$ mkdir EchoClient
$ cp /path/to/EchoClientClient.jar EchoClient/
```

Now we need to just copy the file "synapse.xml" into the correct folder and we should be good to go. Copy the "synapse.xml" file from the folder you unpacked into "/path/to/system-test/experiments/esb/repository/deployment/server/synapse-configs/default/". This will ensure that the ESB is configured correctly with the same timeout as we have run with.

You may also need to change the path in "system-test-2.py" please have a look at C and change it to reflect your setup.

The last thing we need to do is just edit the settings file and ensure that the "[ns3]" section points to the right path. In listing 20 we have included our setting file, which is configured the way we set things up. Just edit the path under "[ns3]" to point to the path to NS3.

Now the setup should be completed and it should now be possible to run the framework. In order to do so just run:

Listing 19: Run MobiEmu

```
$ cd /path/to/system-test
$ sudo su
# ./run.py run
```

You will now be running one of our system tests!

The rest of this appendix will explain the different variables and testing files that we use in our setup, it is not necessary to read this section if you just want to verify our tests.

Below is the full settings files for all of our system test, the comments within it is from the MobiEmu creator, but we will try to explain them all and highlight the ones which has a real effect on our tests.

Listing 20: Setting.cfg

```
#
# This configuration file contains the parameters for the
    experiments run by run.py.
#
# When an experiment starts, scripts are executed in the
    following order (for each repetition):
#
# 1. Scripts specified in config.init_scripts are executed
```

```
7  # 2. Virtual network devices are created
8  # 3. ns-3 is started
9  # 4. One lxc-container is started for each node in the
       experiment
10 # 2. In lxc: Any modules specified in the experiment are
       executed in parallell
11 # 3. In lxc: Wait for config.initial_wait
12 # 5. In lxc: Start experiment script
13 # 6. In lxc: Wait for config.experiment_wait
14 # 7. In lxc: Wait for config.shutdown_wait / 2. If the
       experiment is still running, attempt to kill it.
15 # 8. In lxc: Stop all modules
16 #
17 # This is repeated for every repetition of the experiment.
       Note that multiple experiments may be executed
18 # by separating them by a "," in the configuration.
19 #
20 # When run.py is run without parameters it attempts to
       estimate the total time it will take to execute
21 # the current configuration.
22 #
23 # Modules and experiment-scripts are passed all parameters
       in their configuration and in the general-section as
       environment variables.
24 # Topology scripts (ns-3 simulation scripts) are passed all
       configuration parameters as parameters to the script. E.g
       . --total_nodes=xxx
25 #
26 # Output is logged to core.log and node*.log
27
28 [general]
29 # total nodes in the experiment
30 total_nodes=4
31
32 # intial random seed
33 initial_random_seed=31
34
35 #list of experiments to run. Each experiment must match a
       config section
36 experiments=enoughBandwidth
37
38 # Number of times to repeat each configuration of the
       experiment
39 repetitions=10
40
41 # Time to run the experiment (in seconds)
42 experiment_duration=600
43
44 # Time to wait initially before starting the experiment, e.g
       . for routing protocols to converge and modules to start.
45 initial_wait=10
46
47 # Time to wait after the experiment before shutting down the
       emulator
```

```
48 shutdown_wait=30
49
50 # Config for ns3 script, used to generate the topology and
       connect to the virtual devices. The name must match a
       config section.
51 topology=system-test-2
52
53 # Process priorities for experiment and simulator scripts.
54 simulator_niceness=-20
55 experiment_niceness=19
56
57 # Enable or disable debug messages. This will also output
       all commands executed by run.py
58 show_debug_messages=False
59
60 # Set to True if the lxc-containers use chroot
61 use_chroot=False
62
63 # Script to call before starting. This script is executed
       before the modules.
64 init_scripts=set_long_queues.sh
65
66 [directories]
67 main=.
68 # where to store experiments when they are done
69 results=%(main)s/results
70
71 # where to store output from experiments while the
       experiment is running
72 dumps=%(main)s/dumps
73
74 # if general.use_chroot is true, this directory will be
       bound to MobiEmu within the lxc-container, relative
75 # to chroot_rootpoints
76 chroot_bindpoint=/mobiemu
77
78 # List of chroot root mountpoints, may have wildcards.
79 chroot_rootpoints=../../chroot/node?,../../chroot/node??
80
81 # lxc working directory when using chroot. This is also
       where all output will be stored. It should be
82 # set to an empty directory relative to the chroot
       environment. These directories should also be included
83 # in the [results_archiver]-section, so that they are
       archived after each experiment.
84
85 chroot_working_dir=%(chroot_bindpoint)s/dumps/
86
87 # Path to module scripts (must be available within the lxc
       container)
88 modules=%(main)s/modules
89
90 # Path to experiment scripts (must be available within the
       lxc container)
```

```
91  experiments=%(main)s/experiments
92
93  # Path to topology scripts
94  topologies=%(main)s/topologies
95
96  # Path to configuration file templates
97  configs=%(main)s/configs
98
99  [ns3]
100 # Path to ns-3
101 path=../../ns-allinone-3.13/ns-3.13/
102
103 [source_archiver]
104 # Before each simulation all files and directories listed
        here will be archived and put together with the results
105 include=settings.cfg run.py modules experiments topologies
        configs
106 exclude=
107
108 [results_archiver]
109 # Moves the results from the given directory and stores them
         in the directory specified in directories.results
110 include=dumps/*
111
112 [system-test-2]
113 # System-test-2 see report chapter 8 section 3 under Three
        clients message sending
114 #
115 # Parameters specified here are passed to the script. In
        addition, the special parameters
116 # --nodes, --seed and --duration, are set to the number of
        nodes, the current random seed
117 # and the experiment duration, respectively.
118 # When specifying multiple values separated by a "," the
        experiment will be repeated
119 # an extra time for each value.
120 dataRate=1KBps,5KBps,10KBps,20KBps,40KBps
121 #MTU needs to be defined
122 mtu=2304
123 # The ns-3 script. Must be in directories.topolgies.
124 script=system_test_2.cc
125
126 [enoughBandwidth]
127 #Load modules within lxc prior to loading ns3, but before
        starting experiment. The modules
128 #may have a configuration section with additional parameters
         which are passed to the module as
129 #environment variables.
130 #modules=manualrouting.py,manualarp.py,tcpdump.sh
131 modules=monitoring_service.py,manualarp.py,tcpdump.sh
132 #modules=tcpdump.sh
133
134 #experiment script to run within lxc. Automatically
        terminated (SIGTERM is sent) after
```

```
#%(experiment_duration + shutdown_wait/2)
experiment=system_test_2.py

[tcpdump.sh]
# shell environment variables can be used in filter
filter=
device=eth0

[manualarp.py]
arp_mac=00:16:3e:00:01:%0.2X
arp_ip=10.0.0.%s

[ipv4_multicast_route.sh]
device=eth0
[manualrouting.py]
[monitoring_service.py]
```

The most important variables and their meaning is described below.

- total_nodes=4 - This is the total number of nodes in the testing, this variable is passed to NS3 so it can use it and setup the right amount of nodes. Since we don't have dynamic tests this should be coordinated with the NS3 topology.
- initial_random_seed=31 - This seed is passed to all scripts which use random variables. In our testing we have no random variables so this should not be of great importance.
- repetitions=10 - The number of repetitions for each test. In our setup this means 10 repetitions for each bandwidth.
- experiment_duration=600 - Total duration for the experiment, this must be coordinated with the "system-test-2.py" file you can see in Listing:21.
- dataRate=1KBps,5KBps,10KBps,20KBps,40KBps - These are the different datarates that the experiment is run with. Each of these will be repeated "repetition" number of times.
- experiment=system_test_2.py - This is the experiment in "experiments" to run, please see Listing:21 for more information.

Next we will take a look at our testing file. This file is what starts the ESB, GlassFish and each of the clients. It is run inside its own LXC and as such needs to check which node it is so as to start the right application.

Listing 21: System-test-2.py

```
#!/usr/bin/env python

from os import getenv
from subprocess import Popen
from time import sleep

node_id = int(getenv('node_id'))
path = '/home/qos/server/mobiemu/system-test-2/experiments/'
```

```
9  gfSleep = 60
10 wso2Sleep = 190
11 experimentSleep = 320
12
13 if node_id == 1:
14    #This is the ESB/GlashFish node
15    #Start GlashFish:
16    print 'Starting GlashFish'
17    gf = Popen(['{0}glassfish/bin/./startserv'.format(path)])
18    #Wait for glassfish to start
19    sleep(gfSleep)
20    #Start ESB
21    print 'Starting WSO2'
22    wso2 = Popen(['{0}esb/bin/./wso2server.sh'.format(path)])
23    #Need to sleep to wait for experiment to finish
24    sleep(wso2Sleep + experimentSleep)
25    print 'Done with experiment, terminating GlashFish and
          WSO2'
26    wso2.kill()
27    gf.terminate()
28 elif node_id == 2 or node_id == 3 or node_id == 4:
29    #This is the client node
30    #Need to wait for GF and WSO2 to start
31    print 'Starting node 3, EchoClientClient'
32    sleep(gfSleep + wso2Sleep)
33    echo = Popen(['java', '-Djava.net.preferIPv4Stack=true',
          '-jar', '{0}EchoClient/EchoClientClient.jar'.format(
          path), '{0}client_{1}.config'.format(path, node_id -
          1)])
34    sleep(experimentSleep)
35    #Check if echo has terminated,
36    #poll() returns None if it hasn't terminated
37    echo.poll()
38    if not echo.returncode:
39        #Client has most likely crashed at this point
40        #as such we need to kill it, softly with this song...
41        echo.terminate()
```

There should really be nothing to special about this file. It first starts up GlassFish and then the ESB, each client sleeps until the ESB has started and then they start sending their messages.

- path = '/home/qos/server/mobiemu/system-test-2/experiments/' - This will need to be changed to reflect where it is run from, this has to be this way because it is run with root and it needs an absolute path.

The last file we will look at is the topology file, this defines the topology in NS3 and creates Tap-Bridges which we connect the LXCs to.

Listing 22: System-test-2 Topology

```
1 /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil;
      -*- */
2 /*
```

```
3  * This program is free software; you can redistribute it and
       /or modify
4  * it under the terms of the GNU General Public License
       version 2 as
5  * published by the Free Software Foundation;
6  *
7  * This program is distributed in the hope that it will be
       useful,
8  * but WITHOUT ANY WARRANTY; without even the implied
       warranty of
9  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
       the
10 * GNU General Public License for more details.
11 *
12 * You should have received a copy of the GNU General Public
       License
13 * along with this program; if not, write to the Free
       Software
14 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
       02111-1307  USA
15 */
16
17 // Network topology
18 // //
19 // //             ESB      Client Client  Client
20 // //              |         |       |       |
21 // //              ============================
22 // //                          LAN
23 // //
24
25
26 #include <iostream>
27 #include <fstream>
28
29 #include "ns3/core-module.h"
30 #include "ns3/csma-module.h"
31 #include "ns3/point-to-point-module.h"
32 #include "ns3/internet-module.h"
33 #include "ns3/ipv4-address-helper.h"
34 #include "ns3/stats-module.h"
35 #include "ns3/tap-bridge-helper.h"
36 #include "ns3/ipv4-global-routing-helper.h"
37
38 using namespace ns3;
39 using std::stringstream;
40 using std::string;
41
42 NS_LOG_COMPONENT_DEFINE ("QoS System test 1");
43
44 void printTime(int interval)
45 {
46     Ptr<RealtimeSimulatorImpl> impl = DynamicCast<
        RealtimeSimulatorImpl>(Simulator::GetImplementation
        ());
```

```
47      Time real_time = impl->RealtimeNow();
48
49      Time sim_time = Simulator::Now();
50
51      std::cout << "drift:" << real_time.GetMilliSeconds() <<
            ":" << sim_time.GetMilliSeconds() << ":" << (
            real_time.GetMilliSeconds() - sim_time.
            GetMilliSeconds()) << std::endl;
52      Simulator::Schedule(Seconds(interval), &printTime,
            interval);
53  }
54
55  int main (int argc, char *argv[])
56  {
57      CommandLine cmd;
58      uint32_t run_time= 200, seed = 1, n = 3, mtu = 0;
59      string runID;
60      string dataRate;
61      cmd.AddValue ("duration", "Duration of simulation",
            run_time);
62      cmd.AddValue ("seed", "Seed for the Random generator",
            seed);
63      cmd.AddValue ("runID", "Identity of this run", runID);
64      cmd.AddValue("datarate", "Data rate for LAN", dataRate);
65      cmd.AddValue("nodes", "Not used in this test", n);
66      cmd.AddValue("mtu", "MTU used for TapBridges", mtu);
67
68      cmd.Parse(argc, argv);
69
70      stringstream s;
71      s << "Duration: " << run_time << ", Seed: " << seed << "
            , runID: " << runID << ", DataRate: " << dataRate;
72      std::cout << s.str() << std::endl;
73
74      SeedManager::SetSeed (seed);
75      GlobalValue::Bind ("SimulatorImplementationType",
            StringValue ("ns3::RealtimeSimulatorImpl"));
76      GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true
            ));
77
78      NodeContainer nodes;
79      nodes.Create(4);
80
81      CsmaHelper csma;
82      csma.SetChannelAttribute ("DataRate", StringValue (
            dataRate));
83
84      NetDeviceContainer csmaDevices;
85      csmaDevices = csma.Install (nodes);
86
87      TapBridgeHelper tapBridge;
88      tapBridge.SetAttribute ("Mode", StringValue ("UseLocal")
            );
89      tapBridge.SetAttribute ("Mtu", UintegerValue(mtu));
```

```
90
91      //Tap bridge setup for ESB
92      std::cout << "Adding tap bridge: tap-1\n";
93      tapBridge.SetAttribute ("DeviceName", StringValue ("tap
            -1"));
94      tapBridge.Install (nodes.Get(0), csmaDevices.Get(0));
95
96      //Tap bridge setup for client 1
97      std::cout << "Adding tap bridge: tap-2\n";
98      tapBridge.SetAttribute ("DeviceName", StringValue ("tap
            -2"));
99      tapBridge.Install (nodes.Get(1), csmaDevices.Get(1));
100
101     //Tap bridge setup for client 2
102     std::cout << "Adding tap bridge: tap-3\n";
103     tapBridge.SetAttribute ("DeviceName", StringValue ("tap
            -3"));
104     tapBridge.Install (nodes.Get(2), csmaDevices.Get(2));
105
106     //Tap bridge setup for client 3
107     std::cout << "Adding tap bridge: tap-4\n";
108     tapBridge.SetAttribute ("DeviceName", StringValue ("tap
            -4"));
109     tapBridge.Install (nodes.Get(3), csmaDevices.Get(3));
110
111
112
113     std::cout << "Starting simulation. Will run for " <<
            run_time << " seconds...\n";
114     Simulator::Schedule(Seconds(0), &printTime, 1);
115     Simulator::Stop (Seconds (run_time));
116
117     Simulator::Run();
118
119     Simulator::Destroy ();
120
121     std::cout << "Done.\n";
122 }
```

There should be very little which we could explaining here, as NS3 in itself is quite complex. Therefore we refer you to the NS3[46] documentation if you would like extend the topology.

D Result Parsing

In this apendix we will describe how to use the scripts provided to parse the results we get from testing with MobiEmu.

If tests are run with different bandwidth capacities we can use the script GroupResults.py like this:

[46] Documentation for NS3 http://www.nsnam.org/docs/release/3.13/tutorial/singlehtml/index.html

```
$ python2 GroupResults.py /path/to/results/ /path/to/where/
    you/want/results/grouped/
```

This script will move the results from the first folder to the second, and group them in subfolders with the name of their bandwidth capacities. When choosing a folder to move them to we recommend choosing a subfolder of a general results folder, this way you can use the script parseall.sh like this:

```
$ ./parseall.sh "-m -t -p" /path/to/folder/containing/the/
    folder/you/moved/results/to/ "2 3 4"
```

Here the first argument "-m -t -p" is the optional arguments used for ParseResults.py, and the last argument "2 3 4" are the nodes to parse output from.

E NS3 Problems

When trying to set up the test we wanted, illustrated in section 3.5, we encountered some strange problems which we could not solve in time for this project. Below is an outline of the problem and a possible solution sketched out by the creator of MobiEmu.

The problem seem to stem from NS3 and how it "installs" the different abilities to each node in the network. In listing 23 we have tried to illustrate the problem.

Listing 23: This code snippet does not work

```
NodeContainer net1 (esb, r);
NodeContainer net2 (r, client);
NodeContainer all (esb, r, client);
NodeContainer shortCut (esb, client);

PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue (
    constDataRate));
//pointToPoint.SetChannelAttribute ("Delay", StringValue ("2
    ms"));

NetDeviceContainer esbToRouterDevices;
esbToRouterDevices = p2p.Install (net1);

p2p.SetChannelAttribute ("DataRate", StringValue (dataRate))
    ;
//pointToPoint.SetChannelAttribute ("Delay", StringValue ("2
    ms"));

NetDeviceContainer routerToClientDevices;
routerToClientDevices = p2p.Install (net2);

TapBridgeHelper tapBridge;
tapBridge.SetAttribute ("Mode", StringValue ("UseLocal"));
tapBridge.SetAttribute ("Mtu", UintegerValue(mtu));

//Tap bridge setup for ESB
std::cout << "Adding tap bridge: tap-1\n";
```

```
25 tapBridge.SetAttribute ("DeviceName", StringValue ("tap-1"))
       ;
26 tapBridge.Install (esb, esbToRouterDevices.Get(0));
27
28 //Tap bridge setup for router
29 std::cout << "Adding tap bridge: tap-2\n";
30 tapBridge.SetAttribute ("DeviceName", StringValue ("tap-2"))
       ;
31 tapBridge.Install (r, esbToRouterDevices.Get(1));
32
33 //Tap bridge setup for router
34 std::cout << "Adding tap bridge: tap-3\n";
35 tapBridge.SetAttribute ("DeviceName", StringValue ("tap-3"))
       ;
36 tapBridge.Install (client, routerToClientDevices.Get(1));
```

When we try to run this code inside each LXC we do not get the communication that we expect to see. What happens instead is that only the nodes coming from the same "NodeContainer" get to talk to each other. We can then induce communication through the two other nodes if we change which "NodeContainer" we use when we install the tap bridges. The strange thing is that there is not much which would indicate this inside the source code.

The solution which was illustrated to us by the creator of MobiEmu was to create all the LXCs with two network connections, then inside NS3 create twice as many tap bridges and connect all this up. We did not have time do this as this would require much more knowledge about LXC, we would have to alter MobiEmu substantially to create all the extra tap bridges and we would need to change the NS3 scripts. In addition, this solution was not at all guaranteed to work which would mean that we could put down many hours with out any results. We mentioned this in the testing chapter (ref:7.1.1) that we decided together with the customer that this would take to long time with to little time left in the project.

F Work Breakdown Structure

WBS to be completly implemented later, it is also attached under (G)

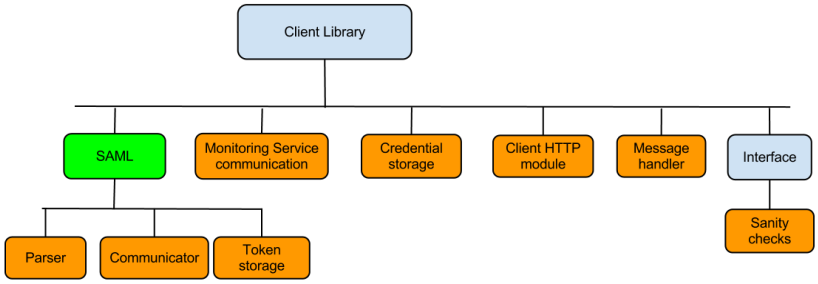


Figure 32: WBS-Client
The work break down structure for the client library.

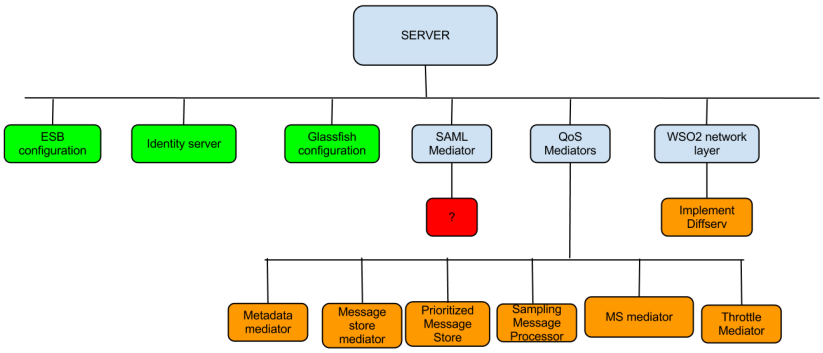


Figure 33: WBS-Server
The work break down structure for the server.

G File Attachments

Files are found in: fileAttachments.zip
Or at: https://github.com/magnuskiro/it2901/ path/to/raw/file

- bachelor-QoS.pdf

  Introduction to the assignment and Quality of Service(QoS) support for Web services in military networks.

- risklist.pdf

  The full risk list attached in it's full glory.

- gantt.html

  Our gantt chart in it's full form.

- ClientClassDiagram.png

  The class diagram describing our classes and their metadata.

- EchoClientClient.jar

  The client we have used for our testing.

- Client.config

  An example configuration for the Echo client.

- TestClient.java

  The source code for the Echo client.

- EchoServiceLargeReply.war

  A simple service which echoes responses back and pads them with 10Kb of text.

- hc.diff

  A diff file which can be applied to the HTTPCore source to apply the changes for DiffServ support.

- no.ntnu.qos.jar

  The server side, neatly packed into a jar ready for deployment in an ESB near you.

- synapse-configs.zip

  The commented Synapse configuration files, the values may not be correct, but the comments are.

- mediator-configuration.zip

  The necessary configuration files for the ESB using our test setup.

- Test-Case-1.tar.gz

  Necessary files for Test Case 1.

- Test-Case-2.tar.gz

  Necessary files for Test Case 2.

- Test-Case-3.tar.gz

  Necessary files for Test Case 3.

- Test-Case-4.tar.gz

  Necessary files for Test Case 4.

- Test-Case-5.tar.gz

  Necessary files for Test Case 5.

- Test-Case-6.tar.gz

  Necessary files for Test Case 6.

Glossary

LaTeX A document preparation system for the TeXtypesetting program http://www.latex-project.org/. 24

Agile methods A group of software development methodologies based on iterative and incremental development http://en.wikipedia.org/wiki/Agile_software_development. 22

Apache Synapse A lightweight and high-performance Enterprise Service Bus http://synapse.apache.org/. 39

Bandwidth Available or consumed data communication resources https://secure.wikimedia.org/wikipedia/en/wiki/Bandwidth_(computing). 4

COTS Commercially available Off-The-Shelf often used to talk about services which the customer wants to use server side https://secure.wikimedia.org/wikipedia/en/wiki/Commercial_off-the-shelf. 16

DiffServ Differentiated services, a field in the IPv4 header http://www.networksorcery.com/enp/rfc/rfc2474.txt. 4

Gantt Chart A type of bar chart that illustrates a project schedule http://en.wikipedia.org/wiki/Gantt. 22

Git A free and open source, distributed version control system http://www.git-scm.com. 24

GitHub A web-based hosting service for software development projects that use the Git version control system http://www.github.com. 24

GlassFish An application server written in Java http://glassfish.java.net/. 5

HTTP Hypertext Transfer Protocol. The foundation of data communication on the World Wide Web http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/HTTP/AsImplemented.html. 16

HTTPComponents A toolset of low level Java components focused on HTTP and associating protocols http://hc.apache.org/. 27

IDE Integrated Development Environment. A software application that provides facilities for software development such as source code editor, compiler etc.. 24

Identity Server http://wso2.com/products/identity-server/. 9

IP address A numerical label assigned to each device connected to the Internet. 42

Java Coding Conventions http://www.oracle.com/technetwork/java/codeconv-138413.html. 24

JUnit A testing framework for the Java programming language http://junit.org/. 24

LXC Linux Containers, http://lxc.sourceforge.net/. 54, 81

Mediator A component in WSO2 ESB which can be used to work on incoming or outgoing messages that passes through the ESB http://synapse.apache.org/Synapse_QuickStart.html. 9

Message SOAP message https://secure.wikimedia.org/wikipedia/en/wiki/SOAP#Message_format. 13

Middleware In the report middleware will refer to the program we are making. Other distinctions should be made explicitly in the text.. 4

MobiEmu Mobility Emulator, A framework for emulating mobile ad-hoc networks with Linux containers and ns-3.. 80

Monitoring Service Monitoring Service, a service that provides bandwidth monitoring, running on the same server as the Tactical Router.. 10

NS3 A network simulator http://www.nsnam.org/. 6

OpenSAML A set of open source C++ Java libraries to support developers working with SAML. https://wiki.shibboleth.net/confluence/display/OpenSAML/Home/. 10

Packet IP packet refers to the format to which a data transmitted over the IP protocol has been formatted tohttp://en.wikipedia.org/wiki/IPv4#Packet_structure. 4

Packet sniffer description. 12

Pcap pcap is short for Packet capture which in our text this usually refers to a program which captures the traffic on a given socket. https://secure.wikimedia.org/wikipedia/en/wiki/Pcap. 14

Proxy A proxy server is a server that acts as an intermediary for requests from clients seeking resources from other servers http://en.wikipedia.org/wiki/Proxy_server. 9

Quality of Service Quality of Service refers to several related aspects of telephony and computer networks that allow the transport of traffic with special requirements http://en.wikipedia.org/wiki/Quality_of_service. 4

SAML Security Assertion Markup Language https://secure.wikimedia.org/wikipedia/en/wiki/SAML. 4

Scrum An agile software development methodology http://en.wikipedia.org/wiki/Scrum_(development). 21

SOAP A lightweight protocol intended for exchanging structured information in the implementation of web services in computer networks http://www.w3.org/TR/soap12-part1/#intro. 4, 5

Subversion Subversion exists to be universally recognized and adopted as an open-source, centralized version control system characterized by its reliability as a safe haven for valuable data; the simplicity of its model and usage; and its ability to support the needs of a wide variety of users and projects, from individuals to large-scale enterprise operations. https://subversion.apache.org/. 24

Tactical router A Multi-topology router used in military networks. 4

Token A SAML token from some form of identity server, possibly with additional meta data.. 9

TOS Type of Service, a field in the IPv4 header, now obsolete and replaced by diffserv http://en.wikipedia.org/wiki/Type_of_Service. 4

Waterfall model A sequential design process often used in software development, in which development is supposed to proceed linearly through the phases of requirements analysis, design, implementation etc http://en.wikipedia.org/wiki/Waterfall_development. 17, 22

WBS Work Breakdown Structure. An oriented decomposition of a project into smaller components http://en.wikipedia.org/wiki/Work_breakdown_structure. 22

Web Service A software system designed to support interoperable machine-to-machine interaction over a network http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#soapmessage. 4

WS-Security An extension to SOAP to apply security to web services. 5

WSO2 ESB An Enterprise Service Bus built on top of Apache Synapse. http://wso2.com/products/enterprise-service-bus/. 5

XACML eXtensible Access Control Markup Language https://secure.wikimedia.org/wikipedia/en/wiki/Xacml. 5

XML eXtensible Markup Language. A markup language defining a set of rules for encoding documents in a format readable for both humans and machines. http://www.w3.org/TR/REC-xml/. 27

XP Extreme programming is a type of agile software development http://en.wikipedia.org/wiki/Extreme_programming_practices. 21

References

[1] Frank Trethan Johnsen, Trude Hafsøe, Mariann Hauge (FFI), and Øyvind Kolbu (University of Oslo). Cross-layer quality of service based admission control for web services. test.

H Attachments

This appendix is a compilation of all our additional documents from the project process. This includes weekly reports, schedules and activity plans among other thing.

H.1 Risk List

H.2 Weekly Reports

H.3 Activity Plans

H.4 Schedules