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

29. februar 2012

Sammendrag

This is the paper's abstract ...

Innhold

Figurer

Tabeller

1 Project Introduction

This is a preliminary report documenting our progress so far in the course IT2901 - Informatikk Prosjektarbeid II. It will give a description of the problem we were presented with, how we have planned the work for the weeks ahead, 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.

2 Task Description and Requirements

Our task is to provide a Quality of Service (QoS) layer to web services for use in military tactical networks. As these networks tend to have severely limited bandwidth, and our QoS-layer must therefore 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 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

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

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.

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.

- 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

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

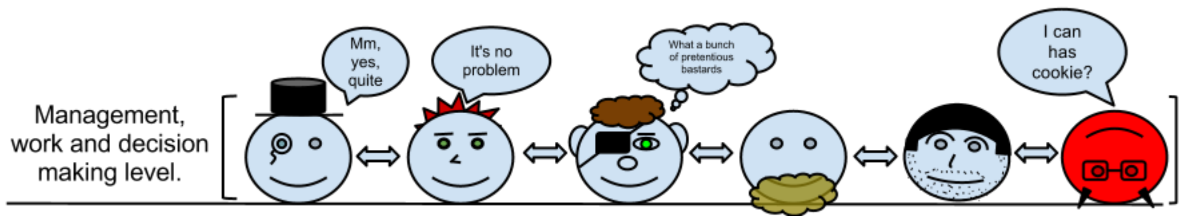
3.1 Team Organization

[The team structure and roles.]

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 and have shifted our focus accordingly over to team communication.

3.1.1 Team Structure

We already know each other coming into the project so we have 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. [1]



Figur 1: 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, 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, and therefore we have to keep people focused on the task they have taken. 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 much time getting up to date at the beginning of every week, which in turn wouldn't yield any measurable gains.

We chose to drop responsibility areas and roles. Instead we look at tasks. We might have tasks representing a responsibility, but it is still a task with a person assigned to the task. This makes the team more flexible. We don't run the risk of a key member of our team becoming ill and the rest of the group don't have a clue as to whats left to do on that particular responsibility area. Instead we say 'that person is ill, and we have to do this task by Monday then I'll do it'. And the task gets done.

[1] 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. Wikipedia: Flat Ogranization Structure

The problem we encounter and discharge is the eventuality of a person becoming ill or incapacitated and at the same time having key knowledge to our project. This seemingly huge problem is not really a problem as we document a lot. So the knowledge of a person is at all times documented, to such a degree that if a person is incapacitated the other team members can pick up right where he left off, thereby expelling the problem altogether.

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 prioritise 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 persons 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 persons 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.

3.1.2 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, 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.

3.2 Risk Assesment

3.3 Process Evaluation

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

3.4 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:	246666666666667		Actuallwork per resource:	21.4166666666667		
		Plan				Follow- up			
Nr	Work package	Activity	Resource	Planned Work(hrs)	Start	Finish	Actual Work(hrs)	Status(%)	Comment
1	Client Library	Sequence diagram	R2	24	20.02.12	21.02.12	10.5	100%	Boy didwe 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 personwas 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 thanwe expected because there were two persons working on it.
16	Server	Updateserver 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

Figur 2: Example Activity plan

The activity plans(Fig:2) 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.

As we can se in the weekly report (Fig:3) the staus report ha a standard setup. We created a template early in the project so that we did not have to redo the work later. In the prosess of creating the template we put some thought in to it so that we would get a template that would work throughout the project without further changes.

The activity plan has a template which we clone every week to create the plan for next week. We update it

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 how to 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 how to 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.

Figur 3: Status report example

4 Development Methodology

We did not follow any established development methodology, such as Scrum or XP, as this project required more planning and configuration of existing solutions, than actual coding. We therefore chose a mix of waterfall and agile methods, 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.

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.

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.

4.1 Project Organization

[How we organize the project]

We have devided the project tasks in to work packages. These packages are represented in a work breakdown structure(WBS) (B). The timeplan for the project is represented in a Gantt diagram (Fig.4). The figure is part of our full gantt diagram. As the full diagram cannot be included nicely in the report we have attached it as an html document (C).

4.2 Software project life cycle

[The development cycle. Scrum and Agile, why and how.]

For our project life cycle we chose agile. Originally we started out with the intention of using Scrum[2] and scrum only. That intention 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[3]. Very early in the project we expected that we could begin coding and prototyping very early. 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 as you already know what you are supposed to do and the major part of the task is to make the required functionality. When the functionality has to be designed and researched extensively scrum becomes

[2] Scrum is a form of agile project management [Wikipedia: Scrum]

[3] Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams [Wikipedia: Agile Software Development]

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	1[illegible]d 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					

Figur 4: Part of our Gantt diagram
Se full diagram in attachments: (C)

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[4]. Individuals and interactions are strongly connected with the organization of our team (3.1). 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

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

customer every week and we send emails back and forth to iron out the bumps of our product. The frequent communication 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.

4.3 System Technology

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

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 affect other parts of the code base 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[5] 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.

[5]Integrated development environment [Wikipedia: IDE]

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 unknown at present time
- WSO2 ESB 4.0.3

5 Prestudy

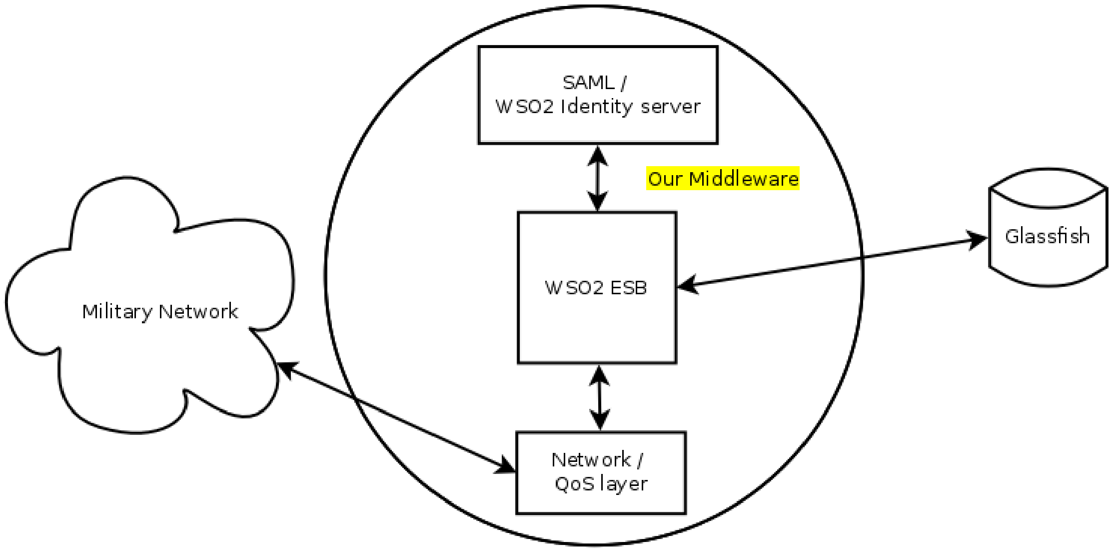
This project is one that requires quite a lot of prestudy before we can begin coding or even designing the architecture. Since the customer wanted us to implement existing technologies, such as Glassfish, WSO2, SAML etc. we need to spend some time researching those technologies to figure out what to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

5.1 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

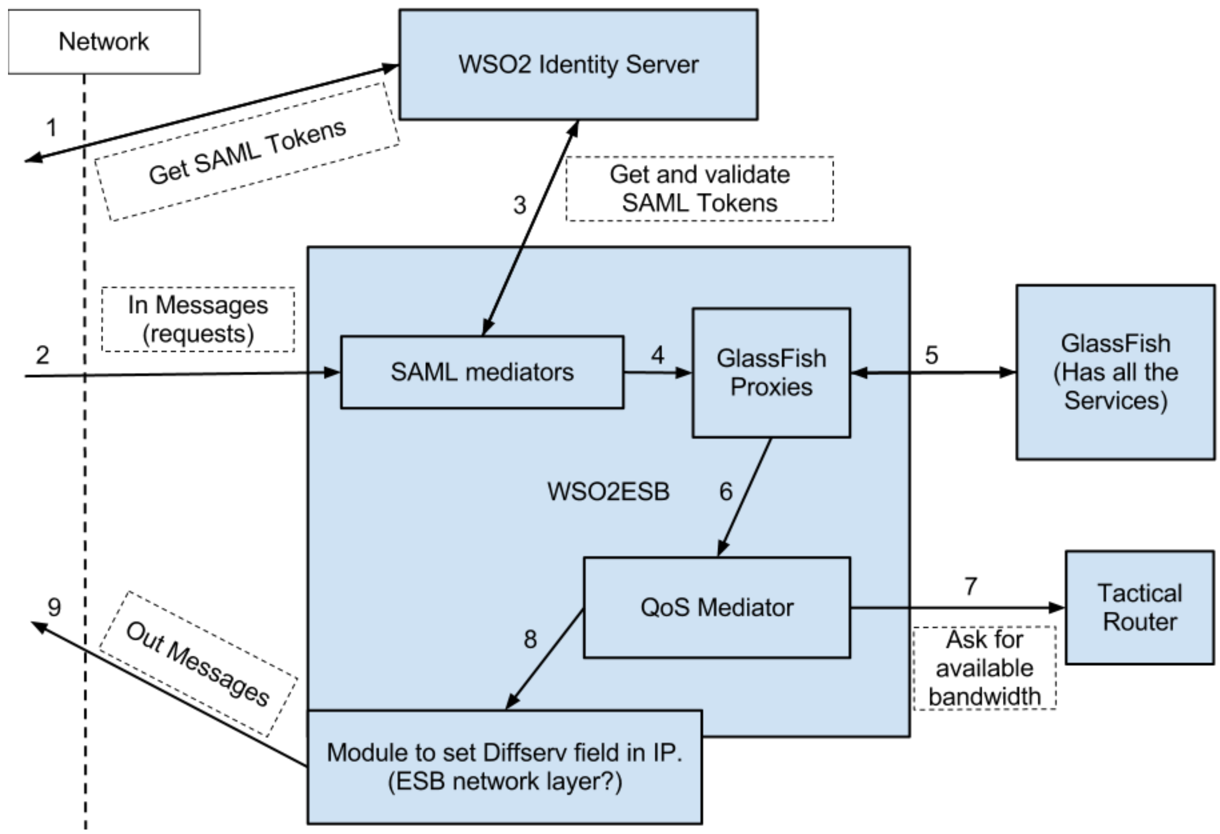


Figur 5: Bird view of the thought of architecture.

a proxy (Fig.6-1). 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.6-2), this mediator contacts the Identity Server to validate the clients ID-token (Fig.6-3). 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.6-4), 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.6-5).

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.6-6). 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 Tactical Router is contacted for bandwidth information (Fig.6-7), 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.6-8) before the message is sent to the client (Fig.6-9). This field is used by the routers in the network to prioritize packet sending. This step is quite important to



Figur 6: The Server side Architecture

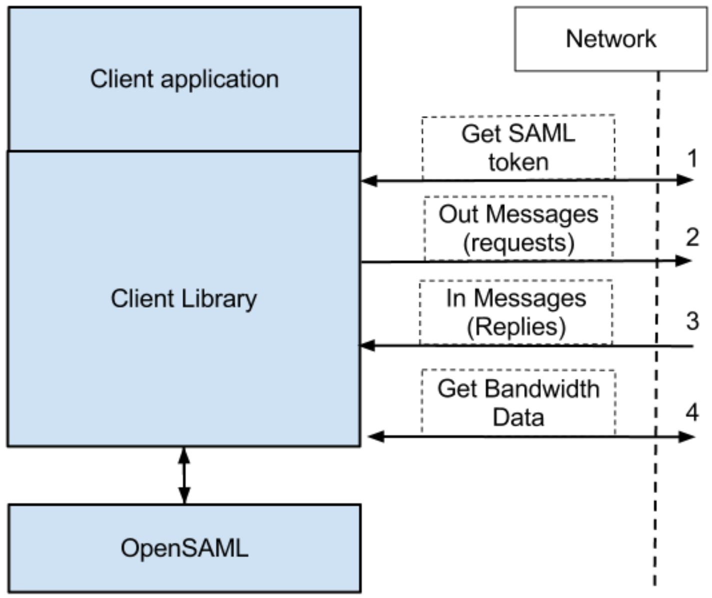
the whole procedure as this is one of the few requirements the customer has given us, as such this step can not be dropped from the final product.

5.2 Client side Architecture

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

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

Before the client library can ask for the data the client needs to get a SAML authentication token from the identity server (Fig.7-1). The communication here will most likely be handled by our library, but the SAML



Figur 7: The Client side Architecture

packages will be created and analyzed by the OpenSAML library.

The client library then sends the request from the client to the server (Fig.7-2), 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.7-3).

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 quite 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.7-4).

5.3 Alternative solutions

[Solutions that we have, thought about but discarded.]

Since this is only the preliminary report we have not yet decided on a final solution to the task at hand. But we have taken some decisions which will influence alternative solutions and will lock out some possible avenues of research.

Since you can see an outline to the architecture in (5) we will instead use this section to outline some of the alternative architectures which we looked at, but rejected for the moment.

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

6 Design

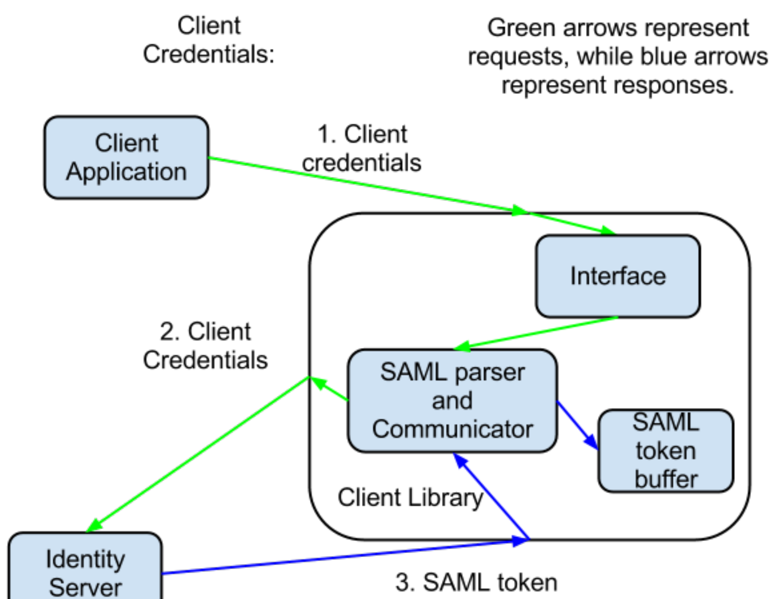
[The actual design of our implementation.]

6.1 Client Side

[The client side design.]

6.1.1 Data Flow

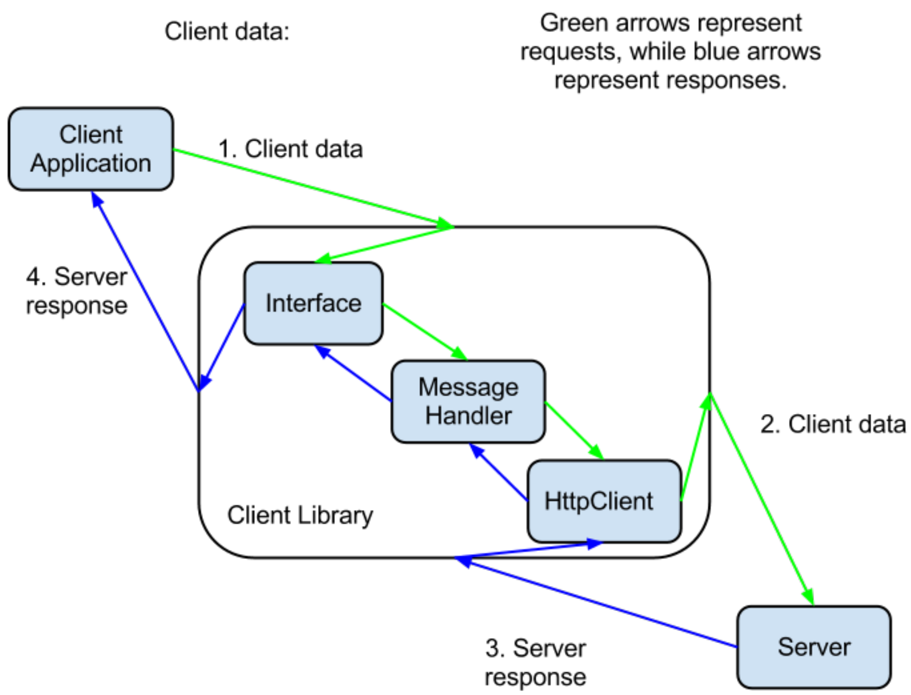
[how the data moves through the client.]



Figur 8: Client Data Flow
How data flows on the client side.

6.2 Server Side Design

[The server side design specifics.]



Figur 9: Client Credentials Flow
The flow of the credentials to and in the client side.

6.2.1 SAML Authentication

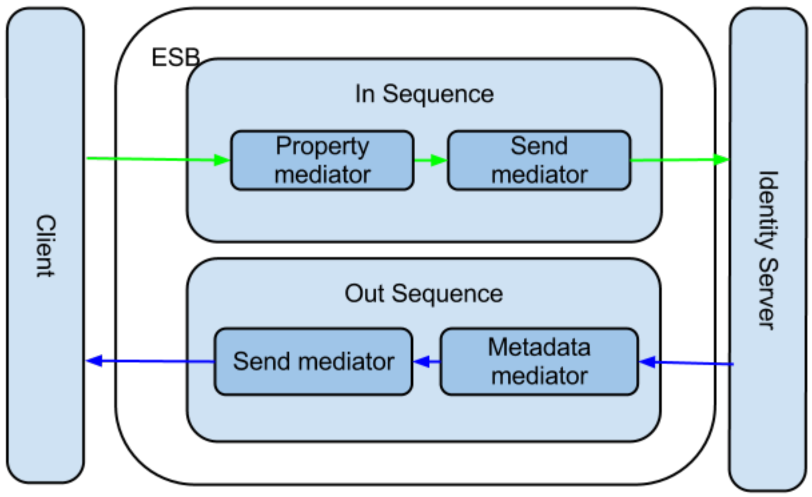
[The SAML authentication flow. How the request and authentication actualle happens.]

6.2.2 Server Data Flow

[How the data moves on the server side of our application.]

7 Implementation

[The specifics of the implementation of the sollution.]



Figur 10: SAML Authentication Flow

8 Testing

[The testing setup and suit. The testing method and how we did the testing.]

9 Results

[A thurough presentation of the results of the project. The test results and other interesting findings for this project.]

10 Conclusion

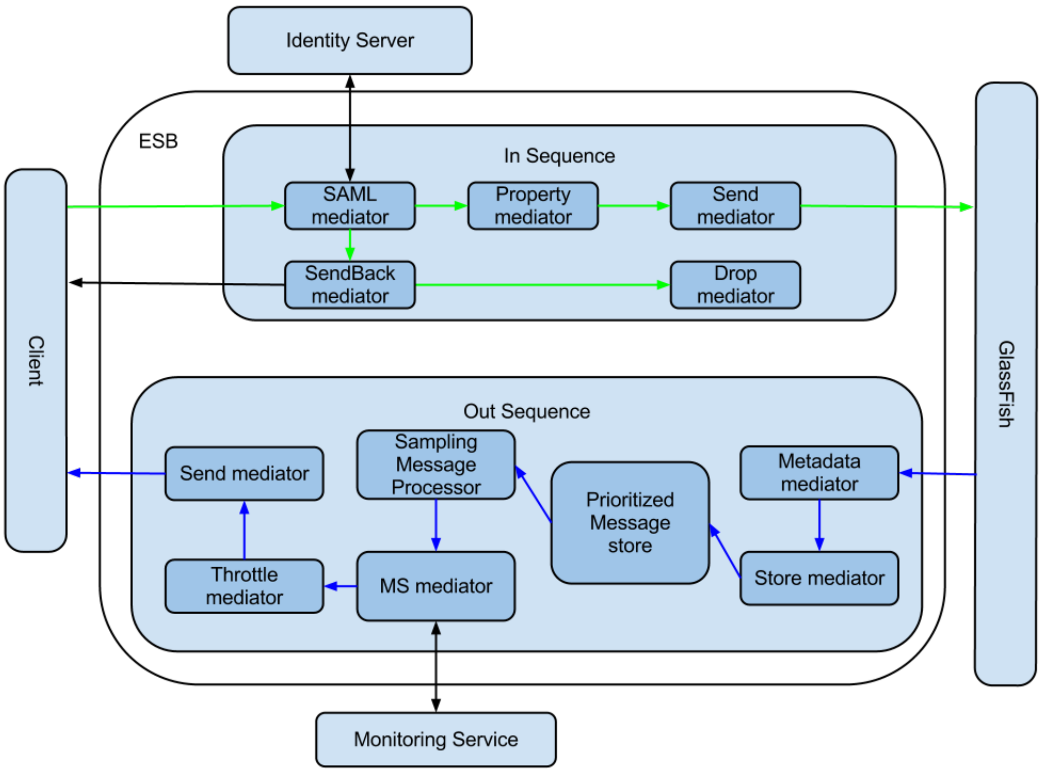
[The summarised findings of the project and presentation of the key findings.]

10.1 Project accomplishments

[Did we reach the goal?]

10.2 Future Work

[What we would have to do in the future to complete or continue the this work.]



Figur 11: Server Data Flow

11 Project Evaluation

[This is the section where we evaluate the project and the prosess that we have had throughout.]

A Techical Glossary

Bandwidth

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

Broker

Our middleware layer works as a QoS broker for services and clients
Broker as referred to by the report given to us from FFI: a centralized

'server' of sorts which gathered Bandwidth data from tactical routers.

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

Credentials
: User-supplied credentials in the form of a username, password, role tripple.

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

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

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

GlassFish
: Application server written in Java
http://glassfish.java.net/

IDE
: Integrated Development Environment

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

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

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

LaTeX

A document preparation system
http://www.latex-project.org/

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

Message

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

Message Context

Component in the ESB, contains the message, as well as all information about it, including network sockets.
http://synapse.apache.org/apidocs/org/apache/synapse/MessageContext.html

Middleware

In most reports middleware will refer to the program we are making. Other distinctions should be made explicitly in the text.

MS

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

NS3

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

OpenSAML

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

Packet

IP packet
https://secure.wikimedia.org/wikipedia/en/wiki/Transmission_Control_Protocol#TCP_segment_structure

QoS
: Quality of Service refers to several related aspects of telephony and computer networks that allow the transport of traffic with special requirements

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

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

Tactical Router
: Router used in military networks, Multi-Topology Router(MTR)

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

TOS
: Type of service, a field in the IPv4 header, now obsolete and replaced by diffserv.

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

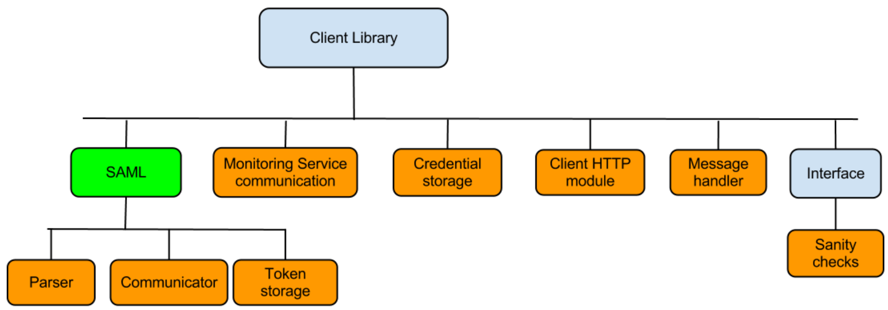
WS-Security
: An extension to SOAP to apply security to web services

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

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

B Work Breakdown Structure

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



Figur 12: WBS-Client
The work break down structure for the client.

C File Attachments

The files can be unpacked using the pdftk program in Linux with the command: pdftk filetest.pdf unpack_files

Risk List

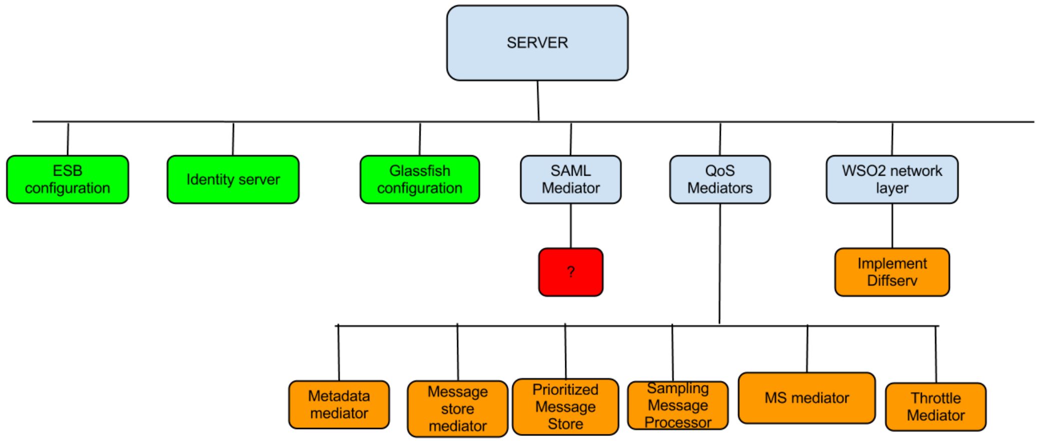
Gantt Diagram

Referanser

[1] Cross-layer Quality of Service Based Admission Control for Web Services (*Soa-qos-pdf*)

Frank T. Johnsen, Trude Hafsøe, Mariann Hauge, Norwegian Defence Research Establishment (FFI)

Øyvind Kolbu, University of Oslo, Norway



Figur 13: WBS-Server
The work break down structure for the server.

[2] Pervasive Web Services Discovery and Invocation in Military Networks, (*FFI-rapport 2011/00257*)

Frank T. Johnsen, Norwegian Defence Research Establishment (FFI)
4. January 2011