# Quality of Service - FFI
# IT2901 - Group 7
# Preliminary Report

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

February 1, 2012

**Abstract**

This is the paper's abstract ...

# Contents

# List of Figures

# List of Tables

# 1 Project Introduction

This report is intended to guide you, through our work, decisions and results. This report will detail how we planned the project, how we designed the architecture and how we tested and evaluated the result. Hopefully by the end of this report you will understand how and why we made the decisions that we did, and additionally we hope you will enjoy or work in the same way that we have enjoyed working on it. We have intended for the results to be both verifiable and interesting for the right audience and we hope our customer will be pleased with it. If we have done this report right you will, by the end, be able to take an informed decision about how and why you should implement quality of service in networks were bandwidth is a scarce resources.

# 2 Task Description and Requirements

## 2.1 Task Description

### 2.1.1 Abstract

[High level desscription of the task.]

Our task is to provide a quality of service (QoS) layer to web services for use in military tactical networks. These networks tend to have severely limited bandwidth, and our QoS-layer must prioritise between different messages, of varying importance, that clients and services want to send. Our middleware 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.

### 2.1.2 Description

[Description of the task at hand.]

Our assignment is to create a Java application which will function as a middleware layer between web services, and clients trying to use these services. The middleware needs to process SOAP messages, which is the communication protocol for most web services, in order to be able to do its task. On the client side, the middleware needs to process messages and understand SAML 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 middleware needs to be able to modify the TOS/Diffserv IP packet header in order for the tactical router to prioritize correctly. Currently NATO has just defined one class, BULK, which is to be used with web services, but this may change in the future and our middleware should handle this upcoming change gracefully.

In addition to this, the middleware needs to be able to retrieve the available bandwidth 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 we have for implementation, like SAML, XACML, WS-Security and WSO2 ESB. In addition to this, our middleware needs to work with GlassFish, as that is the application server the customer uses.

## 2.2 Requirements

- Written in java
- High priority messages must arrive, even at the cost of dropping lower priority messages.
- Use standards where they can be used

  SAML

  Diffserv

  XACML

  WS-Security
- Test thoroughly

  Use NS3 for testing
- Extensive documentation

- Use metadata to determine priority
- GlassFish must be supported as the application server
- Must be able to set priority on network layer packets

  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.

- There are no requirements on resource usage, but we should try to keep it lightweight.

  The customer has only said that we can expect the product to be used on a standard laptop with full Java support

# 3 Project Management

How we organize ourselves in the group and work together.

## 3.1 Team Organization

[The team structure and roles.]

We already know each other coming into the project so we've chosen a flat organisational structure, 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.

Since 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 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.

The customer does not require any prototypes along the way, just a working prototype by the end of the project, so the deadlines we have set for the alpha and beta, are self-imposed.

### 3.2 Risk Assesment

### 3.3 Process Evaluation

## 4 Project Methodoloogy

### 4.1 Project Organization

[How we organize the project]

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.

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.

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.

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.

As the project progresses we'll be switching to a more agile development method, so as to allow for iterative development and facilitate for any necessary changes that may turn up as code is produced. Agile also allows for a far flatter organisational structure, which we believe will greatly help cooperation within the team.

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 two prototypes, an alfa and a beta version, to the customer before the final delivery in May.

We will not be able to have face to face meetings with the customer, but we will have weekly online meetings with them 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. Since we have some challenges in the fact that the customer is in Oslo, we decided that the weekly meetings will be held over Skype.

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 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 also can use to evaluate the product.

We will use Git and GitHub 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, we decided against this as Git gives us more options to develop code which will not greatly effect other parts of the codebase before we decide to integrate it. To this extent we have decided that as much as possible we should take advantage of Git's built in support for 'branches'. 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, which we prefer over standard word processors.

Each of us is free to choose his own IDE 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.

### 4.2 Tools

[The tools that we will use in the project.]

Since we were so free to chose 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 unknown at present time
- WSO2 ESB 4.0.3

## 5 Prestudy

## 6 Design

### 6.1 Server side Architecture

[The architecture of the server side module we are to create.]

The server side architecture consists of several components, the WSO2 ESB, the WSO2 Identity Server, the Tactical Router and the GlassFish server. All of these components are already available, so what we will have to make is mediators 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.

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

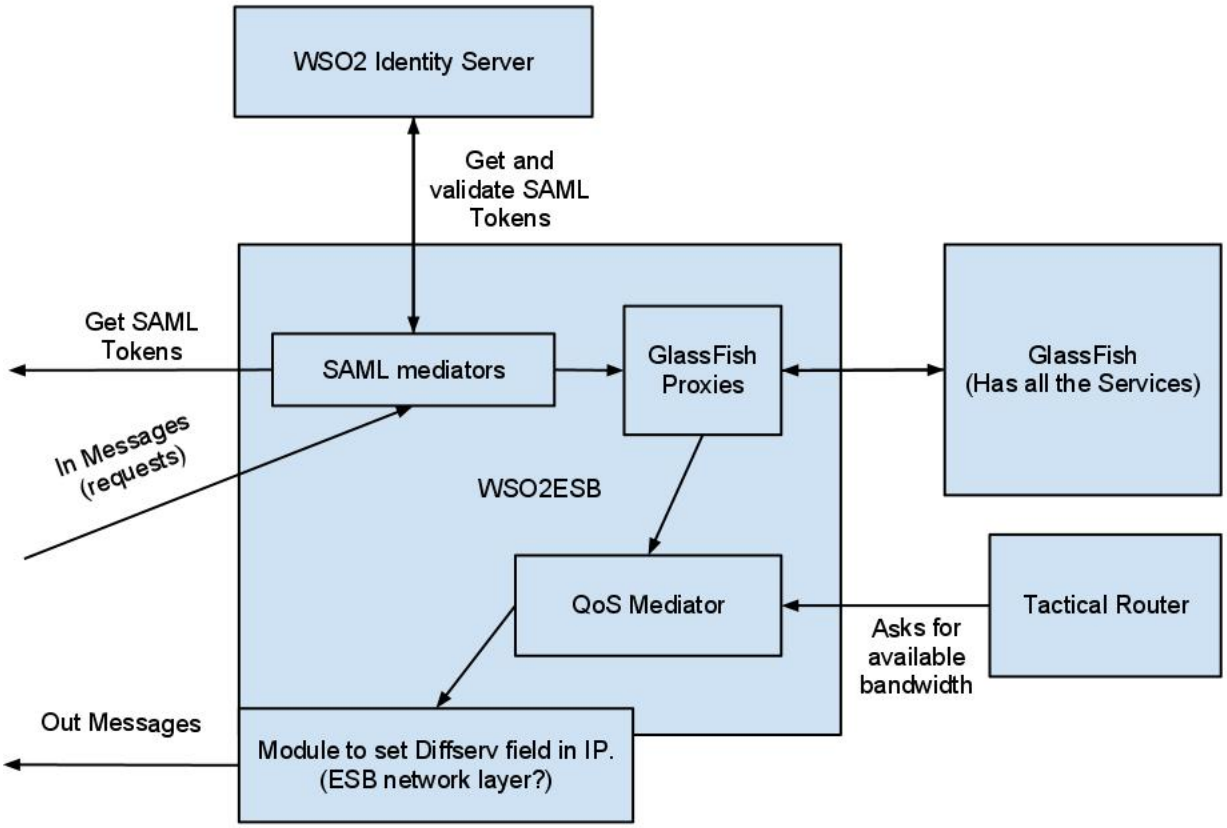


Figure 1: The serverside architecture

mediator, this mediator contacts the Identity Server to validate the clients ID-token. 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, otherwise it is dropped. The ESB acting as a proxy will then send the request along to the requested service on the GlassFish server.

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. 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 Tactical Router is contacted for

bandwidth information, 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 before the message is sent to the client. 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.

### 6.2 Client side Architecture

## 7 Implementation

## 8 Testing

## 9 Results

## 10 Conclusion

### 10.1 Future Work

## 11 Project Evaluation

## A Techical Glossary

The terms bellow are taken to be by them self or with no further details in the surrounding text.

| **Term** | **Our Interpretation** |
|---|---|
| Broker | • Our middleware layer works as a QoS broker for services and clients<br>• Broker as referred to by the report given to us from FFI<br>  ○ In the report this was a centralized 'server' of sorts which gathered Bandwidth data from tactical routers. |
| Packet | IP packet[1] |
| Message | SOAP message[2] |
| Tactical Router | Router used in military networks |
| GlassFish | Application server written in Java[3] |
| WSO2 ESB | An Enterprise Service Bus built on top of Apache Synapse[4] |
| SAML | Security Assertion Markup Language[5] |
| XACML | eXtensible Access Control Markup Language[6] |
| Bandwidth | Available or consumed data communication resources[7] |
| Middleware | • In most reports middleware will refer to the program we are making.<br>• Other distinctions should be made explicitly in the text. |
| COTS | Commercially available Off-The-Shelf often used to talk about services which the customer wants to use server side[8] |
| mediator | Component (Message mediator) in WSO2 ESB[9] |

[1] https://secure.wikimedia.org/wikipedia/en/wiki/Transmission_Control_Protocol#TCP_segment_structure

[2] https://secure.wikimedia.org/wikipedia/en/wiki/SOAP#Message_format

[3] http://glassfish.java.net/

[4] http://wso2.com/products/enterprise-service-bus/

[5] https://secure.wikimedia.org/wikipedia/en/wiki/SAML

[6] https://secure.wikimedia.org/wikipedia/en/wiki/Xacml

[7] https://secure.wikimedia.org/wikipedia/en/wiki/Bandwidth_%28computing%29

[8] https://secure.wikimedia.org/wikipedia/en/wiki/Commercial_off-the-shelf

[9] http://synapse.apache.org/Synapse_QuickStart.html

# B File Attachments

 Risk List  Work breakdown structure
 Paperclip  Graph  PushPin  Tag

# References


[1] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

[2] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.