Resource Management for High Quality IP Services and IPTV

Johan André
c00jae@cs.umu.se

April 28, 2006
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Jerry Eriksson
Examiner: Per Lindström

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN

Abstract

This thesis presents an overview on resource reservations using different Quality of Service techniques in networks offering high quality IP services. The thesis also identifies the basic information to be included in a Service Level Specification (SLS), (“scope”, “performance guarantees”, “service schedule” etc.) when doing IP service offerings over IP networks. The splitting process of SLSs is also taken into account when the negotiation of a SLS spans multiple domains. The SLS must then be divided and shared between the domains to achieve SLS awareness for a chain of actors.

The result of this thesis is a software implementation of a resource management system for doing advance reservations for IPTV. The theory behind this system is inspired by the CADENUS architecture. The development was done using open source software, with web services created in Java that runs on the Linux operating system.

The thesis shows that web services using SOAP can be used to achieve connectivity between resource management components when offering IPTV services. The thesis also shows that resource management systems based on many components with different responsibilities is scaleable when new network technologies arrives.

Contents

List of Figures

Chapter 1

Introduction

A master thesis in Computing Science is presented in this report. This chapter gives an overview over the structure of the thesis, a background and a thesis description.

1.1 Background

In a world with faster ways to communicate and send information, new applications and services appear frequently. Internet has taken a more important role in transporting information, not only e-mails and web pages but also demanding high quality IP services like Voice over IP and IPTV.

Internet and the Internet Protocol (IP) technology were originally constructed to be "best effort" [9], this means that Internet makes its best effort to move data from sender to receiver as quickly as possible without any promises and guarantees. Therefore many IP services need to have Quality of Service (QoS) guarantees, so that end-users can rely on them and be willing to pay for the service. Different techniques for doing QoS are available to handle these guarantees and new solutions are constantly being developed.

In order to offer high quality premium IP services like IPTV between various actors[1] you need more than just QoS techniques. Negotiations must also be done between these actors to automatically provide the end-users with a final QoS guarantee. The CADENUS project, funded by the European Commission 5th framework IST program [8], is one of many inspiring architectures that can help us solve these problems.

The telecommunication industry is constantly expanding the capacity in mobile networks around the world. With faster mobile networks we can bring Internet and high quality IP services to the consumers wireless. QoS guarantees and negotiation between different actors are therefore of interest for many companies and industries around the world.

[1] Customers, service providers and network providers etc.

1.2 Thesis Description

The task was to present an overview of a resource management system inspired by the CADENUS architecture and to implement a working prototype to show that resource management works with high quality IP services.

The thesis was written and accomplished at Ericsson AB, the world-leading supplier in telecommunications [6].

1.3 Structure of the Thesis

This section gives an overview of the thesis structure and consists of descriptions for each chapter.

Chapter 2 describes the research assignment and the theory behind.

Chapter 3 describes the methods used when the work was carried out.

Chapter 4 gives an overview over different techniques used during this project.

Chapter 5 is the theory chapter and describes how resource management can be done.

Chapter 6 gives an overview over the final prototype and how it was developed.

Chapter 7 describes the prototype in action and the final result of this thesis.

Chapter 8 gives the conclusions.

Chapter 9 is the acknowledgement chapter.

The rest of this thesis consists of a list of acronyms, references and a complete installation guide for the final prototype.

Chapter 2

Problem Description

This chapter describes what this thesis is about and how it correlates to previous work at Ericsson. In order to better understand the research assignment a theory section about the CADENUS architecture is first presented.

2.1 CADENUS Architecture

This section describes the CADENUS architecture and is based on facts from [1] [9] [11] [12].

Internet is becoming faster and new applications using its capabilities constantly arise. Many of the applications involves operators providing high quality services to customers, e.g. Video on demand or IPTV. With these more advanced and sophisticated services QoS may be important. Internet and the IP service model can't offer QoS just by itself, as must rely on QoS techniques in the underlying network to provide this.

This may be a problem for services that demands guarantees in some way, e.g. minimum delay when using Voice over IP or minimum throughput when using IPTV. For instance, if a person wants to use Internet to watch IPTV he will not accept and pay for the service if there are disruptions due to lost bandwidth. There is QoS techniques that can help us overcome this problem with policing and scheduling, e.g. DiffServ, IntServ, RSVP and MPLS[2]. But in order to build larger resource reservation systems world-wide we need something cleverer. If we for example want to watch IPTV from USA when we are located in Sweden, the video-stream will cross many domains and networks along the way. Each single network in the domains may use different techniques for offering QoS and some may not even offer QoS at all. How do we handle this and put them all together to a final offering?

The CADENUS architecture is one of many interesting projects that try to solve this problem. It is a highly modular and component-based system with so-called "mediators" that has clearly identified roles and responsibilities. In order to make resource reservations over many types of networks, the CADENUS system relies on the Resource Mediator (RM) and the Network Controller (NC) to do this. Figure 2.1 illustrates how the CADENUS architecture looks like and how the different components relate to each other.

[2] See chapter 4.4 for more information about QoS techniques.

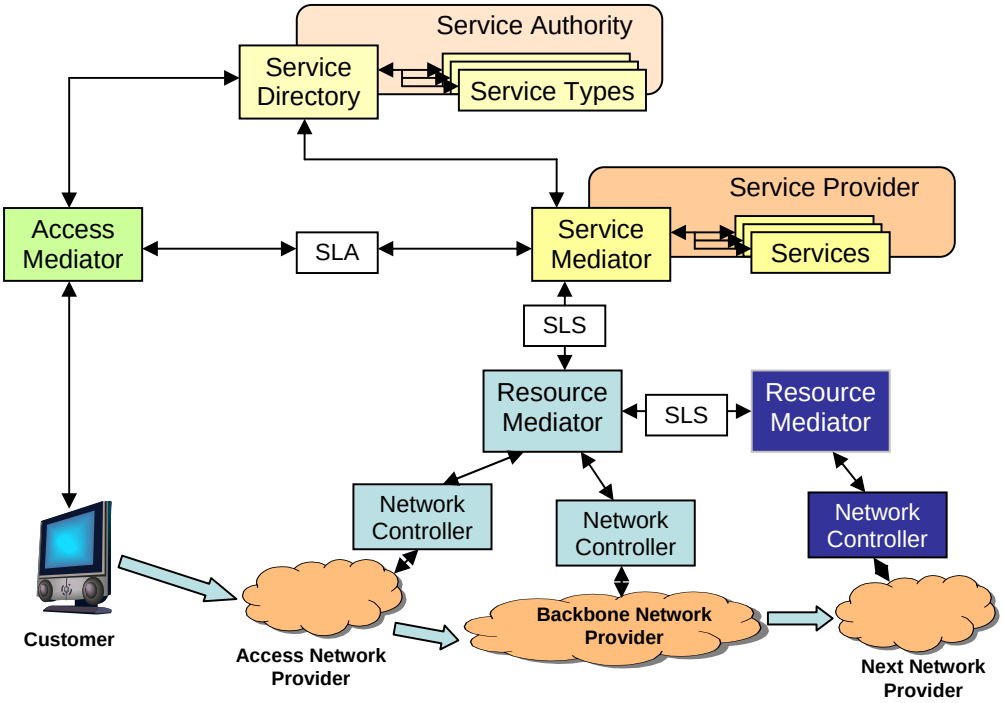

Figure 2.1 – The CADENUS Architecture

There is one RM for each domain[3] and it is in charge of managing all the underlying networks in the domain. The RM can answer questions about the underlying networks, e.g. is it possible to reserve resources between two points in the domain at a specific time interval in the future?

Each network in the domain can use a different QoS technique and some may even not care about QoS and just rely on over provisioning. For each such network a NC rules. The NC is responsible for doing physical things with the network like policing, scheduling, admission controls etc. If a new QoS technique is invented and used for a specific network, you just have to create a new NC for that network. This makes the CADENUS architecture highly modular and scalable.

When a customer wants to buy a specific service he connects to an Access Mediator (AM) that presents him with the available services. The customer then has to choose service, quality of the service and the time the service will occur. The quality of the service in this stage is represented in a simple way, like gold, silver or bronze quality. This information is converted into a Service Level Agreement (SLA) that the AM transfer to the Service Mediator (SM). The SM role is to mediate services from different service providers and to convert the SLA into a

[3] Autonomous System (AS)

Service Level Specification (SLS). The SLS is a more technical description of the service, with parameters describing how the network must perform to achieve the demands in the SLA.

The SM then has to send the SLS to the underlying RM or RMs depending on how the networks are organized. There are two possibilities, the Hub model or the Cascade model. In the Hub model the SM sends the SLS to each and every RM along the service chain and in the Cascade model only the first RM receives the SLS. The RM then has to "split" the SLS and share it with the next RM along the chain.

When the RM receives the SLS he must check if the domain he controls can handle the demands stated in the SLS. The RM does this by asking the underlying NCs if each single network that must be used for providing the service is capable of these demands. If all the networks and domains along the chain are capable of providing these demands an advance reservation can be done. The customer will be presented with a final cost for doing this. The customer will be notified if the networks are not capable of providing these demands.

2.2 Research Assignment

There is a student project at Ericsson inspired by the ideas behind the CADENUS architecture. The goal of the student project is to implement a system similar to the CADENUS system to offer high quality IP-services with perform guarantees. Previous students have already implemented some parts in the system. These parts were mainly concentrated on QoS and policing in the lower regions of the NC. The project therefore lacked most of the upper and vital components in the CADENUS system, the resource management system.

The research assignment was therefore to study a resource management system and implement a working resource management prototype inspired by the CADENUS project. A part of this thesis was also to investigate the contents of a SLS and how the SLS can be used in the resource management system.

2.3 Equipment

The equipment that was available for this thesis project is presented here:

- Test lab with Internet access
- Pentium 3 computers
- TV
- IPTV Set-Top Box
- Video Server
- Ethernet switches
- Cables
- Network cards

Chapter 3

Method Description

3.1 Course of Action

In the beginning of the work the main goal was to establish the purpose of the thesis and to create a preliminary time plan. After this a theoretical in-depth study about resource management[4] and CADENUS was conducted to understand what had previously been done in the Ericsson project and also to better understand the research assignment.

After 2 months a document describing the theoretical parts and a design proposal for a prototype was presented. When this document was accepted by Ericsson the implementation of the prototype started. The implementation ended with a practical demonstration.

The implementation was done with the top-down method, which means that the system was first designed with an easy sketch and gradually the details of the system improved.

3.2 Literature Search

Most of the literature was found using the search engine Google. Some of the search phrases were: “cadenus architecture”, “bandwidth brokers”, “service level specification”, “java web service”, “postgresql wsdp” and “tomcat wsdp”.

The first three phrases helped to find information about the concept behind the resource management system. Last three gave information about techniques used to implement the system.

The books used were old course literature from previously read courses at the department of Computing Science at Umeå University.

[4] See chapter 5 for the deep-study about resource management.

Chapter 4

Overview of Techniques

This chapter describes the different techniques used in the final prototype and thesis. This may help readers without previous experience in this field to understand the context. See the literature in the reference list for a more detail description of each technique.

4.1 XML

Extensible Markup Language (XML) is a simple and flexible way to format text. It is a markup language much like the HTML used to create web pages. The difference between them is that XML is designed to describe data and HTML is designed to display it. Both XML and HTML uses tags to structure the information, but the tags are not predefined in XML and you can create your own [24]. A big advantage with XML is that it is platform-independent and well-supported which makes it easy to exchange information between systems [20].

4.2 Web Services

Web services are used as the fundamental building blocks when doing distributed computing over the Internet. This means that you can create applications on a remote computer and then run that application remotely over the Internet. One of the primary advantages with web services is that it allows programs written in different languages on different platforms to communicate with each other in a standardized way [15].

4.3 SQL

Structured Query Language (SQL) is a standard computer language to create statements to access and update a database [23]. The SQL language is one of the major reasons for the success of relational databases in the commercial world. Because it is a standard language users can easily move their database applications to other types of database systems [14].

4.4 QoS Techniques

In order to guarantee performance and prioritizing data across networks one must use some form of QoS technique. Scheduling and policing are two mechanisms for providing these QoS guarantees.

Scheduling describes in which manner the queued packets are selected for transmission on a link and policing regulates the rate at which a flow is allowed to inject packets into a network.

IntServ (Integrated Services), RSVP, DiffServ (Differentiated Services) and MPLS are common techniques for achieving QoS in networks [9].

4.5 Java

Java is an object oriented, platform independent, programming language from Sun Microsystems. Applications written in Java are portable across multiple platforms without the need of recompiling. It has similarities with the C/C++ language, but the goal with Java was to make a simpler language, but yet powerful [10].

4.6 JavaScript

JavaScript is a programming language used on web pages. It can handle events, e.g. when someone clicks on a button the page will change in some way. It can be used to control that a user has filled a form or to present the current time and date etc. The scripts are executed on the local computer and not on a web server. JavaScript and Java is two different programming languages in both concept and design [21].

4.7 Java WSDP

Java Web Services Developer Pack (Java WSDP) is a toolkit by Sun Microsystems to build, test and deploy web services. Java WSDP contains different components, SAAJ, JAX-RPC etc. to create web services with different interfaces [16].

4.8 Tomcat

Tomcat is a web server and web container that web services run at. Tomcat also runs Java Servlet and JSP. Tomcat is developed under the Jakarta project at the Apache Software Foundation and is free to use [18].

4.9 JSP

JavaServer Pages (JSP) is a technology to create dynamic web content. The JSP technology is server and platform independent and is executed at the server side [17].
The JSP technology share many similarities with the ASP technology from Microsoft. Some of the differences are that ASP is platform dependent and runs on Windows. JSP is platform independent and can be run on many operating systems. The JSP technology also has the advantage that it uses Java language for scripting [19].

4.10 PostgreSQL

PostgreSQL is an object-relational database system that is free to use. It supports platforms like modern Unix-compatible platforms and it also runs natively on Microsoft Windows NT-based operating systems [13].

4.11 SOAP

SOAP (Simple Object Access Protocol) is a communication protocol based on XML that transfers data over HTTP and is used for accessing web services.

SOAP is language independent and platform independent can be used to communicate between various applications created and executed on different platforms. Because SOAP uses HTTP for transferring data, it gets around firewalls[5] [22].

[5] Firewalls usually do not block the HTTP traffic.

4.12 RPC

Remote Procedure Call (RPC) makes it possible from a client program to call a procedure in another program running in a server process [7].

When programmers use RPC for distributed applications they avoid to deal with details regarding the interface of the network [5].

Chapter 5

Resource Management

This chapter describes in details how Resource Management can be done with the CADENUS architecture. The theory in this chapter was essential to understand before the prototype system was designed and developed.

5.1 Negotiation

In order to create a system for high quality premium IP services, negotiation between different parts in the system is essential. This subchapter presents some cases when negotiation is done and what information it ends up with. It is based on facts from [1] [12].

5.1.1 Customer – Service Provider

When a customer wants to buy a specific service from a Service Provider (SP), a negotiation between them is done. The negotiation ends with a contract for that specified service, the so called Service Level Agreement (SLA). The SLA contains basic information that describes what the customer and SP have agreed on:

- Which service
- The time interval when the service should be delivered
- Which quality should the service have
- User authentication module
- Information about availability/reliability of the service

5.1.2 Service Provider – Network Provider

When the SP have received the SLA it is responsible for translating it into one or more Service Level Specifications (SLSs). The SLS is a technical document in opposite to SLA, which is a user-oriented service description. The SP then needs to negotiate with Network Providers (NPs) to try to configure the networks so that it corresponds with the parameters in the SLS. The

parameters are described in details in chapter 5.2.1 – 5.2.7; the following list gives a brief overview:

- Scope
  - The network ingress and egress points for the traffic flow.
- Flow Identification
  - Information to classify the packets belonging to the traffic flow.
- Performance Guarantees
  - Requirements imposed to the traffic flow.
- Traffic Envelope
  - Characterizing the traffic that the service will generate.
- Excess Treatment
  - Instructions how to treat out-of-profile packets.
- Service Schedule
  - Time schedule over which the service has to be exploited.
- Reliability
  - Parameters related to service reliability.

5.2 Service Level Specification

The SLS is a technical document that describes how the network and service must perform in order to maintain the desired service specified in the SLA. It is the interface between the service-aware and the network-aware components:

- Service Mediator – Resource Mediator
- Resource Mediator – Resource Mediator
- Resource Mediator – Network Controller

The information and structure of the SLS is independent of the upper service and the underlying network infrastructure. The main fields in the SLS definition are presented in details in the following subchapters. These facts and findings were gathered from [4] [12].

5.2.1 Scope

The scope parameters of an SLS associated to a given service indicates where the QoS policy for that specific service must be enforced. Therefore the scope uniquely identifies over which geographical/topological area the SLS concern. This is done by indicating the boundaries of that region. The scope is expressed by a couple of ingress and egress interfaces in the following way:

Scope = (ingress, egress)

The interfaces can be represented by IP addresses. The follow combinations of interfaces (ingress, egress) are allowed:

- (1, 1) – one-to-one communication
- (1, N) – one-to-many communication (N>1)
- (1, any) – one-to-any communication
- (N, 1) – many-to-one communication (N>1)
- (any, 1) –any-to-one communication

Either ingress or egress must be specified to exactly one interface. Therefore many-to-many communication (M, N) is excluded in the above list.

5.2.2 Flow Identification

The flow identification of an SLS associated to a given service indicates for which IP packets the QoS guarantees for that specific service must be enforced. The information it contains to identify the stream of IP datagrams can be the following:

- Differentiated Services information
  - (DSCP value, etc.)
- Source information
  - (Source addresses, etc.)
- Destination information
  - (Destination address, etc.)
- Application information
  - (Protocol number and source port, etc.)

5.2.3 Performance Guarantees

The performance guarantees parameters contain requirements on the traffic flow over the region described in scope for IP packets identified by the flow identification. There are four performance parameters:

- Delay
- Jitter
- Throughput
- Packet loss

The delay and jitter parameters indicate the maximum packet transfer delay and packet transfer delay variation from ingress to egress, measured over a time period with a length equal to the time interval specified in service schedule[6].

The packet loss probability is ratio of the lost packets between ingress and egress and the offered packets at ingress. The ratio is measured over a time period with a length equal to the indicated time interval.

The throughput is the rate measured at egress counting all packets identified by the flow identification.

5.2.4 Traffic Envelope

The traffic envelope describes the traffic characteristics of the IP packet stream identified by the flow identification. It contains a set of traffic conformance parameters that describes how the packet stream should look like to get the guarantees indicated by the performance parameters. This information is also needed to be able to decide if the traffic is "out-of-profile", e.g. does not fulfill some policy. The traffic conformance parameters contains following information:

- Peak rate (bits per second)
  - Maximum amount of packets that can be sent under a short time, e.g. max 1500 packets/second.
- Token bucket rate (bits per second)
- Bucket depth (bytes)
- Maximum Transport Unit (MTU) (bytes)
  - Maximum size of a link layer packet.
- Minimum packet size (bytes)

This information is needed when the IP packets in the network is controlled by some policy. For example, the QoS technique DiffServ uses the principle behind "The Leaky Bucket" which is one way of doing this [9].

5.2.5 Excess Treatment

Excess Treatment describes what to do with IP packets that do not conform to the policy, i.e. are "out-of-profile". Following measures can be used:

[6] See chapter 5.2.6.

- Dropping
  - All the packets marked “out-of-profile” is dropped.
- Shaping
  - All the packets marked “out-of-profile” is delayed so that they become “in-profile”.
- Marking
  - Change the classification for which the IP packet belongs.

5.2.6 Service Schedule

The service schedule describes over which time interval the service is going to be available. The time interval can be expressed in different ways, this is one example:

- Time of the day range
- Day of the week range
- Month of the year range

5.2.7 Reliability

Reliability describes the maximum time the service can be down per year and the maximum allowed time to repair, in case of service breakdown. It can be expressed in the following way:

- Mean downtime per year (MDT)
  - Might be expressed in minutes per year.
- Maximum allowed time to repair (TTR)
  - Might be expressed in seconds.

5.3 Resource Mediator Layer

This subchapter describes in details how the RM can act and perform to achieve resource reservations with the CADENUS architecture. All facts are based on [1].

5.3.1 Overall Functionality and Multi-domain splitting

One of the purposes with a resource management system is to be able to reserve resources in advance, e.g. bandwidth, for a given time interval. The SLS contains all the information we need for doing this. When the SM has transferred the SLS to the underlying RM layer, the main task for the RM is to process the SLS in some way.

The RM layer also needs to be able to store the SLS request for future activation. This is done in the Service Level Specification Repository (SLSR). The SLSR contains all the previously reserved requests that wait for activation in the future, see figure 5.1. In order for the SLSR to store requests based on time, it must be time-dependent. The RM layer has therefore full knowledge about what will happen in the future.

The RM layer must also be able to co-operate with other domains, especially when the negotiation of an SLS spans multiple domains[7]. The RMs that belongs to the crossed domains must then be involved in the negotiation phase. Each RM in the chain is responsible for assuring that their domain works in such a way that they together can fulfill the service requirements and guarantees in the SLS.

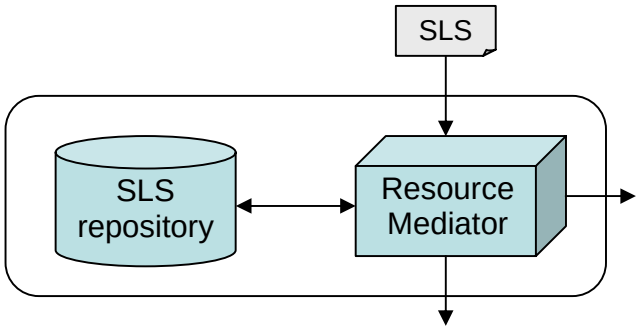

Figure 5.1 – Resource Mediator and SLS Repository

This chapter describes a solution when the Cascade model is used and therefore the SLS must be split and shared between the neighboring RMs, see chapter 2.1. The multi-domain SLS splitting process works like in figure 5.2. If the SLS spans more then one domain it has to be

[7] The ingress and egress points in the scope field are in different domains.

recursively split into two parts: an “own-domain” SLS and a remaining part. The “own-domain” SLS is targeted to the own underlying domain and the rest has to be transferred to the next RM in the chain and be enforced in a similar way there. By doing this we can achieve an end-to-end service configuration just by a single SLS instance.

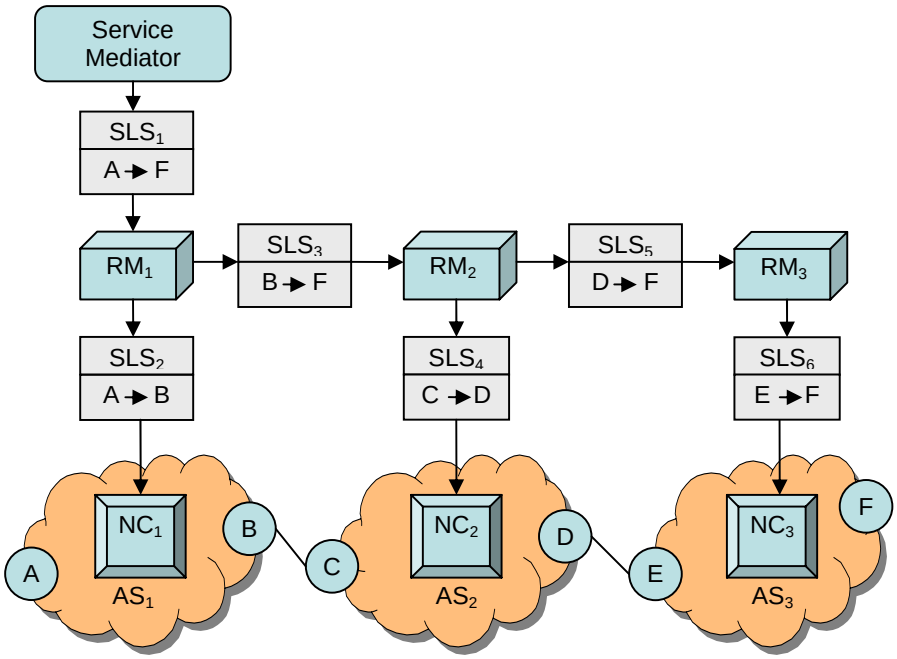

Figure 5.2 – Multi-domain SLS Splitting

When doing multi-domain splitting, the first task for the RM is to determine if the scope parameter tell if the SLS starts in the network domain owned by the RM. This is true if the ingress point belong to the domain that the RM control. If this test fails, the correct RM must be found in some way and the SLS is forwarded to that RM. The RM determines if the SLS scope crosses over the borders of the controlled domain and if it does, the RM split it into two parts. One part is kept and processed in the own domain and the rest is forwarded to the next RM. The next RM does the same check and may split the SLS once again. In order for the RM to know if the SLS belong to the domain or not, he asks the NC. The NC gives information about the boundary router towards a given destination. This information is crucial to determine if splitting is needed and how the new SLS will look like.

5.3.2 Admission Control and Activation

The RM has to process the information inside the SLS for at least two reasons:

- Admission Control
- Service Activation

To be able to accomplish both tasks in an effective way, the SLS must be translated into information that the underlying network infrastructure can understand. The RM is unaware of the underlying network and is therefore not directly responsible for this translating process. It relies on the NCs to deal with these tasks. The NC is the only component in the CADENUS architecture that has a clear picture about the network structure.

When admission control is needed the RM send the SLS to the NC that translates the SLS into a coherent set of network-related parameters, depending on the given network architecture. The NC then performs the admission control and returns the result to the RM. The admission control is based on decisions by exploiting the capabilities of the underlying network.

It is the RM responsibility to determine when a previously reserved request needs to be activated and made available to its customer, but the service activation process is done by the NC. When the NC activates the service, it configures the network devices so that they fulfill the SLS. The NC is unaware of time and must therefore be notified by the RM whenever a service needs to be activated or removed from the network.

5.3.3 Time-dependent Resource Management

Modern architectures for high quality premium IP services must take into account the time parameter. Time is an important factor that influences the QoS. But time also introduces a new dimension and makes the management of the network more complicated. Admission control for example will be influenced by the time parameter, since network resources are guaranteed a long time in advance and future network loads cannot be known at that time. The network architecture must also be able to react to asynchronously generated events and time synchronization may also be an issue.

In the CADENUS architecture, the RM is considered to be completely responsible for everything that involves time, which means that all underlying components are totally time unaware. This RM must therefore activate and give information to the NCs when a certain previously reserved time arrives.

Admission control and time is difficult to implement and will easy fall into a deadlock. This is because admission control is strictly attached to the architectural details of the underlying network. But the time aware RM lacks the detailed network knowledge. A solution to this is to give the NC all the global amount of resources which have already been reserved in a specified time interval. The RM can get this bundle of resources for the time interval by looking in the SLSR and then transfer it to the NC. In this way the NC will have all information about

previous reserved resources and also the new incoming SLS request. Together this information will be enough to decide if the new request can be reserved by doing the admission control.

5.3.4 The Resource Mediator Behavior in Details

Here is a step by step algorithm which describes how reservation in advance may be done with respect of time:

- RM splits up the SLS time interval in “time slices”.
- Check with SLSR to see how much resources are already used for each time slice in the time interval.
- Send the already done reservations including the new reservation query for one time slice to the NC.
- NC checks the old reservations and the new one with admission control.
- NC returns a cost function “cost” and the updated resource bundle for this time slice (to update the SLSR).
- Repeat for all time slices.

When a SLS arrives that concerns a reservation in the future, the RM split up the time interval in smaller parts, “time slice”. Then it checks with the SLSR to see how many resource requests is already reserved for each time slice. This information is passed down to the NC and the result from the admission control tells if we may or may not have capacity to do that reservation in the future. The NC also returns a “cost” that describes how much the cost is for traveling through the domain, see chapter 5.6.

Figure 5.3, the “RM algorithm”, describes in even more details how the RM must behave with respect to both multi-domain and admission control functionality.

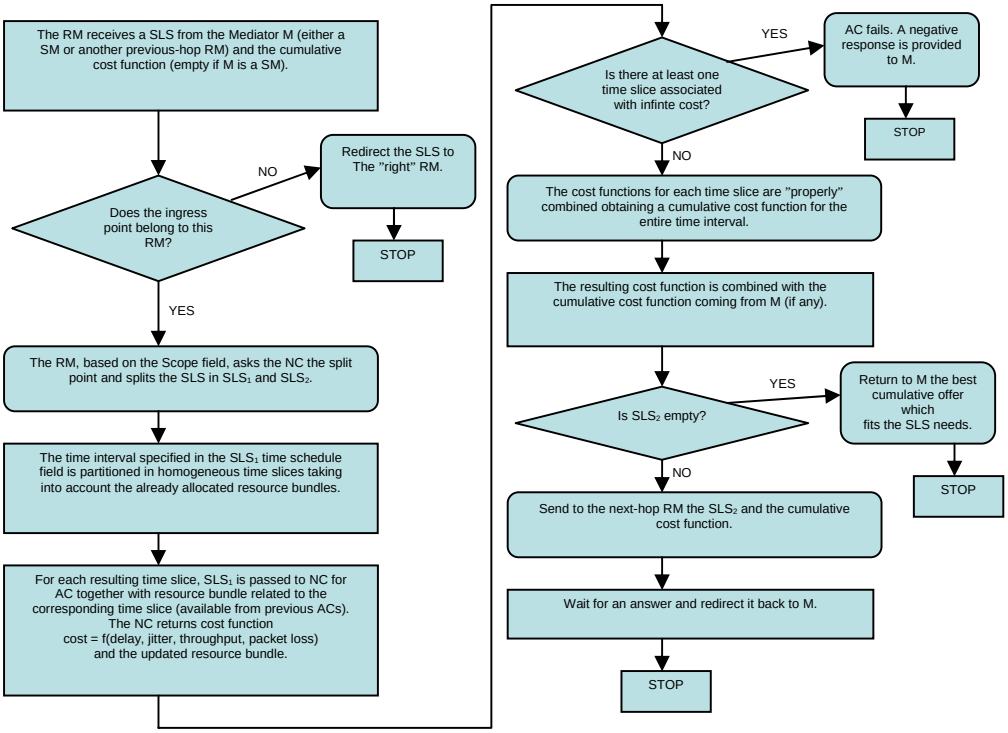

Figure 5.3 – The RM “algorithm”

5.4 Network Controller Layer

The NC is a component that rules over the network to which it belongs. It is constructed differently depending on which QoS technique that is used in the network. Therefore different networks need different implemented NCs. A new NC must be implemented whenever a new network or QoS technology is adopted. That is why the NC lacks all the networks independent functionality, such as inter-domain issues, splitting and time dependency. This makes the CADENUS architecture scaleable and easier to upgrade in the future.

Admission control is the main functionally for the NC layer. The underlying technology is well known and the NC will therefore be able to verify if a SLS can be accepted without jeopardizing the already accepted requests. Admission control is done differently depending on which QoS technique that is used.

When the NC wants to communicate with the network nodes, it can use the COPS (Common Open Policy Service) protocol. COPS is a simple query and response protocol used to exchange policy information between a policy server and its clients. The policy server is

called PDP (Policy Decision Point) and the client PEP (Policy Enforcement Points) [3]. The NC can also act as a PDP and send decisions to the network nodes acting as PEPs. In this way the NC can physically control the network devices[8].

Since the NC is the only component in the CADENUS architecture that has an overall view of the network topology, it must have an interface against the RM to provide it with useable information. When the RM determines if a SLS needs to be split or not, it has to be able to ask the NC for the following information:

- Whether the specified IP address in scope belongs to its own domain.
- The boundary router on the path towards the specified destination.

Therefore methods must be implemented in the NC to provide the RM with that information [1].

5.5 Devices Layer

The devices layer is the lowest layer. It is constructed by a set of network devices, independent of each others. The lowest level of action is performed here, translation of the network configuration policies into configuration commands whose syntax depends upon the network device implementation [1].

5.6 Cost Function

The cost for doing a reservation is also an important factor. When the NC is doing an admission control it may also return a cost. This cost is for doing the network configurations needed for fulfilling the demands in the SLS. A domain may have many solutions and possibilities to fulfill the demands. All these solutions may have different costs, depending on the delay, jitter, packet loss and throughput guarantees. The cost function can be written in the form:

cost = f (delay, jitter, packetloss, throughput)

When admission control is done on “time slices”, the costs must be properly combined to form a total cost for the entire time interval. One other issue is when dealing with multi-domain admission control. The issue is to find the cheapest cost in each domain and combine them in an optimal way so that the customer gets the lowest possible cost for doing the reservation [1].

[8] Routers, switches etc.

Chapter 6

System Development

This chapter describes how the resource management prototype system was implemented and developed.

6.1 Conclusive Parameters

The way the implementation of the resource management system was shaped and developed was based on some facts and parameters gathered during the work. The following subchapters describe this information.

6.1.1 Requirements

The following requirements were stated by Ericsson:

- Work with open source, non-commercial software and tools.
- Build software solutions with the Java programming language.
- Use object oriented structure to be able to reuse code in the future.
- The prototype must be able to interact with previous software solutions done at Ericsson.
- Offer a GUI to be able to demonstrate the implementation.
- Techniques and solutions for resource reservation should be inspired by the CADENUS architecture or similar systems.
- Implementation must be able to run on a x86-computer with Linux operating system.
- Use SOAP and XML technique when transferring data over network.

6.1.2 Limitations

At the physical lab at Ericsson, the possibility of using QoS techniques and admission control on a real network was limited and the development could therefore not be carried out on a real network. A solution to this restriction was needed to be able to implement the system.

6.1.3 Previous Work

Thesis students had previously implemented an API using Java to configure Ethernet switches using COPS[9]. This API makes it possible to set the maximum allowed bandwidth throughput on a physical port on the Ethernet switch. The previous work had to interact with the final prototype implementation in some way.

6.2 Evaluation and Planning

After the conclusive parameters were stated and the theoretical study about resource management was carried out, the evaluation of all these facts started. The final goal was clear; a prototype inspired by the CADENUS architecture with the possibility to make advance resource reservations was to be implemented. Now it was time to start planning and investigate how it could be done with techniques and software solutions. Following questions helped finding answers:

- How much of the functionality in the CADENUS architecture must be implemented to show that resource reservation does work?
- Which components need to be implemented?
- What kind of software solutions and techniques should be used?
- Do we have any software or physical limitations?
- How should the SLS data be represented?
- How can the programming code be structured to achieve easier future upgrades?

The AM and the ability to create a SLA are not vital for creating a simple prototype. Not even a SP is needed. A simple SM able to send a basic SLS, a RM and a NC is enough to build a working prototype. To limit the task and make it suitable for a 20 credit thesis project only these vital components with following capabilities were stated to be developed:

- Service Mediator
  - Offer a GUI to create a SLS by filling a form.
  - Send a SLS to one RM (Cascade model).
- Resource Mediator
  - Communicate with neighboring RM.
  - Splitting SLSs.
  - Store reserved SLSs in a SLSR database.
  - Show the reserved content in the SLSR as a GUI.

[9] See chapter 5.4 for more information about COPS.

    - Ask the NC for admission control.
    - Ask the NC about the topology in network.
    - Handle time and events.
    - Activate the NC on events.
- Network Controller
    - Receive and process SLSs.
    - Communicate with upper RM.
    - The ability to make admission control on a network.
    - Return answers about the network topology.
    - Show as a GUI, all SLSs that is currently activated.

The lack of physical QoS networks to work with can be solved by introducing a virtual test network. This means that that the NCs have a topological map of a simulated network instead of a real physical one.

Java was the chosen language due to the stated requirements. In order to implement a prototype with the ability to be upgraded and expanded in the future the Java code was suggested to be structured in packages and compressed to a single Java archive file[10].

6.3 Prototype Overview

The prototype was constructed by five components, one SM, two RMs and two NCs as figure 6.1 shows. Each component is represented by one Pentium computer running Linux. Together there are five computers, “avocado”, “selleri”, “paprika”, “tomat” and “gurka” connected to each other by a local area network.

This prototype system can reserve resources by creating a simple SLS through a GUI at the SM “avocado”. The SM then sends the SLS to the nearest RM “selleri”. The RM “selleri” then perform multi-domain splitting if it is needed. The RM “selleri” has the SLSR database and it is also time aware and can therefore see all previous reserved requests. The RM takes out all the previous reserved requests[11] that coincide with the time interval specified in the newly created SLS. The old reserved SLSs together with the new SLS form a resource bundle. This resource bundle is then sent to the NC “tomat” for admission control.

The NC “tomat” has a simulated virtual network topology map that it performs the admission control on. If this “virtual network 1” can handle and accept the resources stated in the new SLS and the old already reserved ones, the cost value for doing this is returned to the RM “selleri”. If the SLS were split, the RM “selleri” sends the second SLS part to the neighboring RM “paprika”. The RM “paprika” also asks the underlying NC “gurka” for admission control and cost for doing reservation in the same way as before. When the RM “paprika” knows the cost for doing reservation in his “virtual network 2”, the cost value returns

[10] “JAR” file.

[11] Each is represented by a single SLS.

to the RM “selleri”. The RM “selleri” then creates a total cost value by adding the two cost values and returns it to the SM “avocado”.

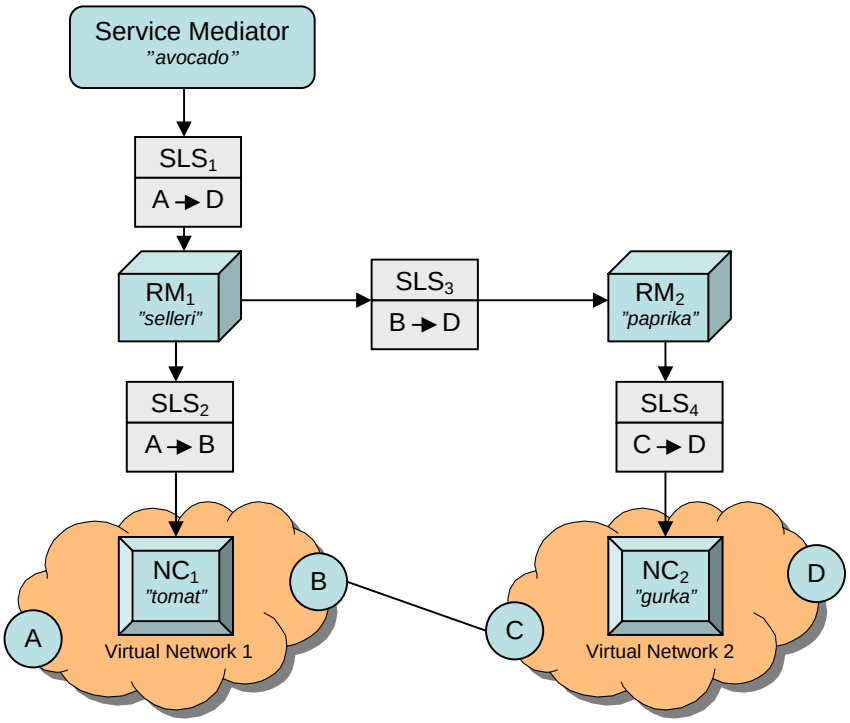

Figure 6.1 – Prototype Overview

6.4 System Description

This subchapter describes how the different parts in the prototype system was constructed and built. The Service Mediator, Resource Mediator, Network Controller, SLS and SLSR are the most important parts and are described in the following subchapters.

6.4.1 SLS

One of the most important parts in this system is the Service Level Specification. This technical document must be represented as a data structure in some way. Because the chosen programming language was Java it was natural to represent the SLS as a Java object. The final solution was therefore to create a Java class with the parameters found in chapter 5.2. The class is constructed with two types of methods, set and get. The set methods are used to set values in the SLS objects and the get methods to get the values. Here is a code example from the SLS class in the `SLS.java` file:

```
private String scopeIngress, scopeEgress;

// ------------- <Scope> -------------
public void setScopeIngress(String str)
{
   scopeIngress = str;
}

public void setScopeEgress(String str)
{
   scopeEgress = str;
}
// ----------------------------------

// ------------- <Scope> -------------
public String getScopeIngress()
{
   return scopeIngress;
}

public String getScopeEgress()
{
   return scopeEgress;
}
// ----------------------------------
```

The SLS class implements “Serializable” so that SLS objects can be serialized. This makes it possible to write SLS object directly into a database or send it over networks by using RPC.

6.4.2 Transferring Data

The prototype is built around the toolkit Java WSDP that makes it possible to create web services that uses SOAP/XML technique.

The RM and NC components are each a single web services, run and deployed on a web server. The Tomcat web server is used as the web container for each web service. Java WSDP contains different kind of techniques and solutions for creating web services, but in this prototype the Java API for XML-based RPC (JAX-RPC) was used. This technique makes it possible to make remote procedure calls using SOAP and XML.

To transfer data between the components, methods are called remotely with serialized Java objects containing data. When a Java object is serialized then it is converted into a byte-array containing the binary representation of that object. When the remote method gets a serialized object it must deserialize it back to a Java object before it can be used.

6.4.3 Service Mediator

In a final system the SM needs to be able to offer and manage different services and negotiate with an AM among other things, but in this prototype system the functionality of the SM is very limited. The main goal was to implement a small part of the SM, the ability to create a SLS and send it to a RM.

The SM has a web page as GUI with the ability to fill a form to create a SLS. When the form is filled the SLS Java object is created. Then the SM calls a remote method called "`recvSLS`" on the RM and passes the SLS object as a parameter to the RM. The value that the remote method returns is either the total cost value for doing the reservation or a string with a message that the reservation could not be done. Figure 6.2 shows how the SM looks inside in terms of functionality.

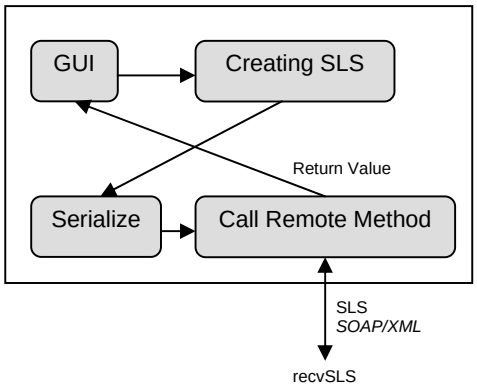

Figure 6.2 – Inside the Service Mediator

6.4.4 Resource Mediator

The RM was implemented as a web service with the "`rescSLS`" method as an interface to the SM. The RM is time aware and has an internal clock that keeps track of the time.

When the RM receives an incoming SLS from another component, the RM can read out the egress point in the SLS to determine if the destination IP address belongs to the own domain. The RM needs to ask the NC for this information because the RM does not have any knowledge about the underlying network. The NC is also implemented as a web service that has a remote method "`isIPAddrMine`" that helps us with providing that information. The method takes the egress address as a parameter and return a true or false value if the address belongs to the domain or not. If the answer is true, splitting is not needed and the SLS will only be processed by this domain. If the answer is false, splitting is needed and more information from the NC it vital. In order for the RM to split the SLS in a correct way, information about the boundary router in the domain is needed. This information is once again received by calling a remote method "`getBoundaryRouterForDest`" from the NC with egress address as parameter.

Now we have enough information for creating a second SLS for the neighboring domain. The RM sends this SLS to the next RM along the service chain by calling the remote method "`sendToNextRm`" on the neighboring RM with the SLS as parameter. The neighboring RM will now make the same splitting check. Eventually all RM along the chain from ingress point to egress point will have a SLS concerning their domain. For more information about multi-domain splitting, see chapter 5.3.1.

The RM now needs to evaluate if the requirements in the SLS can be fulfilled. This is done by doing admission control. Once again the RM needs to call a remote method on the NC for doing this because of the unawareness of the network. But as the algorithm in chapter 5.3.4 suggests, the SLS needs to be sliced up in "time slices" in order to do reservations with respect of time. In the prototype a time slice is equivalent to one minute and is represented in seconds from 1 January 1970, the so called "Unix time" [2]. For each time slice, the already reserved resources in the SLSR database that intersect in time with the new request, is gathered together to a resource bundle. The resource bundle and the new request are transferred to the NC for admission control. This is done by calling the remote method "`admissionControl`" on the NC, with the resource bundle and SLS as parameter.

The remote method will return a cost value for each time slice, and together it gives a total cost value for that time interval. If one of the cost values is smaller then zero, admission control fails and reservations could not be done. If all the cost values are bigger then zero, the RM is ready to reserve these resources. This is done by adding the SLS to the SLSR database. The following code taken from the Java file `ResourceMediatorImpl.java` describes how the admission controls and time slicing was written:

```
UnixTime ut = new UnixTime();
long startTime = ut.getElapsedStartTime(sls);
long endTime = ut.getElapsedEndTime(sls);
ResourceBundle rb;
long timeInterval = 60;
…

for(long t=startTime; t<endTime; t=t+timeInterval)
{
   rb = slsr.getResourceBundle(t);
   cost = nct.admissionControl(rb, sls);
   …
}
```

The object "`ut`" has methods for converting the service schedule parameters in the SLS to Unix time with a time interval, start time to end time. The "`getResourceBundle`" method in the SLSR returns all reserved SLSs that intersect with the time "`t`" as a resource bundle. The "`nct`" object is for calling the remote methods at the NC.

The RM has a Java thread running in the background that keeps track of time. Once every minute, the thread goes through the SLSR database to see if there are any reserved SLSs that needs to be activated. If there are, the RM removes these SLSs from the database and sends the SLSs to the NC for activation. This is done by calling the remote method "`activateSLS`" on the NC with all the concerning SLSs as parameter. The thread in the RM can also in an opposite way deactivate the current activation in the NC by calling the remote method "`freeResources`". This is done when the SLSs no longer are scheduled to be active.

A simple GUI is also implemented on the RM and is constantly updated when changes occurs in the SLSR. It is represented as a web page that gives information about future resource reservations stored in the SLSR database.

Figure 6.3 gives an internal overview of how the RM works.

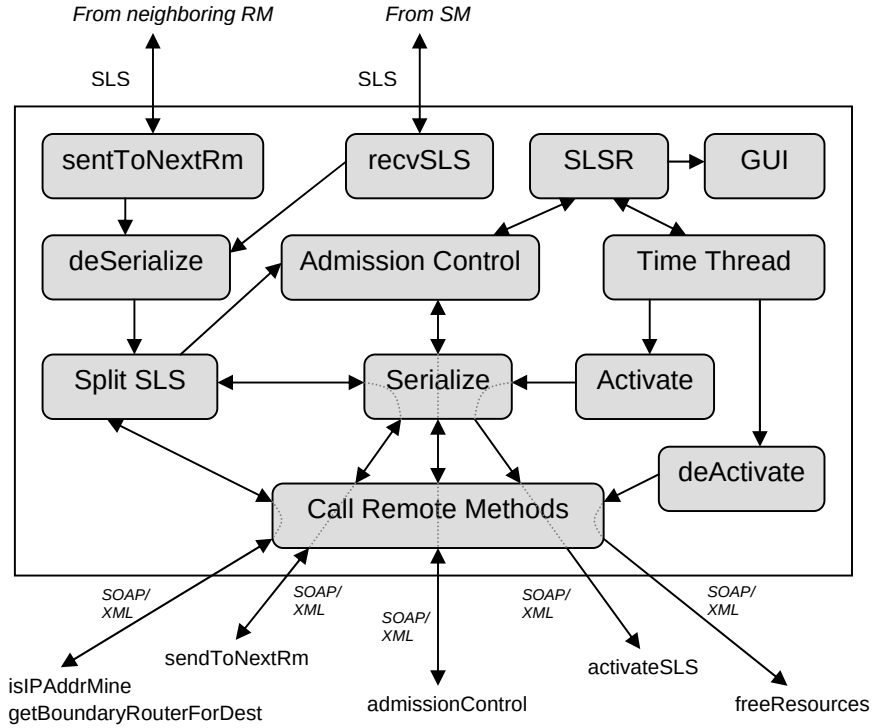

Figure 6.3 – Inside the Resource Mediator

6.4.5 SLS Repository

The SLSR is the component inside the RM that is responsible of storing all the SLSs that previously has been reserved. The PostgreSQL database system was used to store this information. PostgreSQL uses the SQL language to access and update the database.

A separate Java class called "`DatabaseConnection`" was constructed just for accessing the database. This makes it easy to change database system in the future, just rewrite this single class. Another Java class called "`SLSRepository`" contains all the methods and SQL statement for adding or removing reservations etc.

The database is constructed with two tables, "`slsdata`" and "`slstimecost`" as figure 6.4 shows.

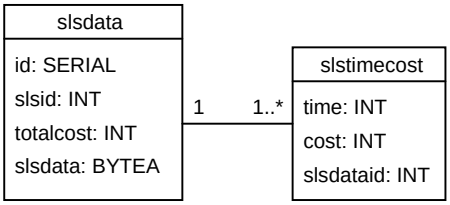

Figure 6.4 – Physical Data Model for SLSR

Each time a resource reservation is done by adding a SLS to the SLSR a new row is inserted into the table "`slsdata`". For each new row the "`id`" value is increased by 1 and the SLS is serialized and stored as a byte-array. For each minute the SLS is scheduled to be active, a new "`slstimecost`" row is created with the current minute time stored as the "`time`" value. Each such row holds a "`slsdataid`" value that is a reference back to the "`id`" value for the concerned "`slsdata`" instance. In this way we can easily search the "`slstimecost`" table to find stored SLSs that intersect in time with each other without having to work against the serialized SLS data stored in "`slsdata`". This also means that we only have to keep one copy of each SLS in the database and therefore storage space is not wasted.

Here follows an example of a SQL statement that can be used on the final SLSR:

```
SELECT D.slsdata, TC.cost
FROM slsdata D, slstimecost TC
WHERE TC.time = '1143047100' and D.id = TC.slsdataid
```

(Selects all SLSs that intersect with the time "1143047100")

6.4.6 Network Controller

The NC is also implemented as a web service, just like the RM. It has five methods that can be accessed remotely from the RM: "`isIPAddrMine`", "`getBoundaryRouterForDest`", "`admissionControl`", "`activateSLS`" and "`freeResources`".

When the RM needs to know if the egress address in the SLS belong to the own domain, it calls the "`isIPAddrMine`" method on the NC with egress address as parameter. The NC then has to check with a topologic map over the underlying network to be able to answer that question.

In this prototype a simulated topologic map was used instead of a real network. The virtual test network was constructed with three Java classes, "`Topology`", "`Path`" and "`Router`". Together these classes simulate a complete virtual network with cost values for crossing paths and also maximum capacities over different paths. In the prototype setup two NCs were created, "tomat" and "gurka". These two together sees the virtual network in figure 6.5.

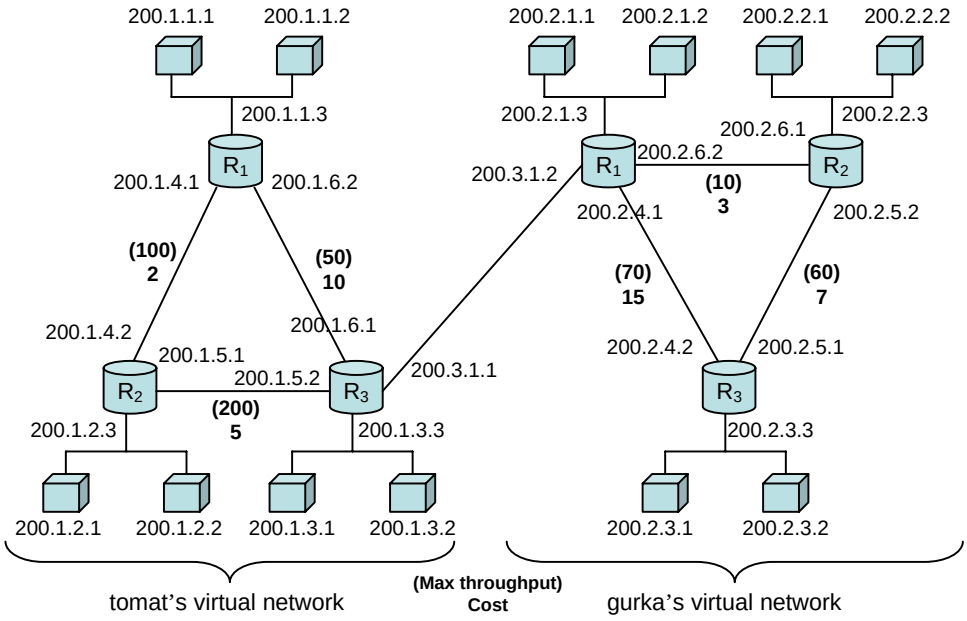

Figure 6.5 – Virtual Test Network

If the RM above one of the NCs wants to do splitting he must know the boundary router in the domain. This is done by calling the remote method "`getBoundaryRouterForDest`" in the NC. The NC will then look into the simulated topological map over the virtual network and answer that question.

In this prototype we can only reserve and guarantee a maximum throughput in terms of bandwidth over a path specified by the egress and ingress addresses in the SLS. The RM needs to perform admission control with the throughput performance guarantees specified in the SLS to be able to determine if it is possible to offer guarantees and therefore reserve them. This is done by calling the remote method "`admissionControl`" on the NC. The NC will then evaluate if the maximum capacity on each path between the ingress and egress points in the own domain is enough for accepting this SLS. If it is enough the cost value for traveling over these paths is returned to the RM. If not, a negative value is returned to indicate that capacity is not enough and therefore reservation is not possible at this time.

When a reserved SLS is to be activated, the physical network needs to be configured in a way that makes the network perform as specified in the guarantees of the SLS. In this prototype the NC is very limited due to the virtual network and therefore the task of configuring the network becomes a little bit tricky. Therefore a simple workaround have been implemented. Previous students have created a PDP for controlling a PEP and Ethernet switch with COPS.

This makes it possible to control the maximum throughput on a physical port on the Ethernet switch. When a hard coded "virtual IP address"[12] is set as the egress value in the SLS, the PDP will trigger the PEP to configure the Ethernet switch with the throughput guarantees stated in the SLS. The result is a prototype with the ability to reserve throughput guarantees in future time, and when the time arrives, configure a physical Ethernet switch depending on the throughput value in the SLS.

When the time comes to deactivate the current configuration, the RM calls the remote method "`freeResources`" at the NC. This method configures the Ethernet switch to perform with a low throughput to make the network unusable for the service.

A GUI is implemented to show SLSs currently active on the NC. A web page shows this information.

Figure 6.6 gives an internal overview of the functionality in the Network Controller.

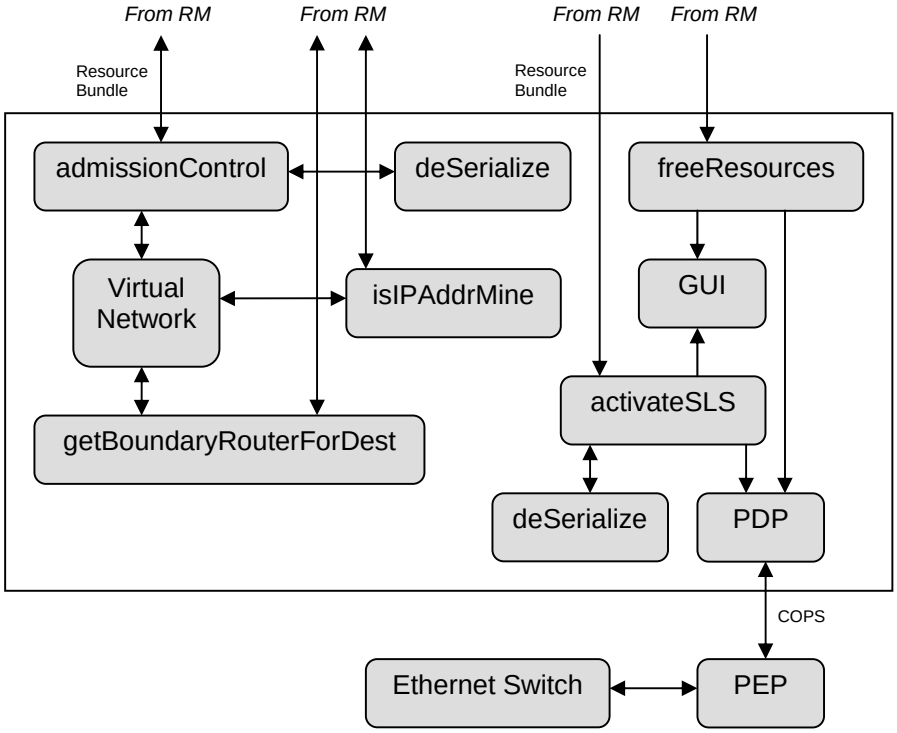

Figure 6.6 – Inside the Network Controller

[12] E.g. 200.2.3.1 in the virtual network in figure 6.5.

Chapter 7

Results

The final result of this master thesis is the prototype itself. This chapter describes how it can be used in a practical matter to make bandwidth reservations for IPTV. Some screenshots and figures are also presented to give a better view of the physical system and how it works.

7.1 Prototype Setup

The prototype system consists of 5 Linux computers connected in a 10 mbit/s local prototype network, acting as 1 SM, 2 RMs and 2 NCs[13]. Previous students at Ericsson have developed a management network with a PDP and PEP for controlling an Ethernet switch. This management network together with an IPTV Set-Top box capable of receiving video streams also became a part of the final prototype system.

One of the RMs, "selleri", acts as an Internet firewall with an extra network card to provide the local prototype network with Internet access. The Internet access is vital so that all computers can synchronize the internal clock to a remote time server using the Network Time Protocol (NTP). It is important that all components are configured with the same time when doing reservations with time dependencies.

The NC "gurka" also has an extra network card so that it can access the management network and control the PDP/PEP and Ethernet switch. The IPTV Set-Top Box is connected to a video server streaming video through the Ethernet switch. With this setup it is possible to control the bandwidth throughput between the Set-Top box and the video server with the final prototype system. The NC "gurka" will configure the Ethernet switch with the PDP when a hard coded virtual IP address is present as the egress address in the SLS. The NC will then limit the bandwidth throughput on a specific physical port with the throughput guarantees stated in the SLS. The following virtual IP addresses are hard coded to a specific physical port on the Ethernet switch:

- 200.2.3.1
  - Configures the physical port 9 on the Ethernet switch
- 200.2.2.2
  - Configures the physical port 14 on the Ethernet switch

[13] See chapter 6.3 for a Prototype Overview.

Figure 7.1 shows the final prototype setup with all the acting components that belongs to it.

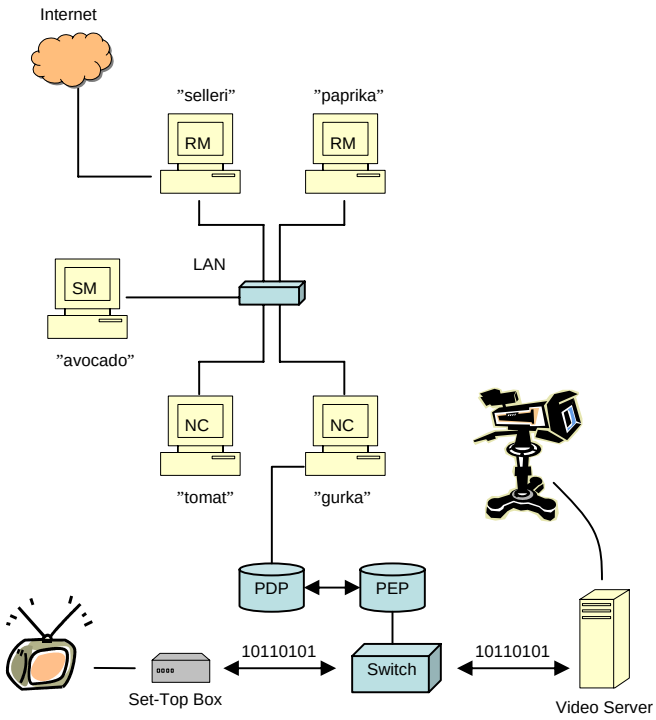

Figure 7.1 – Final Prototype Setup

7.2 Prototype in Action

In this subchapter a step by step scenario will be presented to show how the prototype looks like in action. This is shown with some screenshots from the actual software.

7.2.1 Step 1 – Create SLS

By opening the web page on the SM “avocado” we can create a SLS. The window to the left in figure 7.2 shows the GUI on the SM.

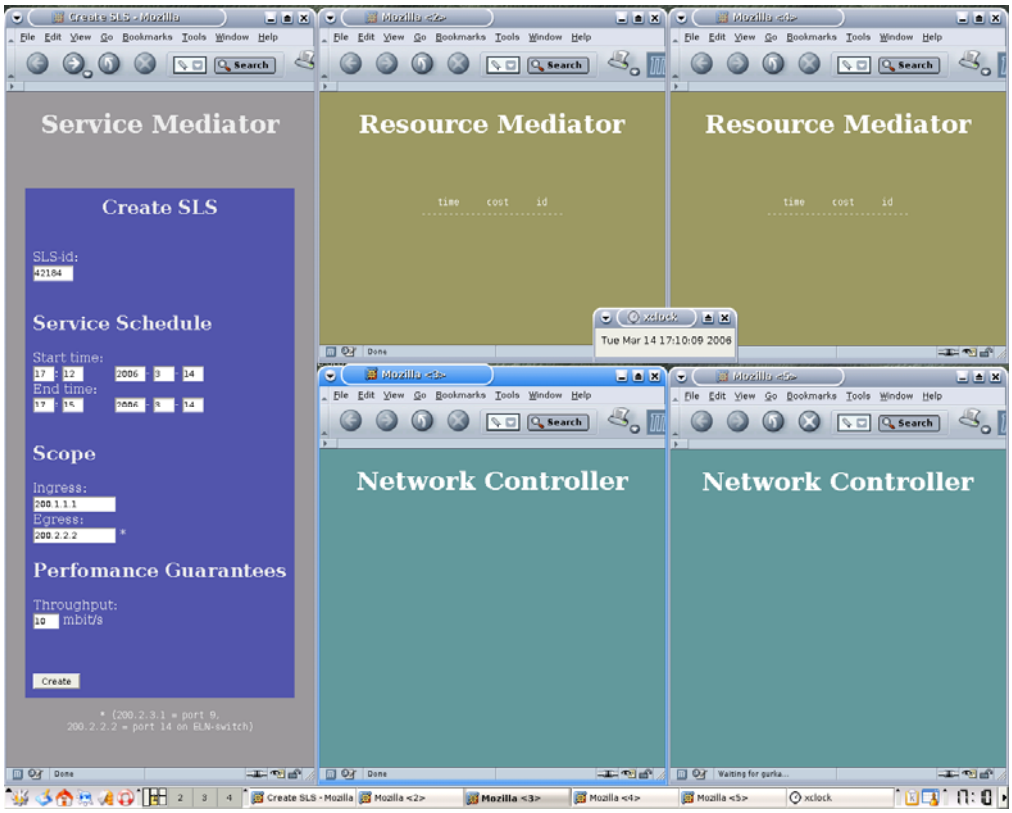

Figure 7.2 – Screenshot from the Final Prototype

The SLS is created by filling the form on the SM web page. The only data that the SLS contains in this prototype is the id, service schedule, scope and the throughput performance guarantee. The web GUI is created with JSP and JavaScript and fills the current date and time automatically once every 45 seconds. The time field is filled in a way so that the time interval will start within 2 minutes and stop after 3 minutes of activation. This makes the prototype easy to demonstrate, just press the “Create” button and you always will reserve requests in advance in a 3 minute time span. In this example the reservation is done between 17:12 and 17:15 on March 14, 2006. In the field “Egress” the virtual IP address is stated. In this example “200.2.2.2” is typed in, which means that the physical port 14 on the Ethernet switch will be configured using COPS. With COPS the switch will be configured to perform with the value stated in “Throughput”, in this scenario 10mbit/s. This will make our IPTV Set-Top box receive a video stream fast enough to be able to be watched on TV.

7.2.2 Step 2 – Cost Value and SLSR Storing

Now the SLS will be transferred to the first RM, be splitted, forwarded to the next RM, admission control will be performed and finally the RMs will reserve and store the SLS in the SLSR and the cost for doing this reservation will be returned to the SM.

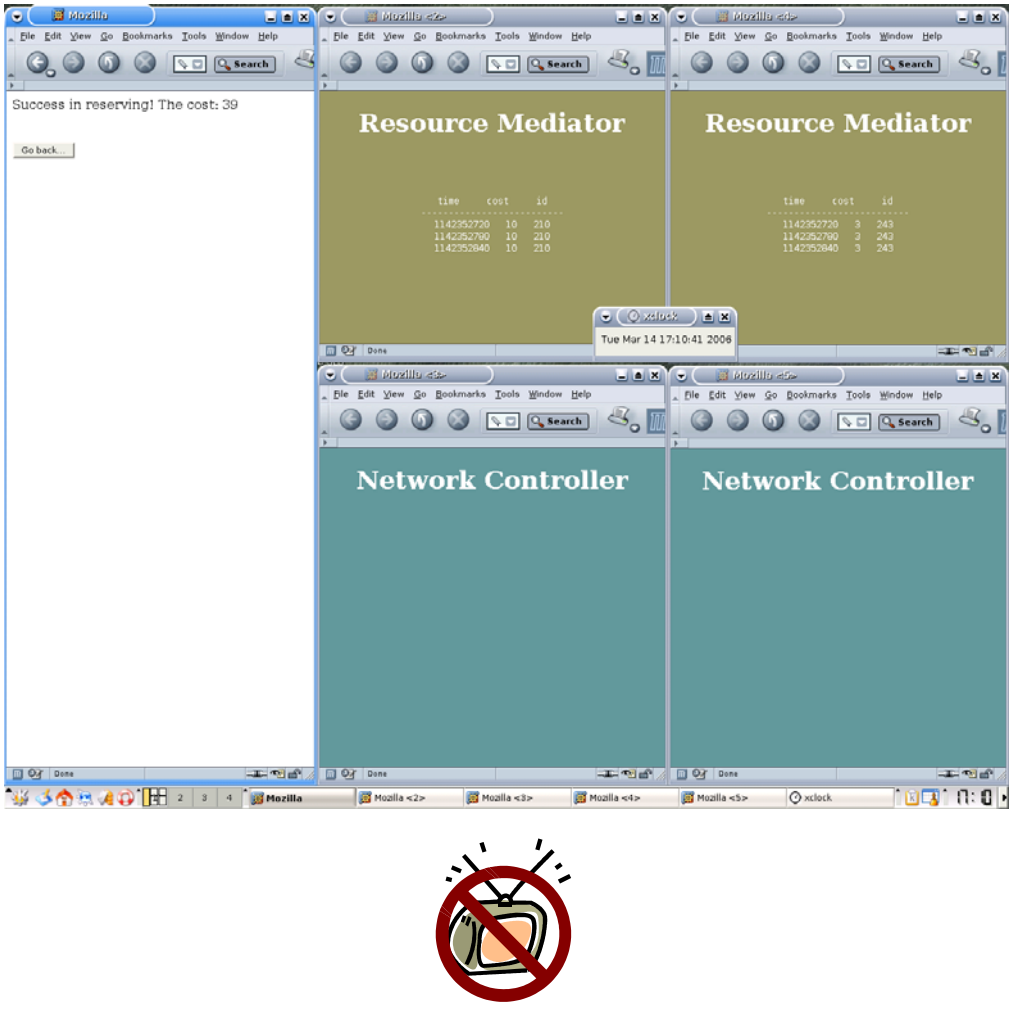

Figure 7.3 – Advance Reservation with returned Cost Value.

In figure 7.3 we can see that the SM has presented the final cost for doing this reservation, “39”. This cost value is just a value at this time and not converted to any currency. The value “39” is based on a reservation done in the virtual network described in chapter 6.4.6. In figure 7.2 we can see that the reservation was done between the ingress and egress points “200.1.1.1” and “200.2.2.2” during 3 minutes. The cost for traveling over a path per minute in the virtual network is shown in the topologic map in figure 6.5. In this reservation we have to travel over 2 paths with cost “10” and “3” which gives the final cost (10+3) * 3 = 39.

The upper two windows in figure 7.3 show the GUI from the two RMs. There are 3 lines in each RM displaying the table content in the SLSR. Each line represent one minute reserved in advance.

7.2.3 Step 3 – Activation

When the time arrives to activate the reserved SLSs the RMs transfers from the SLSR the reserved SLSs to the NCs. The NCs will then display the active SLSs in the web page GUI. The two lower windows in figure 7.4 show this.

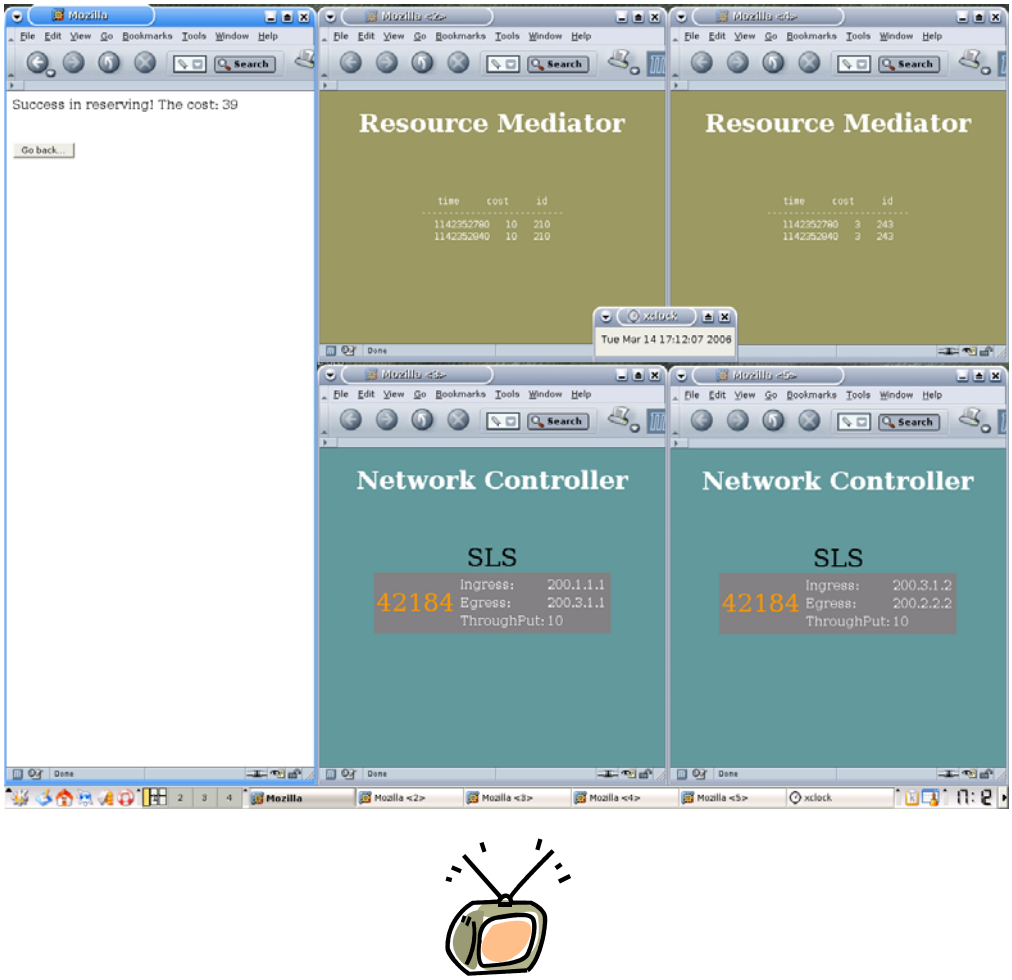

Figure 7.4 – Service Activation

The RMs will now have one line less then before and the NCs displays the active SLSs. The NCs also shows some details taken from the SLSs, like ingress, egress, id and throughput.

Now the flow between the IPTV Set-Top box and the video server will increase and we have the ability to see TV.

7.2.4 Time Consumption

It takes around 3-4 seconds from that the “Create SLS” button is pressed until the splitting, admission control and storing of the SLS is done. This seems quite fast but it can not be related to a real scenario network, because we are using the local prototype network and a simulated virtual test network.

Chapter 8

Conclusions

This thesis consists of two parts, the theoretical part about resource management and a practical system development part where the theories were implemented.

8.1 Resource Management

Resource management is an important and interesting subject that is becoming more and more important with more demanding IP services. In order to develop a resource management system that can interact between different domains and actors on the network market, standardization is needed. The CADENUS architecture is therefore just an interesting concept that should be observed as a good resource management example until standardization is done.

The idea of using many components with different responsibilities seems to be a clever way to achieve a system that is easy to upgrade when new technology arrive. The use of a virtual network in the NC clearly implies this and shows that such a system is scaleable.

8.2 Prototype

The practical implementation of the resource management system shows that each component can be implemented as a web service to achieve connectivity between each other. The thesis also shows that the use of RPC over SOAP/XML and multi-domain splitting of the SLSs is possible.

The time for doing a reservation took about 3-4 seconds. This may seem quite fast, but that is on a local prototype network. In a real scenario, when traveling over many domains with different network providers, it may take longer. One thing that may contribute to this speed is that the resource bundle sent to the NC when doing admission control contains all the previously reserved SLSs that intersect with the concerned time. In a real scenario hundreds and probably thousands of SLSs will intersect and the resource bundle will be huge. The time for sending and handling such a large bundle may be a problem that has to be solved in the future.

8.3 Limitations and Future Work

There are some limitations and things that should be corrected and upgraded in the prototype system in the future. Here is a list of things to improve:

- SLS parameters in the SLS Java class
  - In this prototype only some of the most basic parameters are present, in the future more parameters is needed to be able to achieve better resource management.
- Network Controller
  - The entire NC should be replaced with a NC that can achieve admission control and network configurations on a real network.
- Virtual test network
  - If the current NC is not replaced and work is continued on the virtual test network, there are some limitations. The cost and maximum throughput for traveling between different virtual domains are 0. There should be a negotiation between the domains about the cost value.
- Service Mediator and upper layer
  - The SM is very limited in this prototype. In the future the entire upper layer with a Service Mediator, Service Providers, Access Mediator, Service Directory and SLA etc. must be implemented to complete the CADENUS inspired system.

8.4 Problems and Reflections

This subchapter contains my personal reflection of my thesis work.

When I started to work with my thesis at Ericsson during September of 2005 the hardest part was to understand what the project was about. Two previous master thesis students helped to understand what had previously been done on the project. The project was big and it took almost 1 month just to get a complete picture.

There was also a bit of confusion in the beginning, what was my actual task in the project? After 1 1/2 month of research I presented my own idea of what my contribution could be. This resulted in a power point presentation with an overview of my research findings and my ideas of a prototype implementation. This paper was accepted by my supervisor at Ericsson and I started to work. The power point document helped a lot and is the base of the final master thesis and prototype implementation.

After 3 months of programming I had accomplished all my ideas and goals and that resulted in the final prototype. It took more time than what was expected. I had problems to compile and create the web services and that contributed even more in delaying my time plan.

I have learned a lot of new thing during my time at Ericsson and I have found that the resource management subject is a very interesting field. When I look back on the thesis project I am very satisfied with the result. It took longer than expected, but it left me with a great working prototype.

Chapter 9

Acknowledgements

I would like to start and thank my external supervisor Örjan Sahlin for his support and for giving me the opportunity to conduct my master thesis at Ericsson. I am also very grateful for all the previous projects Örjan have given me through out the years. The experiences have taught me so much.

Department Manager Per Ljungberg at Ericsson, for his support and ideas.

Ila Sabet for being such a nice roommate and for all her warm encouragement and support. Without her my days at Ericsson would have been lonely.

Everyone at my department at Ericsson for cheering and encouraging me.

Jerry Eriksson and Per Lindström my internal supervisors at the department of Computing Science at Umeå University, for helping me achieve an academic written thesis.

My family for constantly encouraging and believing in me. A special thanks to my dad, Erik André and his colleagues at Sony Ericsson for all the Friday lunches they invited me to.

Niklas Larsson my thesis opponent for being critical and helping me improve the thesis report.

The previous master thesis students, Tong Liu and Tao Huang for introducing me to the Ericsson project.

Last but not least, a special thanks to my friends at UmU for the encouragement and knowledge they gave me when I ran into technical problems.

Thank you all!

Johan André

April 28, 2006
Stockholm, Sweden

Acronyms

AC	Admission Control
AM	Access Mediator
API	Application Programming Interface
AS	Autonomous System
COPS	Common Open Policy Service Protocol
GUI	Graphical User Interface
IP	Internet Protocol
IPTV	Internet Protocol Television
JAR	Java Archive File
JAX	Java API for XML
JDK	Java Development Kit
JSP	JavaServer Pages
NC	Network Controller
NP	Network Provider
NTP	Network Time Protocol
PDP	Policy Decision Point
PEP	Policy Enforcement Point
QoS	Quality of Service
RM	Resource Mediator
RPC	Remote Procedure Call
SLA	Service Level Agreement
SLS	Service Level Specification
SLSR	Service Level Specification Repository
SM	Service Mediator
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query Language
WSDP	Web Services Developer Platform
XML	Extensible Markup Language

References

[1] A. Diaconescu, S. D'Antonio, M. Esposito, S. P. Romano and M. Potts. Resource Management in SLA Networks. Deliverable D2.3, rev. 5.0. IST-1999-11017.

[2] Ben Sintay. UnixTimeStamp.com. Webpage, 28 April 2006. http://www.unixtimestamp.com/.

[3] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan and A. Sastry. RFC 2748 - The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[4] D. Goderis, Y. T'joens, C. Jacquenet, G. Memenious, G. Pavlou, R. Egan, D. Griffin, P. Georgatsos, L. Georgiadis and P. V. Heuven. Service Level Specification Semantics and Parameters. Internet Draft, November 2000. http://www.ist-tequila.org/standards/draft-tequila-sls-00.txt (visited 2006-03-29).

[5] Dave Marshall. Remote Procedure Calls (RPC). Webpage, 30 March 2006. http://www.cs.cf.ac.uk/Dave/C/node33.html.

[6] Ericsson AB. Webpage, 31 March 2006. http://www.ericsson.com/.

[7] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems: Concept and Design, third edition. Pearson Education Ltd, 2001.

[8] ISTweb. IST Projects. Webpage, 28 April 2006. http://cordis.europa.eu/ist/projects/projects.htm.

[9] J. F. Kurose and K. W. Ross. Computer Networking: A Top-Down Approach Featuring the Internet, second edition. Pearson Education, Inc., 2003.

[10] J. Gosling and H. McGilton. The Java Language Environment: Contents. White Paper, Sun Microsystems, Inc., 1996. http://java.sun.com/docs/white/langenv/index.html (visited 2006-03-29).

[11] M. Esposito, M. Mondini and S. P. Romano. The Network Independent Policy Repository. Internet Draft, October 2001. http://wwwcadenus.fokus.fraunhofer.de/deliverables/NIPR-Definition-1Aug2001.pdf (visited 2006-03-29).

[12] M. Potts, R. Fiutem, S. Cirolini, P. Centoletti, S. D'Antonio, M. Esposito and S. P. Romano. Service Scenarios. White Paper, rev. 4.0. IST-1999-11017.

[13] PostgreSQL Global Development Group. PostgreSQL: Frequently Asked Questions (FAQ) for PostgreSQL. Webpage, 02 March 2006. http://www.postgresql.org/docs/faqs.FAQ.html#item1.1.

[14] R. Elmasri and S. B. Navathe. Fundamentals of Database Systems, third edition. Addison-Wesley, 2000.

[15] R. Wolter. XML Web Services Basics. Webpage, 28 February 2006. http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp.

[16] Sun Microsystems, Inc. Java Web Services Developer Pack (Java WSDP) Version 2.0. Webpage, 02 March 2006. http://java.sun.com/webservices/jwsdp/.

[17] Sun Microsystems, Inc. JavaServer Pages Overview. Webpage, 02 March 2006. http://java.sun.com/products/jsp/overview.html.

[18] Sun Microsystems, Inc. JavaServer Pages[tm] – Apache Tomcat - FAQ. Webpage, 02 March 2006. http://java.sun.com/products/jsp/tomcat/faq.html.

[19] Sun Microsystems, Inc. JavaServer Pages[tm] Technology – Comparison with ASP. Webpage, 02 March 2006. http://java.sun.com/products/jsp/jsp-asp.html.

[20] W3C Communcations Team. XML in 10 points. Webpage, 28 February 2006. http://www.w3.org/XML/1999/XML-in-10-points.html.

[21] W3Schools. JavaScript Introduction. Webpage, 28 February 2006. http://www.w3schools.com/js/js_intro.asp.

[22] W3Schools. SOAP Introduction. Webpage, 29 March 2006. http://www.w3schools.com/soap/soap_intro.asp.

[23] W3Schools. SQL Tutorial. Webpage, 28 February 2006. http://www.w3schools.com/sql/default.asp.

[24] W3Schools. XML Introduction – What is XML? Webpage, 28 February 2006. http://www.w3schools.com/xml/xml_whatis.asp.

Appendix A

Installation Guide

This guide describes how to install and compile the source code for the prototype implementation on a Linux operating system.

A.1 Preparations

1. Install the Linux operating system (e.g. debian) with a PostgreSQL database system.

2. Create a user "`admin`".

A.2 Download and Unpack the Prototype Code

1. Download the prototype code:

```
wget http:// -- concealed address -- /cadenusproject-060215.zip
```

2. Unpack the zip-file in home directory `/home/admin/`

```
unzip cadenusproject-060215.zip
```

3. Copy installation files to a new directory:

```
mkdir /home/admin/program
cp /home/admin/cadenusproject/download/* /home/admin/program
```

A.3 Install Sun Java JDK

1. Change directory to `/home/admin/program/`

2. Execute the installation file and follow the directions on screen:

```
./jdk-1_5_0_05-linux-i586.bin
```

3. Export some variables:

```
export JAVA_HOME="/home/admin/program/jdk1.5.0_05/"
export PATH=$PATH:/home/admin/program/jdk1.5.0_05/bin
```

A.4 Install Tomcat

1. Change directory to `/home/admin/program/`

2. Unzip the tomcat installation file:

```
unzip tomcat50-jwsdp.zip
```

A.5 Install Java WSDP

1. Change directory to `/home/admin/program/`

2. Export the display (this must be done because of a bug? in Java WSDP):

```
export DISPLAY=””
```

3. Execute the installation file:

```
./jwsdp-1_6.unix.sh –console
```

4. Make the following configurations in the installation process:

```
- choose your jdk: "/home/admin/program/jdk1.5.0_05"
- choose your container: "/home/admin/program/tomcat50-jwsdp"
- install in directory "/home/admin/program/jwsdp-1.6"
- choose typical installation
```

5. Export a variable:

```
export JWSDP_HOME="/home/admin/program/jwsdp-1.6"
```

6. Start tomcat server:

```
/home/admin/program/tomcat50-jwsdp/bin/startup.sh
```

A.6 Install the Network Controller

1. Change directory to `cadenusproject/software/NetworkController`

2. Configure the Virtual-Network by editing the code in:

```
src/server/NetworkControllerImpl.java
```

3. If you want this NC to be nr 2 “gurka” use the code under:

```
// *** Network controller 2: ***
```

4. If you want this NC to be able to control the PDP and Ethernet-switch use code under:

```
// *** Use this to control the PDP: ***
```

5. Compile the Network Controller and deploy it as a web service by running the script:

```
./compileAndDeploy
```

6. Restart the tomcat server:

```
/home/admin/program/tomcat50-jwsdp/bin/shutdown.sh
/home/admin/program/tomcat50-jwsdp/bin/startup.sh
```

7. Open the page http://nc-hostname:8080/nc.html to see the NC GUI (the web page is not available until a SLS is activated in the NC)

A.7 Install the Resource Mediator

1. Change directory to `cadenusproject/software/ResourceMediator`

2. Change user to “`postgres`”:

```
su
su postgres
```

3. Create the database and a database user by running the script:

```
./createDatabase
```

4. Change back to user `admin`.

5. Create the tables in the database by running the script:

```
./createDatabase2
```

6. Edit the file `config/wscompile_config.xml`

7. In the `<wsdl location="http://xxxxxx:8080/NetworkController/...">` tag, change `xxxxxx` to the name of the Network Controller that this Resource Mediator will connect to.

8. Edit the file `config/wscompile_config2.xml`

9. In the `<wsdl location="http://xxxxxx:8080/ResourceMediator/...">` tag, change `xxxxxx` to the name of the neighboring Resource Mediator (if there is one).

10. Create stubs for the web services by running the scripts:

```
./createNCSTUB
./createRMSTUB
```
(run only if you have a neighboring Resource Mediator)

11. Edit the file `src/server/ResourceMediatorImpl.java`

12. If you want this RM to be able to connect to a neighboring RM, uncomment the code after:

```
// *** For multi RM support: ***
```

13. Compile the Resource Mediator and deploy it as a web service by running the script:

```
./compileAndDeploy
```

14. Restart the tomcat server:

```
/home/admin/program/tomcat50-jwsdp/bin/shutdown.sh
/home/admin/program/tomcat50-jwsdp/bin/startup.sh
```

15. Open the page http://rm-hostname:8080/rm.html to see the RM GUI (the web page is not available until a SLS is reserved in the SLS Repository)

A.8 Install the Service Mediator

1. Change directory to `cadenusproject/software/ServiceMediator`

2. Edit the file `config/wscompile_config.xml`

3. In the `<wsdl location="http://xxxxxx:8080/ResourceMediator/...">` tag, change `xxxxxx` to the name of the Resource Mediator that the Service Mediator will connect to.

4. Create stubs for the Resource Mediator web service by running the scripts:

```
./createSTUB
```

5. Compile the Service Mediator by running the script:

```
./compileClient
```

6. Install the Service Mediator web page GUI by typing:

```
cd homepage
./install
```

7. Restart the tomcat server:

```
/home/admin/program/tomcat50-jwsdp/bin/shutdown.sh
/home/admin/program/tomcat50-jwsdp/bin/startup.sh
```

8. Open the page http://sm-hostname:8080/jsp-examples/sm/ to create a SLS and send it to the RM.