IP Management for Tele2

A N D R E A S   Ö H R V A L L

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LECS-2004-11

IP Management for Tele2

A N D R E A S   Ö H R V A L L

Examinator: Vladimir Vlassov
Industrial supervisor: Karin Ruta

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LECS-2004-11

1 Abstract

The number of computers connected to the Internet in the world is growing very fast and soon there will not be any more IPv4 addresses available. This has forced the organizations handling the IP addresses to demand much information before distributing and assigning any IPv4 addresses. Because of this, Internet operators like Tele2 spend resources for handling, organizing and distributing IP addresses.

By designing an application that handles the distribution and organization of IP addresses and by integrating it with other support systems and routers, resources can be spent more efficiently. Statistics and history can make the system automatically reorganize available resources and even apply for new addresses.

The goal of this thesis project was to design and implement a prototype of an application that manages Tele2's IP addresses.

A system integrated with many other systems must be flexible, robust and designed in a way that makes it possible to change the functionality according to changes around the system and new functionality.

During the implementation phase Tele2 added new requirements. The system had to serve the ADSL authentication servers with the customers IP addresses. This suddenly put great availability requirements on the application.

Due to the new requirements some functionality had to be delayed, but the most important functions are implemented, tested and put in production environment.

The test results were possitive, but some functionality remains to be implemented.

2 Table of Contents

2.2 Figures

3. Introduction

In this master thesis I will describe a project I have performed at Tele2 Sweden and KTH. The aim of the project was to study, design and implement a prototype of an application that takes care of Tele2's and Tele2's customers IP addresses. The application should make it easy for customers to apply for IP addresses, the workers at Tele2 to find customers, IP addresses and to assign addresses to different customers and services. It is also crucial that Tele2 get a better overview of all IP addresses and how they are used. For more details, look at the requirement section.

The goal of this introduction is that it should be possible for anyone to read even though they do not have any specific knowledge about software design. To read the rest of the report the reader must have more knowledge about Internet, software design and programming.

3.1 Tele2 AB

Tele2 is a pan European telephone and Internet service provider [16]. Right now Tele2 is expanding in many parts of Europe both with telephony and Internet services. This indicates that maintaining IP addresses will become more and more complex. In addition new telephony services such as 3G need IP addresses as well. In order to ensure that it is easy to implement the administration of IP addresses in a new country, an efficient tool is needed.

3.2 Internet and IP addresses

The Internet connects local networks of computers into a big network that contains millions of computers. To support this big network, four “RIR”, Regional Internet Registers, have been established by IANA, Internet assigned numbers authority [10]. IANA allocates IP addresses to the RIRs. The RIRs allocate those addresses to LIRs, Local Internet Registers, which typically are big companies or ISPs (Internet Service Providers). The LIRs assign the addresses to their customers and their own services and computers. Tele2 is one of those LIRs.

The number of computers that connects to the Internet is constantly growing and consists today of more then 50 million computers [1]. To ensure that you can connect to any of these computers, your computer needs a unique address. This address is today a number ranging from 1 to about 4 billion. The protocol using those addresses is called Internet Protocol version 4. Most of these addresses can not be used because all organizations connected to the Internet must be assigned many addresses that come after each other. In addition those addresses must be in the same range that other organizations nearby have. This might seem a little bit odd at first, but the reason is that when a node in the network decides where to redirect traffic that you send to someone, it can not have an “address book” that contains all 50 million computers in it. It would simply be too slow to find the right address and it would be very difficult to keep the address book updated. Instead, the nodes have different ranges that point in different directions. Only the closest node has the exact address to the computer in the address book. The result of this solution is that Internet is running out of addresses. The answer to this problem is called Internet Protocol version 6, IPv6. IPv6 has far more addresses, but it will take some time before it is implemented, and we will have to live with both IPv6 and IPv4 addresses for quite a long.

From a LIR's point of view, IPv4 addresses are difficult to work with since you have to be so careful, not “wasting” any addresses. You must always monitor that the IP addresses are used efficiently. IPv6 addresses do not have this problem, since 128 bits are dedicated for addresses, but it might be more difficult to monitor all addresses since they are so many. There are other differences between IPv4 and IPv6, but in this project we are only interested in the address space.

RIPE is the RIR for the European region [15]. RIPE maintains a big database where you, among other things, can search for any IP address in this region and find out who is responsible for it. It is up to the LIRs to maintain this database. When you are assigned a range of addresses you are obliged to report to RIPE what you are using the addresses for, and who is responsible for what address. Different LIRs have different status at RIPE. They are allowed to assign different number addresses to their customers depending on this status. If a customer needs more addresses, the applications has to pass through RIPE.

A special designed database and application can relieve the staff a lot of administrative work and help the organization to get a better overview of the IP addresses.

3 Background

3.1 Existing System

The IP ranges assigned to Tele2 are kept in separate plain text files. In those text files the range is divided into subranges and every subrange has a status – assigned or free. If it is assigned, the line also contains the name of the customer, the date of the assignment and the order numbers of the orders for that customer. Separate files are maintained for IP ranges that are supposed to be returned after a loan or a cancellation of a service. A few lines from the file could look like this:

IP range	Date	Customer	Order #	Signature
213.100.2.0/24	2002-01-12	Company Customer 1	129983	AO
213.100.3.0/24	2002-01-12	Company Customer 2	129984	AO

Today the work with IP addresses at Tele2 AB is done manually. When a customer applies for an IP address range, the application is checked to verify how many addresses the customer actually should get. After this, a suitable range is found in the text file and the customer, the staff handling the routers and the seller are informed by email. After that, an email is created that is sent to RIPE to update the information in RIPE’s database. This should be handled within a few hours after the request for IP addresses has reached Tele2.

Regularly scans are made in the order system to look after new customers who have not applied for any IP addresses.

It is not possible to get automatic warnings when more addresses are needed and it is also difficult to get an overview over the addresses used when an application should be sent to RIPE.

3.2 Requirements

Tele2 needs a tool to administrate the IP addresses more efficiently. The staff at the Registry department has put together the requirements and I will go through those in this section. The requirements will be investigated further in the Design chapter.

Users

The ”tool” must be able to handle many users from all over Europe. Different users must have different access depending on what functions the users need. The user is the staff at Tele2. To begin with only two user types are needed, but it should be easy to extend this in the future. The user types are ADMIN and USER. ADMIN can update information, but USER can only watch the information.

Database

The database must contain all IP addresses and information about the addresses, where it is located etc. It must be possible to track IP allocations through time to make prognosis about the need for IP addresses in the future.

IP applications

The (end) customer must be able to make an ”IP application” (i.e. an application for IP addresses) using a form on the web. This IP application must follow the recommendations from RIPE.

Automatic updates and notifications.

There must be automatic updates of the RIPE database when changes are made, for example when a customer is assigned new IP addresses. It is possible to update RIPE both asynchronically and synchronically. Asynchronically is done by email and is the regular way of updating RIPE. It has recently been added a function to RIPE that makes it possible to update the database via http which is synchron.

Notifications must be sent to customers and network operators with information about changes in the assignments.

Integration with other systems
The application must be integrated with the Tele2 order system called SPAN. SPAN contains customer information and all orders the customers make.

The system must be easy to expand to make automatic updates of routers and similar equipment.

Other functions
Traceroutes and pings must be possible to do on a number of IP addresses to verify that addresses are unassigned and to find loops. It must also be possible to do consistency checks against other systems.

Hardware
The system must run on a Sun Solaris machine.

3.3 IP management tools

It is quite obvious from above that in order to keep track of all addresses efficiently you need help from a database system. Those systems are often called "IP management" tools and they usually work so that you can enter an IP range and then distribute that range between different services. Often they are also integrated with a DNS system. DNS stands for Domain Name Server and is the network of servers on Internet that translates a regular name into an IP address.

There are several IP management tools on the market and a couple of them are described here together with referenses.

Commercial:
Voyager IP [17]
Nominum Management Center [12]

The commercial tools have some features that Tele2 needs. This includes databases over the ip addresses in use and well developed graphical user interface. However, it lacks the integration with the rest of Tele2's systems and they additionally have many features that are not needed. This means that additionally from the cost of buying licenses costs would be added to change and integrate the systems. This could turn out very expensive and that is why Tele2 already have decided not to use any of those systems.

Free, open source:
FreeIPdb [6]
NorthStar [13]

The open source products are facing the same problem that the commercial products have. Moreover it lacks many features that the commercial products have. However it would be possible to use the open source products as a basis for the final product, depending on what open source license they use. However, those open source projects lack so many features that it would take longer time to change them to Tele2's needs than creating a new application from scratch. It is useful to study them to get some hints and tips on user interface and other features, but the system should be implemented from the beginning.

3.4 Related work

Tele2's order system SPAN should be integrated with the new tool handling IP addresses to make it possible to see the assigned IP addresses to a customer when looking at the customers order. SPAN is basically an Oracle database with an Oracle Form used as a user interface.

The following information should be exchanged with SPAN.

- New customers with unfinished orders. For example: A customer has ordered an IP Port, but not supplied any IP application.
- The node to which a customer is connected.
  Status of the IP application. What addresses has been assigned – and when.

3.5 Technologies

This section contains description and an evaluation of the technologies that could be used for this project, together with a conclution and a desition. The two main options considered was; 1) using T2Server together with apache and PHP and 2) JSP with an application server such as J2EE. But first the motivation why not considering a stand-alone application or a non-web solution.

By looking at the requirements in the previous section a few things are noticeable before considering any specific technologies. The system must handle many users at the same time and there must be a central server to handle the database. This can be accomplished either by having a stand-alone client that all users must use, or the users can use a web interface with their standard Internet browser. A separate stand-alone client has the drawbacks such as having to be updated for all users when functionality is changed. A web interface may in some contexts lack the possibility to get all functionality needed. You would, for example, avoid placing a high end 3D game over the web interface because of performance problems. However, in this application, no such performance problems exist, therefore it is preferable to use a web interface. Because of this fact, I have not evaluated any stand-alone clients and concentrated on alternatives using a web interface for the end user.

Modern web applications use a three-tier solution [4]. This differs from the early database solutions with just two layers. Those two layer solutions provide a user interface as a front of the database (you could argue that if you only have one database that people interact with directly, it is a one-tier application). In a three-tier solution, you separate the actual application from the user interface. This separation has some benefits. For example, it makes it possible to have a web designer designing the user interface and a system designer to design the actual application. It has also big architectural benefits since you can provide different interfaces to the same application.

Below comes an overview of all technologies I have considered. I have evaluated all technologies and in the end of this section I will justify my choice of technologies.

3.5.1 T2Server

A common misunderstanding even within Tele2 is that T2Server is the name of *one* (server) computer. This is incorrect. T2Server is an in-house made middleware used in several different applications. T2Server is used for transporting messages between different applications. This way, if the support for T2Server is implemented, access to all other applications that use T2Server will be obtained. You don't have to implement support for every specific system. Many different versions T2Server are used and T2Server is constantly being developed. T2Server is designed to be small and fast. It is used in many applications in the company for example as radius server and to connect the content providers to the content billing gateway. There exists no documentation about T2server and the code is not documented, so here comes a small overview about the system, since no references are available.

T2Server is written in plain C. However, it follows certain code conventions that imitate certain aspects of object oriented languages. For example, all structs are declared in the c-files instead of the h-files and all variable manipulation must be made via get and set functions. Simple asserts are put throughout the code with “magic numbers” that verifies that the functions are applied on the right kind of struct. This is a quite weak test but it prevents some coding mistakes.

A quite extensive library of functions is available called Libsw. This library contains many functions that are commonly used including database calls and memory management. The library is also almost completely undocumented.

The central part of T2Server is the key – value pairs. Those keys – value pairs have get and set methods. At the same time, you call T2Server you also need to specify what namespace you want to set the key – value pair in. You can also specify a sub namespace and you can use the method getAll, which usually returns all possible get and set methods available.

There are two namespaces predefined. They are called ID and Session. The ID is a read-only namespace where for example identities are stored. The session namespace contains session specific information, which includes for example expiration of the current session and the client's IP number. It is also possible to define your own namespaces. This is done by designing your own c-file according to the namespace definition. This library can in turn access any other C code included in the make file. After compilation of the new library into T2Server some configuration in the configuration file, T2Server is ready to be started.

Apart from defining your own namespaces, it is also possible to define authentication modules. Those modules are similar to namespaces but here you specify other methods. The authentication modules are used to administrate the accesses of the namespaces.

T2Server has a client part that is called Libtele2. There are several different clients implemented, for example in Java and Perl. However, the best-supported one is a PHP / apache client. By extending PHP and apache with a specifict module, it becomes possible to make the calls @ap_get(<namespace>.<subnamespace>,<key>) and @ap_set(<namespace>.<subnamespace>,<key>,<value>) together with a couple of other ap-commands. Apache implements the session handling together with T2Server by using a module. Since there is no documentation for this part it is difficult to find out how to compile and configure PHP and apache without help from someone who has done it before.

However, when considering using T2Server the best choice would be to use apache and PHP since those technologies are the best supported for T2Server. Therefore I will give an overview over those two technologies next, before I consider other options.

The rest of the technologies I will describe are all well documented elsewhere. I will give a small overview of them together with good references for further reading.

The Apache Web Server is the most popular Web server because of its stability, efficiency and portability [4,9]. Apache is maintained by the Apache Software Foundation and the web server is completely open source. The opinions about open source are quite divided and a complete essay could be written about the benefits and drawbacks of open source versus commercial applications. The fact is that in this case (for T2Server) it was crucial to modify the code to integrate the session handling, so open source must be considered as a good thing.

It is possible to run Apache Web Server at many different platforms and it is also possible to use different dynamic languages like JSP, Perl and PHP with the help of plugins. A SSL version is also available so that it is possible to encrypt the data sent between the web client and web server. SSL stands for “The Secure Sockets Layer” and it uses a variety of symmetric techniques to send data to and from servers and clients [9]. Keys for the symmetric encryption are exchanged during the handshaking (which is done with the help of asymmetric encryption). For more information, installation and support of Apache, please visit Apache's homepage [1].

PHP Hypertext Preprocessor is also open source and is one of the most popular server side scripting language available [4,14]. PHP differs from other similar scripting languages since it was explicitly written for generating dynamic Web pages. The fact that it is open source made it quite suitable for developing support for libtele2 and the connection to T2server.

3.5.2 Java Enterprise Technologies

I have so far described most of the technologies that I would use if I use T2Server. However, I do not have to use T2Server at all. I will discuss a couple of alternatives, including JSP, and JSP together with the J2EE application server.

Java servlets are server side Java applications that are used for generating web pages. If you know how to write Java applications, it is easy to learn how to write Java

Servlets. An extension of the Java Servlets is JSP – JavaServer Pages. JSP is more similar to PHP in the sense that you can write Java code to create dynamic content inside a regular xhtml document. It is also possible to extend JSP pages with dynamic functionality in XML tags. This makes it possible for a developer who does not know about Java to access databases etceteras [4,11]. The strong advantages of JSP and Java servlets are that they use a syntax that is well known and additionally they are completely portable to a number of platforms. You get access to the complete Java API of well-tested functionality.

In 1998 Sun released the first version of Enterprise JavaBeans [5]. Enterprise JavaBeans or EJB is designed to supply software designers with a powerful tool that handles transactions, threads and load balancing. By specifying contracts between different interfaces towards the end user (for example through a JSP page) and the interface towards the database EJB basically handles the rest for you. If you need more advanced data processing, it is possible to extend this in EJB. EJB has been extended over the years and it is now possible to use EJB with J2EE 1.4.

The biggest commercial alternative to Java and T2Server comes from Microsoft. However, if you choose one of technologies from Microsoft – the old .asp or the new platform .NET portability is compromised. The fact that this application must run on a Sun/Solaris machine
Microsoft is not an option.

3.5.3 Databases

No matter if I choose Java or T2Server, a database is needed in the third tier of the application to permanently store the data. There are several options here. Other similar applications at Tele2 use Oracle and a few uses MySql. Libsw contains support for both. Oracle contains much more functionality than MySQL and since Tele2 has licenses for Oracle, my choice falls on Oracle. Especially the foreign keys, transaction support, trigger and stored procedure functionality is stronger in Oracle. However, this might change in the future, so I will try to keep my application as independent as possible from the database.

3.5.4 XHTML and CCS2

No matter what technology is used to generate web pages I need to generate web pages that are well-understood and easy to read by web browsers and humans. HTML is widely used on Internet to define documents. It has been extended many times from the original specification including for example multimedia capabilities [18]. All extensions in different directions have made interoperability between different web browsers difficult. One of the reasons why XHTML was introduced was to address this problem. Since XHTML is completely compatible with HTML there is no reason not to use XHTML. However, when making dynamic web content, which basically mean that you generate the XHTML with the help of a program, it is important to be careful all the time so that all code generated is compatible with XHTML.

By separating the structure of data from the document's layout, the document becomes easier to maintain. This can be done with CSS or CSS2 [4]. CSS stands for Cascading Style Sheets. All modern web browsers support CSS and the alternative – to mix structure of data and layout - is not a good way of designing a web page. I will use both CSS and XHTML to make the pages easy to maintain and to ensure that as many as possible can use them

3.5.5 Evaluation

I implemented three small prototypes to try the functionality and to get a feeling of the different environments. The three different configurations were T2Server + PHP + Apache, JSP with a Servlet and EJB. The prototype was an XHTML form and a table in a database. It was possible to view the table, add an element and delete an element from the table.

It was a little problematic to get T2Server to work properly, but with some help from other developers at Tele2 it worked fine. I implemented a namespace that used a connection to an oracle database. It was quite easy to find out how Libsw worked by looking at other code in other applications that used the same functions. It would however be much better if good documentation was available. Functional languages can be difficult to use when you are used to use object orientation. However, this is probably more dependent on what you are used to work with then the concepts of functional or object oriented languages.

A JSP and servlet version was easy to implement. The JSP executed set and get on the servlet which used JDBC towards the database. No problems occurred and since I have used Java before it was easy to implement. I used the Apache Tomcat stand-alone web server and JDK 1.4.

The EJB took longer time to get working properly. When all interfaces were in place, it took a while to remove all compilation errors. Since apache does not provide any EJB implementation I used the Orion WebServer implementation.

It is not possible to compare the performance between those small implementations but it would be difficult for the Java implementations to beat the performance of T2Server's plain C. It is even more difficult to say something about how long time it takes to develop in the different environments[1]. The same thing applies for how correct the program becomes.

If this project were not related to Tele2 I would choose for EJB. The reason is that it provides strong support for transaction management and it is easy distribute over many nodes if any performance issues would occur. Besides, it provides object orientation that I believe is a good thing. The different views of object oriented programming are quite personal, but I prefer it. It could be argued that T2Server has higher performance since it was written in C. This is probably correct but the performance requirements for this application is not very high. It is unlikely that this application will have more then 15 concurrent users and no heavy calculations are needed.

However, the fact that many other applications at Tele2 use T2Server for other similar applications, the cost for maintaining the application would be lower if it looked similar to other applications. This fact is stronger than the earlier statements and therefore I choose to use T2Server together with Apache and PHP.

4 Design

Inspired by RUP [8] I choose to use use-cases in the design phase. However, I chose a variant of use cases that is based on use cases from [7]. After those use cases are analysed I have made the database and system design that follows in the next subsections. The use cases follows this template.

A. CHARACTERISTIC INFORMATION
  - **Goal in context:**
  - **Scope:**
  - **Level:**
  - **Preconditions:**
  - **Success end condition:**
  - **Failed end condition:**

B. MAIN SUCCESS SCENARIO
  1.

C. EXTENSIONS
  1.

D. VARIATIONS
  1.

E. RELATED INFORMATION
  - **Priority:**
  - **Performance target:**
  - **Frequency:**
  - **Superordinate use case:**
  - **Subordinate use case:**
  - **Channel to primary actor:**
  - **Secondary actors:**

F. SCHEDULE
  1. **Due date:**

G. OPEN ISSUES

All use cases can be found in the appendix. Additionally all use cases will be followed up in the evaluation section.

4.1 System design

In the evaluation of the technologies above I decided to use T2Server together with Apache and PHP. This indicates that the system will be a so-called three-tier system. However, the user interface is divided into two parts. One part is the PHP and the other part is the XHTML and CCS (and maybe some Javascript), that is executed on the client machine.

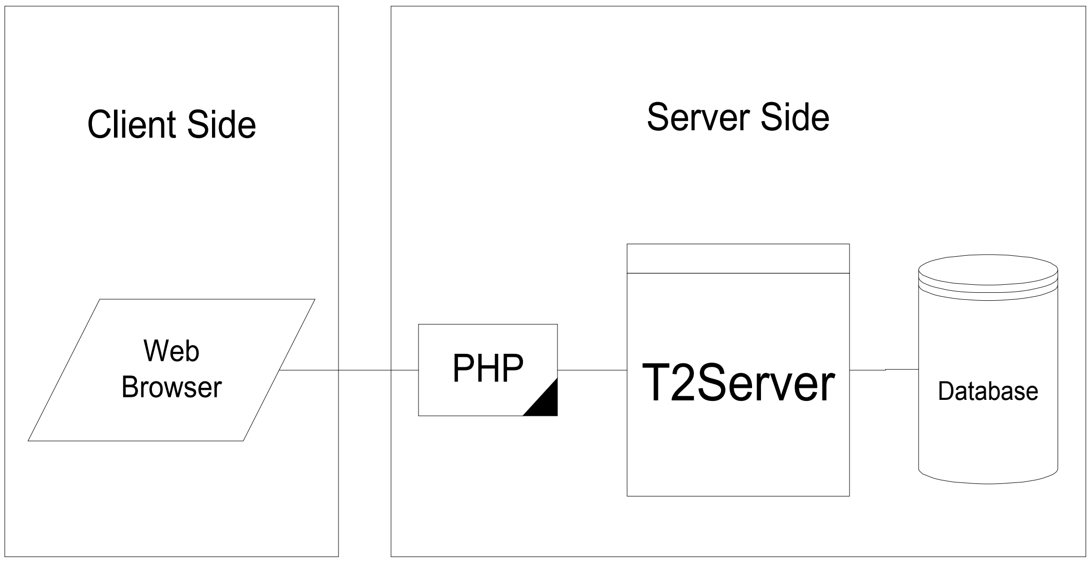


Figure 1: Overview of the system.

There should not be any application functionality in the PHP-layer since it should be possible for another application to call T2Server directly and access all functionality. This indicates that no form validation should take place in the client side, because this would duplicate business logic and that should be avoided.

The goal for the plugins of T2Server should also be not to duplicate any information and also to keep the code as modular as possible. All database specific code should be kept in one place and the validation of fields in another. The validation criteria's should be possible to change via configuration variables and not encoded hard into the code.

The following diagram shows all functional parts of the system and with what interfaces they are communicating.

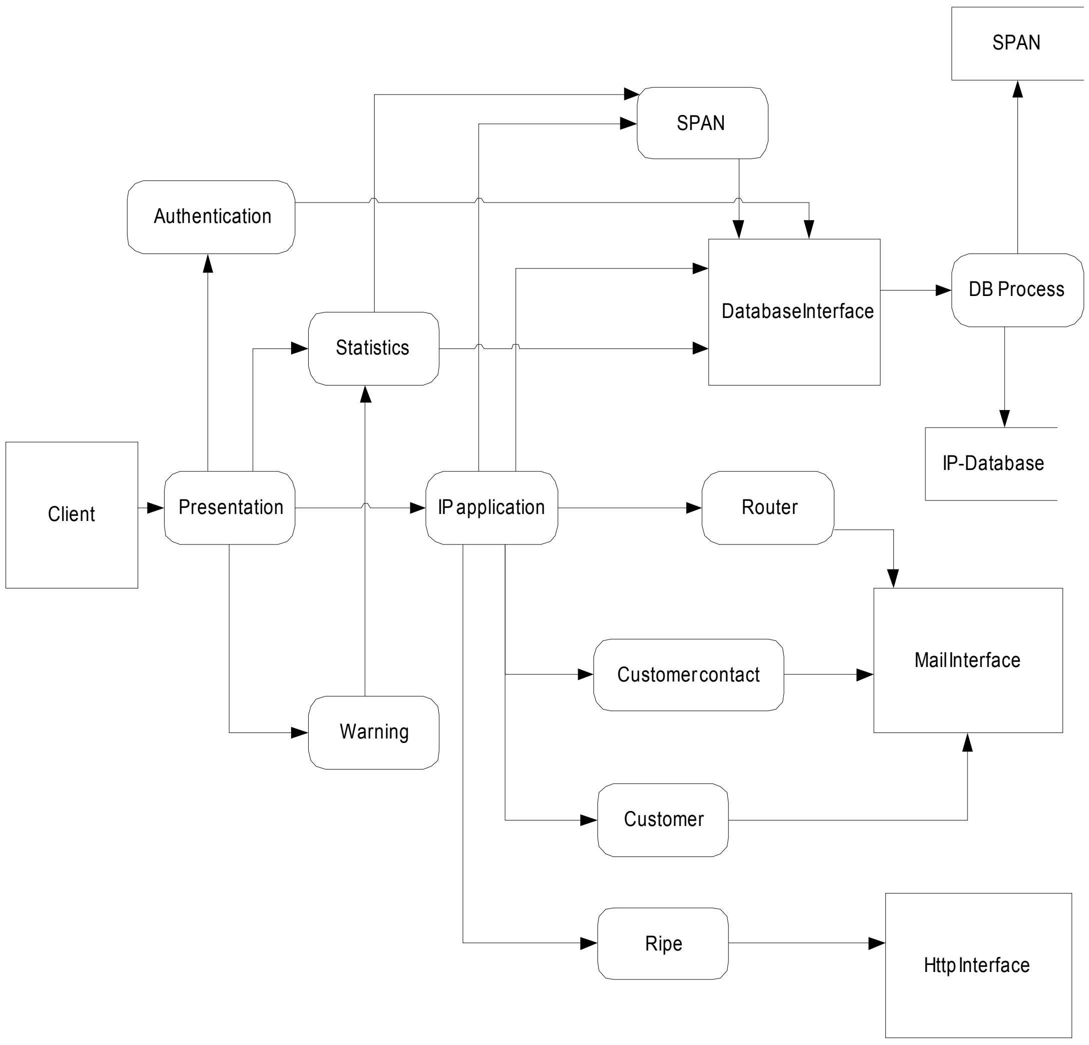


Figure 2: Detailed view of the system.

It should be noted that all round boxes represent plugins or parts of plugins to T2Server. The square boxes show how the different plugins are communicating with their respective system. This diagram replaces a class diagram, and the reason is that C does not support object orientation and therefore it is confusing to insert a class diagram. The figure shows the client, who is using a web browser to access the presentation layer. The presentation layer is talking with four different parts of the application, which are all located in the T2Server, Authentication, Statistics, Warning and IP application. The IP application is the core part of the system and is handling all IP assignments. When an IP assignment is to be performed, it performs changes toward SPAN, Routers, the Customer contact, the customer and ripe. Depending on what kind of transaction should be performed, different number of those systems will be modified. The square boxes show the different kind of interfaces that are needed. By writing the interfaces as configurateable as possible it should be possible to only implement them once, but use them for several different systems.

4.2 Database design

This section contain some notes about the choice of representation of data. The most important table is the IP table. I chose to use IP and the Prefix instead of a “from IP” and “to IP”. This choice can certainly be discussed. The advantage of using the second alternative would be that RIPE is using this way of representing IP’s and that a search would result in a more accurate hit. The first alternative – the one I chose has the advantage that it is the same way that IP addresses are represented in routers. This makes it easier to maintain clean routing tables. I will have database constraints to ensure that it is not possible to insert data with incorrect values. It will not be possible to insert the same IP two times for a given time period.

Another detail to look closer at is the representation of the personal objects. This is inspired by the RIPE way of representing such objects and this also makes it possible to extend the system with more RIPE database types in the same model.

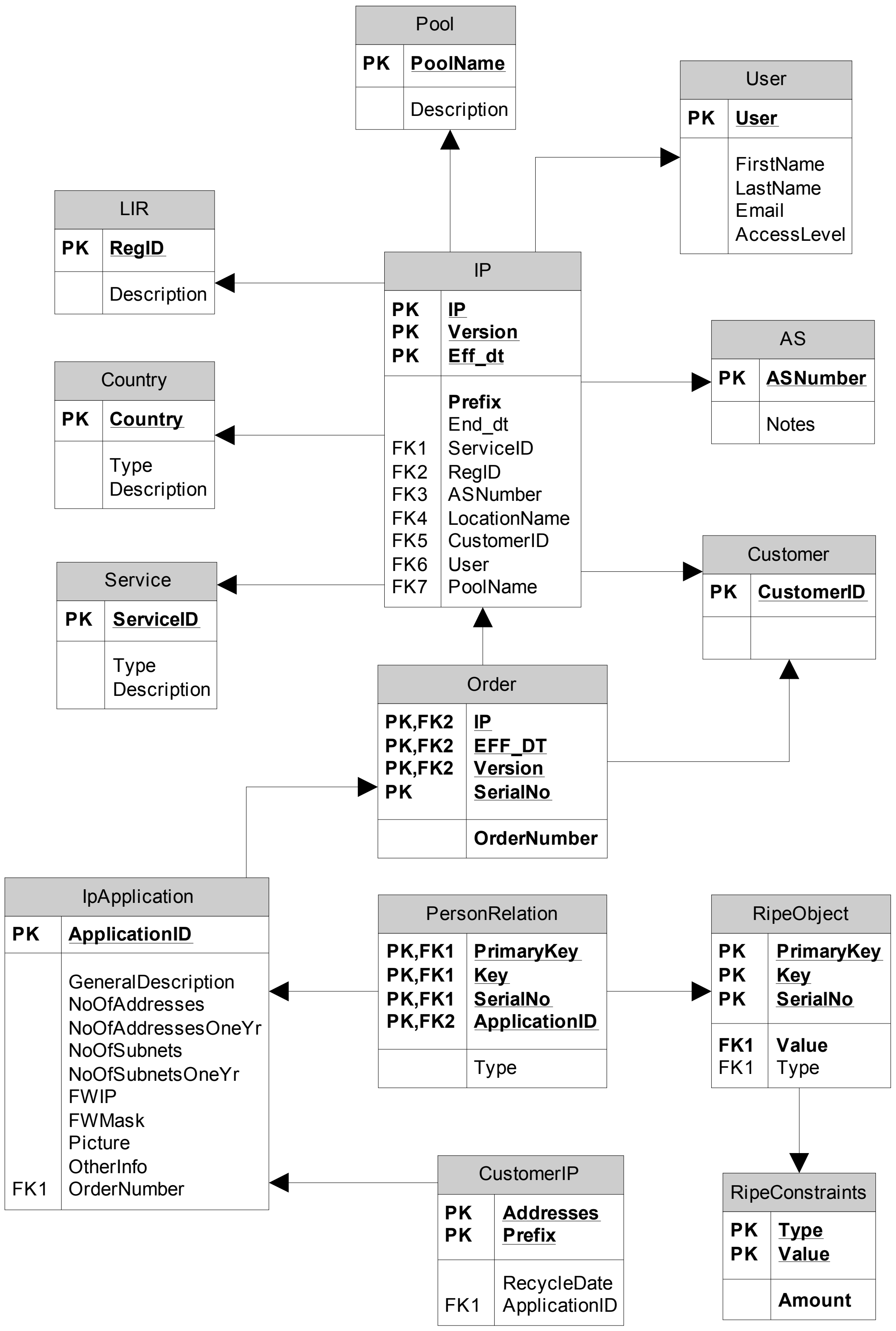


Figure 3: Database class diagram the system.

5 Method

5.1 Changes from the original design

The system was names IMS, which is short for [Tele2's] IP Management System.

It is certainly not good to change the design of a system during the implementation phase. However, it is not always possible to avoid when surrounding systems change or when new functionality is required by the surrounding organization.

A new Tele2 service had to be implemented. This service provided fixed IP addresses to ADSL business customers. This means that every time a customer log on, they will be assigned the same IP address. The project formed to implement this service wanted the IP management system to handle the IP address assignments for the fixed IP address systems.

This new feature puts a couple of important new requirements on the system.

Radius Interface
The radius servers is handling the authentication and authorization of ADSL customers. This server must have an interface against the IMS. Tele2's radius servers are clients to T2Server which means that the communication already was in place. However new implementation is needed to provide integration with the T2Server that handles the regular ADSL customers.

Response times
It is crucial that the IP address lookup will not delay the login time for the end customers. There will unavoidably be a longer response-time, but it must not be noticeable for the customers.

Availability
If the IMS has downtimes ADSL business customers will not be able to use the service. This means that the availablity must be very high.

Those changes also meant that the implementation of other features had to be delayed. Moreover, the system had to moved to another network and a firewall had to be opened in order for the system to be accessed by radius servers and at the same time access internal Tele2 systems. A completly new server was set up for this reason.

5.2 General problems

The biggest problems during the implementation phase were to get practical things together. Compilation problems with Apache's T2Server support and different system environments variables took some time to sort out. Additionally there was administrative problems in the beginning with access to internal Tele2 servers and services.

5.3 Implementation choices

The core part of the system is to handle IP addresses. The choice in my design was to keep track of the IP addresses by “base” IP and a prefix, instead of entering all ip addresses seperately into the database. For example, instead of entering the IPv4 addresses 130.244.10.0, 130.244.10.1, 130.240.10.2 .... 130.240.10.255 I enter 130.240.10.0 and the prefix 24. “24” stands in this case for the number of significant bits in the IP address.This way of representing many consequtive ip addresses is widely used, especially in routing. Thousands of entrys in the data structures are saved by doing this. For more information about this, please see design chapter.

The problem that occurs when choosing this way of representing IP addresses is to ensure that in a given date one IP address only occurs in one range. In order to ensure this and that effective dates are before end dates, I choose to use database constraints. PL/SQL is the scripting language that oracle use and it was not complicated to create those contraints. It could be argued that those kind of constraints will not be needed if you use the “right” database model. However, it is easier to manipulate a table that you understand than to use many abstract tables.

When complicated operations are executed against the database, stored procedures should be used. The IMS uses stored procedures for everything except the simplest inserts and selects. This way the most performance is accomplished and the sql generation parts of the application could be kept very simple. The SQL that should be used towards stored procedures, database views and tables are generated from configuration files. The original plan was to generate stored procedures, views, database indexes etc from those configuration files as well. This way the logic about the database would be kept in one place only and missmatchs would be avoided. However, this functionality had to be delayed due to time contraints.

The first implementation of functionality in a T2Server namespace was an insert into a table. This was not so difficult to accomplish, and work started to implement more namespaces in the same manner. Howerver, it turned out to be very difficult to keep track of all SQL, and excessivly boring to do the same thing over and over again. Therefor another approach was used, based on configuration files.

The configuration files contain all information about all tables and their fields. Additionally it contains similar information about all stored procedures. Almost all functionality that is needed for the T2Server is to perform select, insert or update a table or execute a stored procedure in the database. Additionally to this some functionality is needed for access control and verification of the correctness of inserted values. Therefore two modules were implemented. One for database access and one for data verification and access control.

The first module is the data verification module. It looks up in the configuration file what kind of value that has been set and verifies that the user has access to set it. After that it checks that the value is of the correct type and formats it so that it becomes easy for the sql generator to use. For example, if it is a string it changes STRING to “STRING”.

The second module is the sql generator. It goes through all incoming data and generates SQL for oracle. When all data is gathered it executes the SQL and makes a result set available. If the SQL is a select, then it is possible to go through all columns and all rows by simple commands. It should be fairly easy to convert this module to access any other kind of database – for example MYSQL. You only have to edit this module for this kind of conversion, and of course you must solve the problem with stored procedures written for Oracle.

Both those modules are written in a way so that they can be used in any other application that needs access to Oracle databases.

Only one type of namespace is needed for all functionality. Since almost all operations are passed on to Oracle the only thing the namespace need to do is to initialize the two modules described above. The same namespace is started once for every table and procedure with the name of the table of procedure as argument. If new functionality is needed it should be by extending the configurability of the namespace and corresponding modules. If functionality is needed that conflicts with other functionality, or is so different that it should not be integrated with the other parts, then a plugin should be loaded that take care of this. This kind of plugin will be needed when the integration with for example BGP or RIPE.

In order to spend as much time as possible on the T2Server parts, the design of the web interface was inspired by an open source web design repository. I modified the original design to become more modular and suitable for php to dynamically generate menues. Special php functions was created for placing menues in different locations. By calling those functions in different order, with different arguments the menues show up in the corresponding location. The menues are placed in locations where the number of menu alternatives and buttons does not affect the rest of the content. This way it is possible to just add and remove menues in a php file that later is included in the corresponding page's php file. The colors choosen was inspired by the original designer. Light blue in different nuances gives a feeling of warmth and wellbeing. For some reason it placed the main menu above the topic of the page when using Opera. No solution to this problem has been found since the page looks perfect on Mozilla/Netscape and Internet Explorer. The first page can be seen in Figure 4 (Mozilla) and Figure 5 (Opera).

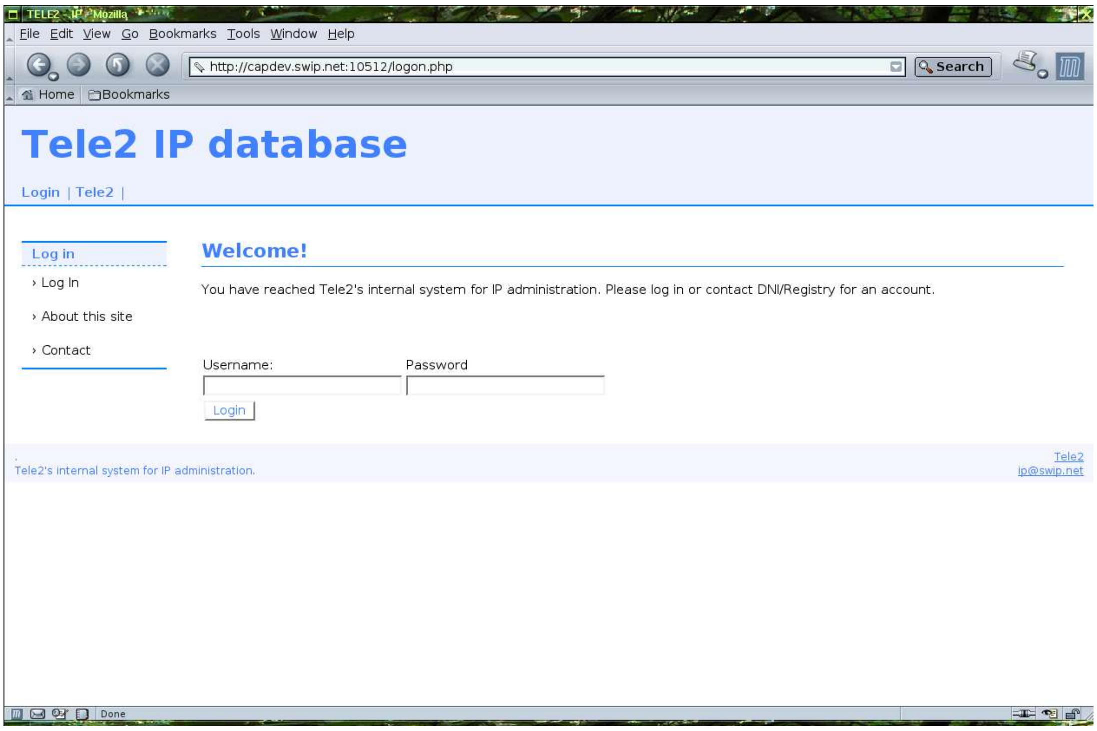
The pages was all tested in several different browsers including Internet Explorer, Netscape/Mozilla and Opera. Internet Explorer version 6 does not support all css2 features, for example some color changing when moving the cursor over a form field. Additionally there was a problem with Opera. Opera was placing the main menues above the title. No documentation of this problem could be found and since it was not so serious it was left alone. Additional to those browsers, all pages was tested in Lynx. It must be possible to administrate this system and access all functionallity via a kerberos telnet or SSH session.

Since the site is based on T2Server, it is also easy to make a stand alone shell program to perform some simple and often used functionality. This is useful when you fast need a new IP address for a backup server or something similar.

5.4 Implementation

In total 54 php files were implemented. Most of the functionality is concentrated into 10 php files and the rest are files used for configurating particular files using the other php files. Four namespaces using two modules, written in C, takes care of the database connections and other backend functionality described further down. The code written for this particular application consists in total of about 9 500 lines of code.

Additional to the php files that handles the menues and graphics a php file was created for handling the forms. XHTML forms is used for the user to enter data. Two different types of forms was found. One kind is used for changing, updating and deleting data in the database. The other kind is used for searching for data. A php file was created for handling both those kinds of forms. The same php file also takes care of the result of the post and sends it to T2Server. By creating array's of variables containing information about each field and pass those to the page generator all logic can be seperated from the data. The arrays contain information like how big the field should be, what kind of field it should be, what should be done with the data (send to T2Server or display it for the user for example), dependencies to other fields etcetera. If the form should be a search function, it also has values for links and presentation format of the result.


TELE2 - IP - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://capdev.swip.net:10512/logon.php
Search
Home
Bookmarks

Tele2 IP database

Login | Tele2 |

Log in

› Log In

› About this site

› Contact

Welcome!

You have reached Tele2's internal system for IP administration. Please log in or contact DNI/Registry for an account.

Username:	Password

Login

Tele2's internal system for IP administration.

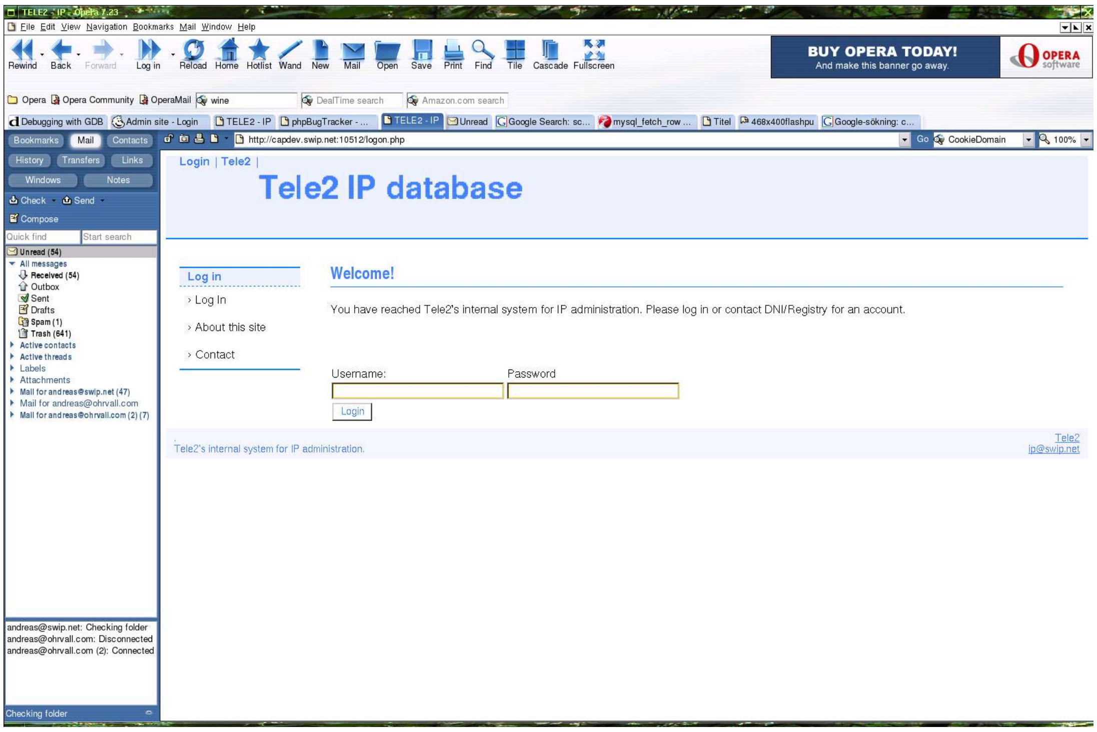
Tele2
ip@swip.net

Done

Figure 4: Screenshot of the user interface with Mozilla.

This solution makes it possible to set up a complete page by setting some arrays and call the function together with some text. Instead of defining variables both for the form and for the recieving function it is only needed to define it once. This reduces the probability or errors significantly. All those requests are quite dumb. All they do is forwarding messages to T2Server. This way the logic is reduced to a minimum in the php layer. PHP should only be used for giving the user a userinterface to use.

No verification logic of the data is placed in the php file, it is all placed on the T2Server side. This way the same verification code can be used even if the namespace is accessed directly, without passing through Apache. In the section where the T2Server implementation is described this will be explained further.


TELE2 - IP - Opera 7.23
File Edit View Navigation Bookmarks Mail Window Help
Rewind Back Forward Log in Reload Home Hotlist Wand New Mail Open Save Print Find Tile Cascade Fullscreen
BUY OPERA TODAY!
And make this banner go away.
OPERA software
http://capdev.swip.net:10512/logon.php
Bookmarks Mail Contacts
History Transfers Links
Windows Notes
Check Send
Compose
Unread (54)
All messages
Received (54)
Outbox
Sent
Drafts
Spam (1)
Trash (641)
Checking folder

Login | Tele2 |

Tele2 IP database

Log in

› Log In

› About this site

› Contact

Welcome!

You have reached Tele2's internal system for IP administration. Please log in or contact DNI/Registry for an account.

Username: Password

Login

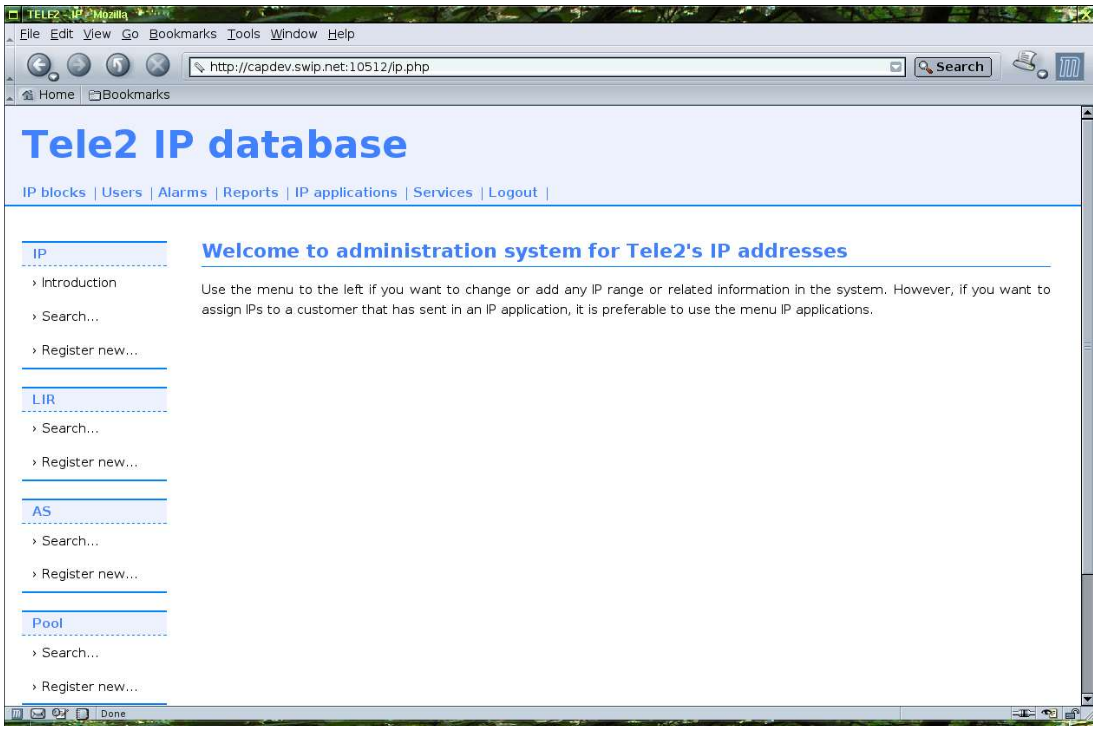
Tele2's internal system for IP administration.

Tele2
ip@swip.net

Figure 5: Screenshot of the user interface with Opera.

A couple of complicated php files were created to handle all kinds of dynamic xhtml forms, menues, graphics and other functionality. This way a new page with many different forms accessing different namespaces and generating graphics could be configured in one place, by including other php files. The interaction with T2Server is kept in one place and this would make it easy to reuse the code in a different place. The first experiences showed that it was very likely to make mistakes when the same variables had to be written twice everywhere. One variable for the form and one for the receiving function. Besides, it is very boring to do that kind of programming. Bored programmers tend to make mistakes.

The userinterface has a main menu that is always accessible from the top of the page. Submenues to the left that varies depending on what main menu that has been chosen makes it possible to reach any function in the whole application in three clicks, no matter from what location you start. It is possible to see the first page and the main menu in figure 6.



Welcome to administration system for Tele2's IP addresses

Use the menu to the left if you want to change or add any IP range or related information in the system. However, if you want to assign IPs to a customer that has sent in an IP application, it is preferable to use the menu IP applications.

Figure 6: Screenshot of the welocome page of the graphical userinterface.

The main menu contains the following entrys;

- IP blocks

  Under IP blocks you can find the core functionality including adding and removing IP addresses. You can also manage IP pools, LIR's and AS's. An example of the page to register a new IP block can be found in Figure 7.

- Users

  You can add or remove users. It is also possible to change passwords and access levels.

- Alarms

  For future use. It will be possible to create alarms so that emails or snmp traps are generated when the number of IP addresses in a particular pool is too low.

- Reports

  It is possible to generate graphs of the historical usage of IP addresses.

- IP applications

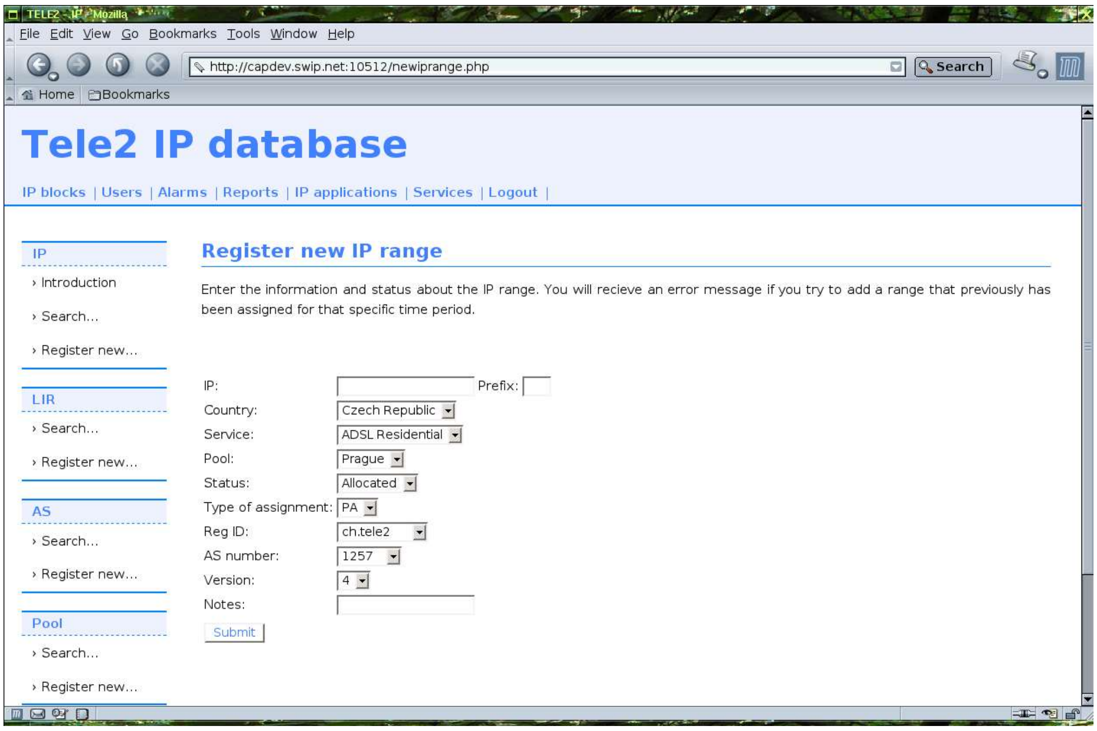
  This IP application section contains the possibilty to look up SPAN order numbers and their IP addresses. In the future it will also be possible to see all incoming customer IP applications.

- Services

In the Services menu it is possible to administrate different services that the IP addresses are associated with.

- Logout

  Log out of the system.



Tele2 IP database

Register new IP range

Enter the information and status about the IP range. You will recieve an error message if you try to add a range that previously has been assigned for that specific time period.

IP:	Prefix:
Country:	Czech Republic
Service:	ADSL Residential
Pool:	Prague
Status:	Allocated
Type of assignment:	PA
Reg ID:	ch.tele2
AS number:	1257
Version:	4
Notes:	

Figure 7: Screenshot of register new IP range.

Here is an extract of the php code that generates the page used for registering new IP addresses shown in Figure 7. Several dots (....) after eachother means that the code is shortened down.

```
<?php
  include ("inc/sessioncheck.php");      # Used for checking if
                                         # user is logged in
  include ("inc/formhandler.php")        # Functions to create forms
  include ("inc/designhandler.php");     # To print menues and layout
  include ("inc/scripthandler.php");     # Including Javascripts..
  $NSNAME1 = "IPAIP";                    # Namespace to call
  $today = date("Ymd");

# Here follows an array of all fields that is shown in the figure
$fields1[] = array( 'type' => 'text',       # XHTML type..
                    'name' => 'IP',         # Name of field
                    'size' => '20',         # Size of field
                    'tag' => 'NORMAL',      # Tag for style
                    'maxlength' => '40',    # Max char length
                    # What should be shown before field
                    'prefix' => '<tr><td>IP:</td><td> ',
                    'errname' => 'UNUSED', # How to present errors
                    'preerr' => ''         # Show before error
                    'pk' => 'Y',           # Is it a primary key
                                           # Used for lookups.
                    'suffix' => '</td>'); # Shown after the field

                    ..... <The rest of the fields> .....
        # If the next stage is "process", the fields are passed into
        # a function that communicates with T2Server, and get a
        # result back.

  if (isset($_POST['stage']) && ('process' == $_POST['stage']))
  {
    $processresult = process_form
                      ($fields1,$NSNAME1,"Set","Commit","Search");
  }
                    ..... <all text shown> .....
        # Print the form header.
        print_form_header($errormessage,"post",$NSNAME1);
        # And the fields are printed if the next stage was not
        # process or if there was any errors (which are passed into
        # the printer functions that prints the error messages in
        # the correct place). Data for javascript is
        # returned. If the process was a success, the result is
        # printed instead.
        $scriptoutput = print_form_fields
                  ($fields1,$NSNAME1,$processresult, "Search");
        # And the form footer
        print_form_footer();
                    ....... <the page footer> .....
?>
```

Here follows a small extract of code from the IPAIP namespace plugin of T2Server that handles the incoming php requests that will end up in the database. Depending on what command is received, the sign's in the SQL generator is set to different things.
"p" is a pointer to the command. "p_key" and "p_value" is the pointers to the incoming key and value. "Ipadb" is the module that generates SQL.

```
if (SW_SequalsCN (p, "earch",5))
      {
      if (SW_SequalsCN (p, "earchMin",8)
                  && !SW_SequalsCN (p, "earchMinMax",11))
      {
          IPAdb_SetCriteria (self->ipadb, &p_key[9],">=",p_value);
      }
      else if (SW_SequalsCN (p, "earchMax",8))
      {
          IPAdb_SetCriteria (self->ipadb, &p_key[9],"<=",p_value);
      }
      else if (SW_SequalsCN (p, "earchLike",9))
      {
          IPAdb_SetCriteria (self->ipadb, &p_key[10],
                                          "like",p_value);
      }
      else if (SW_SequalsCN (p, "earchMinMax",11))
      {
          IPAdb_SetCriteria (self->ipadb, &p_key[12],"<=",p_value);
```

```
                              IPAdb_SetCriteria (self->ipadb,
                              &p_key[12],">=",p_value);
        }
        else
        {
            IPAdb_SetCriteria (self->ipadb, &p_key[6],"=",p_value);
        }
```

The graphs of the IP assignments is generated with the help of the php extention GD. GD is used for generating pictures by sending information to it what to print in what part of the picture. For example you can tell it to write a solid green box starting from point (45,34) to (67,100). This is a quite difficult way of printing graphs, so a free utility called PHPrint is very useful. PHPrint can be told to print a particular kind of graph and together with some data it uses GD to print the graph.

It was difficult to set up the character sets and conversions properly. T2Server is in the default configuration expecting iso character set from php, and then it converts it to utf8 in the database. When T2Server converted the characters back for the php to view in the web browser it had transformed into garbage. In the database was, instead of the utf8 code '¤X', 'CX', where X is the “correct” utf8 code for the swedish letter. It took a long time to resolve this problem. The problem was that oracle was expecting a regular 7 bit ascii as input, and when T2Server sent 8 bits UTF8 code it simply dropped the most signifcant bit. 'C' is exactly 128 slots before '¤' in the 8 bits ascii table and that is the reason why 'C' was showed instead of '¤'. The solution was to set an environmental variable so that oracle understood that the incoming data was utf8. Since most modern web browsers already handles utf-8, the conversion in T2Server was completely deactivated.

The implementation of IPv6 support in the IMS is not complete. All functionality for administation of IPv4 has not been implemented in a IPv6 version yet, but the basics are implemented, and it will not be difficult to implement this for IPv6 later. The interaction with the Tele2 order system SPAN had to be delayed due to time contraints.

6 Analysis

6.1 Evaluation

The old text files was migrated completely to the new IMS by a php script written for this purpose. For a couple of weeks several users were using both systems in parallell to evaluate and test the new system. This was done in a quite early stage when the system still were quite unstable and lacked several functions. However, it was still better than keeping the files in plain text file.

There were a couple of bugs that ware corrected immediately to make sure that the data was kept consistent. Lots of manual SQL lookups and comparisons were made and compared with the old system to find any flaws in the core handling of IP addresses.

All use cases that was found in the design phase has been examined and followed up together with the staff at Tele2. The result can be found in the next chapter. Since the work is still going on, this could be considered a snap shot at the current state of the development.

In order to test the part of the application that had to serve customers with very high availability requirements, a load test that was developed for radius servers in another project, was used. The load test gave the result that even under heavy load, the system responded in a about 8 milliseconds. This is about twice the time it takes for the system to respond without the connection to the IP management system, and the reason for this is probably that there has to be two database lookups per requests instead of only one in the old system, where there was no need to lookup any IP addresses. It is not possible to make the database lookups concurrent since data from the first lookup is needed in the second.

6.2 Analysis

The response from the users was very positive. The biggest problem with the old text files was the risc for data corruption. This has been reduced to a minimum in the new system. Additionally the users find it very comfortable that the IMS can handle all the IP prefix calculations automatically.

USE CASE 1 – New IP applications

This use case would provide a user interaface for the end customer to make an application for IP addresses using a web browser. The basic functionality is implemented. The tables are designed and implemented and it is possible to assign IP addresses to customers in all different kinds of variations. The SPAN update functionality is implemented to some extent, but it is still missing some functionality that is needed in SPAN. The routers are not updated automatically, and there is no plans to do this in the near future.

It was decided to move the actual user interface to the regular customer user interface. This way the user interface would be more integrated with other parts of the Tele2 homepage. This is not yet implemented, but according to the current plan, it will be integrated during 2004.

USE CASE 2 – Alarm when more addresses soon are needed

It is possible to lookup how many addresses that are assigned and draw graphs of it manually. However, it is not possible to configure any alarms yet. All is set up manually.

USE CASE 3 – Recycling of IP addresses

It is possible to recycle IP addresses. However there was a problem that occurs when an IP address is put in quarantine. When should the IP address range be joined with the range next to it? If an address range is put in quarantine and the range next to it is in free. It can not be joined right away since the other range is in status free. But maybe the other range should not be used unless really necessary, since if it is not used until the quarantine range is freed, then we get a bigger range free. This is still an open issue. No quarantined ranges are joined with any other range for the time being, however this will be changed in the future when a suitable solution to this problem shows up.

USE CASE 4 – New IP allocation

This use case is implemented as specified. No particular problems occurred.

USE CASE 5 – Deallocation of IP address ranges

It is possible to change the status of the IP addresses to unused. It is not possible to completely remove them from the system, since this might cause data inconsistencies. It is quite rare that this action needs to be taken, so it does not have high priority to be solved.

USE CASE 6 – Move of address range

It is possible to move addresses between different customers, statuses, countries, services and pools. The only restriction is that it has to be done on all sub ranges that is assigned in the system. This might cause problems if there for example are a lot of different customers in one range that needs to be moved into another pool. It is rare that this problem comes up, so at this time this has been solved by manually writing SQL. This help function will be implemented in the future.

USE CASE 7 – Consistency checks

There are consistency checks implemented so that you can verify a particular range against RIPE and routers. Those are not automatical yet, due to time restrictions.

USE CASE 8 - History

It is possible to get graphical reports out of the system. You can make a search, specify the criteria's you need and generate a graph. More advanced statistical data and predictions of the future is not implemented, and it will probably never be implemented.

6.3 Results

Constant user tests made the development of the product easier. The fact that the users had plenty of time and knowledge made it easier for me to meet the requirements. Any flaws in the design could easy and fast be corrected.

The goal of the load tests was to ensure that the extra functionality for the radius servers to lookup IP addresses in the IMS were that it would not be noticeable for the end user. Even under heavy load the extra time it took for the end users to log in was only 4 milliseconds. This is not considered to be noticeable and the requirement was met.

7 Conclusions

In this thesis I have presented a design and a prototype for an IP management system. An IP management system is needed because of the complexity to keep track of IP addresses in an organization of the size of Tele2.

The developed product is useful. This has been proved since it is already in use. It meets the requirements to some extent, but it lacks some functionality. This functionality will be implemented in the near future.

It is very satisfying that the product is already in use, and that the response has been possitive.

In a more general perspective a couple of more conclusions could be reached. The most important is that it is crucial to use technologie that is easy to integrate with other applications in the same organization. In this particular case it would have taken at least ten times more work to integrate the application with the Radius servers if the IMS would have been built on Java technology. Additionally it would take much less time for a new developer to understand a technologie that is similar to what he is already working with.

Another lessons that was learned was the fact that you should try to avoid to duplicate code. Do not write two SELECT SQL statements that look similar to eachother. It is much better to write one function doing the work with different arguments.

Configurationfiles is another powerful tool. By using configuration files for every possible reason it can save a lot of work if something changes. For example if a field in the database change from type INT to type VARCHAR, then it is only needed to change the configuration file. In the first approach that I used it would have required a code change and recompilation of the program. To just change the configuration file can save many hours of work in the long run.

A lot of time was spent on the analysis. It could certainly be questioned how well the time defining use cases and similar activities was spent. The problem was when the requirements changed the use cases became useless. Probably in this project it would have been more efficient to spent less time on the use cases and make a few small protypes instead. This way more code would have been thrown away since it wouldn't fit the users, but more functionality would hopefulle have been implemented in the end. Of course the design phase should not have been completely removed, but it could have been shortend a lot.

8 Future work

There are certain functions that will be implemented in the IMS soon . The most intresseting is to implement a BGP client that can listen for new IP's directly from the network. This way alarms of inconsistency in the database can be detected and new kinds of reports can be generated. The requirements that had to be delayed will additionally be implemented in the near future.

It is also interesting to look at how the IMS could be integrated with different kind of DNS services.

There is quite much more work needed with the implementation of IPv6 support in the IMS.

9 Abbreviations

3G: Third generation mobile phone system

ASCII: The american standard code for information interchange. A character set with 127 different letters, numbers and symbols. Compare with UTF.

ADSL: Assymetric Digital Subscriber Line

BGP: Border Gateway Protocol

CSS: Cascading Style Sheets

DNS: Domain Name Server

EJB: Enterprise Java Beans

HTML: HyperText Markup Language

IANA: Internet Assigned Numbers Authority

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

IMS: [Tele2] IP management system

ISP: Internet Service Provider

J2EE: Java 2 Platform Enterprice Edition

JSP: Java Server Pages

LIR: Local Internet Registers

PHP: PHP Hypertext Preprocessor

PL/SQL: Procedural Language extention to SQL

RIPE: The RIPE Network Coordination Centre (RIPE NCC) is the RIR for Europe

RIR: Regional Internet Registers

RUP: Rational Unified Process

SPAN: The Tele2 order system

SSL: The secure sockets layer

SQL: Structured Query Language

UTF8: Unicode Transformation Format-8. A character set with the possibilty to represent most characters in the world.

VARCHAR: Character string with variable character string.

XHTML: eXtensible HyperText Markup Language

XML: eXtensible Markup Language

10 References

1. Prechelt, L. March 2000. *”An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl.”* IEEE Computer 2000
2. Apache Software Foundation. 2003. <http://www.apache.org>
3. Comer, D. E. 2000. ”Internetworking with TCP/IP Principles, protocols, and architectures.”. ISBN 0-13-018380. New Jersey. Prentice Hall, Inc
4. Deitel, H.M., Deitel P.J and Nieto T.R. 2002. *”Internet & World wide web. How to Program”.* ISBN 0-13-030897-8, New Jersey, Prentice-Hall, Inc.
5. Enterprise JavaBeans Technology. 2003. <http://java.sun.com/products/ejb/>
6. FreeIPDb. 2003. <http://www.freeipdb.org/>
7. Hunt, A and Thomas, D. 2000 “*The pragmatic programmer*”. ISBN 0-201-61622-X. United States. Addison Wesley.
8. IBM Rational Software. 2003. <http://www.rational.com>
9. Ince, D. 2002. ”Developing Distributed and E-commerce Applications”. ISBN 0-201-73046-4. Addison Wesley
10. Internet Assigned Numbers Authority (IANA). 2003. <http://www.iana.net>
11. JavaServer Pages Technology. 2003. <http://java.sun.com/products/jsp/>
12. Nominum, Inc. 2003. <http://www.nominum.com/product.php?id=FMC>
13. NorthStar IP Management System. 2003. <http://www.brownkid.net/NorthStar/>
14. PHP: Hypertext Preprocessor. 2003. <http://www.php.net/>
15. RIPE NCC Homepage. 2003. <http://www.ripe.net>
16. Tele2 AB. 2003. <http://www.tele2.com>
17. Voyager Enterprise Systems. 2003. <http://www.vger.com/v_ip>
18. XHTML 1.0: The extensible hypertext markup language (Second ed.). 2003. <http://www.w3.org/TR/xhtml1/>

Appendix 1. Use cases

USE CASE 1: New IP application.

A. CHARACTERISTIC INFORMATION
   - **Goal in context:** Customer sends an application for new IP-Addresses via email or fax, and expects to get IP-Addresses assigned within one week.
   - **Scope:** Company
   - **Level:** Summary
   - **Channel to primary actor:** Fax, email
   - **Secondary actors:** SPAN, Routers
   - **Preconditions:** We know the customer and the customer has a link installed. Info about the customer and the node are available in SPAN.
   - **Success end condition:** The customer has the right to get IP-addresses and we (Tele2), has IP-addresses available in the customers region.
   - **Failed end condition:** The customer has no right to get IP-addresses.

B. MAIN SUCCESS SCENARIO
   1. Customer sends an IP-application by fax.
   2. Tele2 checks up customer in SPAN. Verifies that customer has an order.
   3. Tele2 searches RIPE for IP-addresses already assigned to customer by Tele2, RIPE, or other LIR.
   4. Tele2 checks that customer motivates that it needs more IP addresses properly
   5. Tele2 finds a suitable IP-range to assign to customer.
   6. Tele2 modifies router to route the assigned IP-range to the customer.
   7. Tele2 makes a note in SPAN about the new IP-range for the customer.
   8. Tele2 updates RIPE database.
   9. Tele2 sends verification to customer.

C. EXTENSIONS
   1. (4) The customer needs to provide more information. Iterate back to 1), when more information is faxed.
   2. (4) The customer has an IP-range to assign himself that the customer wants to use.
   3. (4) The customer is already assigned the address range on another link. And it should be rerouted.
   4. (5) The customer get some addresses assigned and some addresses reserved up to one year.
   5. (5) The customer borrows addresses from Tele2.
   6. (5) The customer takes over addresses from another customer. Execute USE CASE 3, and then continue.
   7. (5) The customer is temporary assigned addresses. When complementing information is provided, iterate back to stage 5).
   8. (5) The customer wants to be reassigned the

D. VARIATIONS
   1. (1) The customer sends application by email.
   2. (1) The IP application comes from another Tele2 department and will be used a Tele2 product.
   3. (5) If the customer has an IP-Range from before, the IP's next to that range should be assigned, if available.
   4. (5) If the customer wants several addresses at different places, the addresses

can be moved from where they originally are allocated. User Case 6.

5. (9) DNS reverse is generated for Tele2 a service.

E. RELATED INFORMATION

USE CASE 2: Alarm when more addresses soon are needed.

A. CHARACTERISTIC INFORMATION
- **Goal in context:** To send an email warning when more addresses are needed in a certain node/country/service
- **Scope:** Company
- **Level:** Summary
- **Preconditions:** The areas to be searched and the amount of needed free addresses and sizes.
- **Success end condition:** There are not enough addresses available.
- **Failed end condition:** There are enough addresses available.

B. MAIN SUCCESS SCENARIO
1. The check is initiated by the fact that number of addresses in a certain allocation has been modified.
2. Tele2 counts number of addresses that are available and occupied.
3. The number of available addresses is found to be below the predefined level.
4. An email is sent to staff.

C. EXTENSIONS
1. (1) The usage of a service (for example top usage of ADSL France) is provided by external part and reported to the application.

D. VARIATIONS
1. (3) The number of available addresses in a specific range is found to be too low.

E. RELATED INFORMATION
- **Priority:** Top
- **Performance target:** Search should take less then five seconds.
- **Frequency:** Should be executed every time address space is modified.
- **Superordinate use case:** Use case 1,
- **Subordinate use case:** User case 4 – New IP allocation from RIPE,
- **Channel to primary actor:** email
- **Secondary actors:**

F. SCHEDULE
1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 3: Recycling of IP addresses

A. CHARACTERISTIC INFORMATION
   - **Goal in context:** To add and address range to the list of available addresses
   - **Scope:** Company
   - **Level:** Summary
   - **Preconditions:** A definition of quarantine time for IP addresses.
   - **Success end condition:** The address has been unused the whole quarantine time.
   - **Failed end condition:** The customer "changes" his mind, and wants to get the address back.

B. MAIN SUCCESS SCENARIO
   1. Produce a list of customers that no longer need their addresses (with the help of SPAN).
   2. Remove the address range's route from switch router, when notification arrives from operator.
   3. Traceroute is made to verify that the addresses are not used.
   4. Verify information in SPAN.
   5. Update the information at RIPE.
   6. Remove reverse DNS information.
   7. Change the status of the address range to "quarantine".
   8. Wait until the quarantine time has passed, and then change the status of the range to available.

C. EXTENSIONS
   1. (1) The no longer used address range is detected by the system by consistency checks against SPAN, RIPE and traceroutes.
   2. (4-8) Address range should not be put in quarantine, since it's supposed to be assigned directly to another customer.

D. VARIATIONS
   1. (1) The list comes from another source.
   2. (2) The customer changes his mind, before the given date, and wants to delay the disconnection.

E. RELATED INFORMATION
   - **Priority:** Top
   - **Performance target:** Response time within seconds.
   - **Frequency:** about 20 / month
   - **Superordinate use case:**
   - **Subordinate use case:**
   - **Channel to primary actor:** link to SPAN and router.
   - **Secondary actors:** RIPE

F. SCHEDULE
   1. **Due date:** Release 1.0

G. OPEN ISSUES

   - **Priority:** Top
   - **Performance target:** Some activities are allowed to take longer time, but the system must respond fast to the user.
   - **Frequency:** Up to 50/day

- **Superordinate use case:**
- **Subordinate use case:** 6. Move of address range

F. SCHEDULE

1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 4: New IP allocation

A. CHARACTERISTIC INFORMATION
   - **Goal in context:** To add number of available IP's in list.
   - **Scope:** Company
   - **Level:** Summary
   - **Preconditions:**
   - **Success end condition:** The application is approved by RIPE.
   - **Failed end condition:** The application is not approved by RIPE..

B. MAIN SUCCESS SCENARIO
   1. Tele2 sends an email to RIPE with information about how many IP addresses are needed.
   2. Ripe Answers with more questions
   3. Step 1 & 2 is repeated until RIPE approves of addresses.
   4. Ripe sends email with the allocation of IP addresses.
   5. The addresses are added into the pool of available addresses.

C. EXTENSIONS
   1.

D. VARIATIONS
   1. (1) The new addresses come from another source.
   2. (3) The addresses might be split up to different parts of the network.

E. RELATED INFORMATION
   - **Priority:** Top
   - **Performance target:**
   - **Frequency:** Less then one/week.
   - **Superordinate use case:**
   - **Subordinate use case:** User Case 6.
   - **Channel to primary actor:** Email
   - **Secondary actors:**

F. SCHEDULE
   1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 5: Deallocation of IP address range

A. CHARACTERISTIC INFORMATION
- **Goal in context:** To be able to remove an IP address range
- **Scope:** Company
- **Level:** Summary
- **Preconditions:** The IP address range must not be allocated (or reserved).
- **Success end condition:** The address range can safely be removed from the router.
- **Failed end condition:** Some addresses are routed to customers.

B. MAIN SUCCESS SCENARIO
1. The address range that should be removed is received from somewhere.
2. Tele2 checks that no addresses are assigned to customers.
3. Tele2 removes the address range from that specific router.

C. EXTENSIONS
1.

D. VARIATIONS
1.

E. RELATED INFORMATION
- **Priority:** Top
- **Performance target:** ?
- **Frequency: ?**
- **Superordinate use case:**
- **Subordinate use case:** Use case 6; move of address range.
- **Channel to primary actor:**
- **Secondary actors:**

F. SCHEDULE
1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 6: Move of address range

A. CHARACTERISTIC INFORMATION
  - **Goal in context:** To move the allocation of address range
  - **Scope:** Company
  - **Level:** Level
  - **Preconditions:** That the address range is not routed to customer.
  - **Success end condition:** Both User Case 4 and User Case 5 is successful.
  - **Failed end condition:** One of User Case 4 and User Case 5 fails.

B. MAIN SUCCESS SCENARIO
  1. A customer is approved (User Case 1) to move an address range.
  2. Execute User Case 4: delete range. (The old location)
  3. Execute User Case 5: add range. (The new location)
  4. Continue User Case 1.

C. EXTENSIONS
  1.

D. VARIATIONS
  1.

E. RELATED INFORMATION
  - **Priority:** Top
  - **Performance target:**
  - **Frequency:**
  - **Superordinate use case:** User case 1.
  - **Subordinate use case:**
  - **Channel to primary actor:**
  - **Secondary actors:**

F. SCHEDULE
  1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 7: Consistency check

A. CHARACTERISTIC INFORMATION
- **Goal in context:** To verify info in database against RIPE and routers
- **Scope:** Company
- **Level:** Summary
- **Preconditions:** The IP range to be checked must be known.
- **Success end condition:** The information is consistent in switch, ripe and database.
- **Failed end condition:** The information is not consistent.

B. MAIN SUCCESS SCENARIO
1. IP range to check is entered into the application.
2. Information about the IP ranged is gathered from database, RIPE and routers.
3. The information is compared and it is verified that the information is consistent.
4. The result is returned to the user.

C. EXTENSIONS
1.

D. VARIATIONS
1.

E. RELATED INFORMATION
- **Priority:** High
- **Performance target:** The performance limit should be from traceroute and RIPE, not from this program.
- **Frequency:**
- **Superordinate use case:**
- **Subordinate use case:**
- **Channel to primary actor:** Mail
- **Secondary actors:** traceroute

F. SCHEDULE
1. **Due date:** Release 1.0

G. OPEN ISSUES

USE CASE 8: History

A. CHARACTERISTIC INFORMATION
  - **Goal in context:** To provide historical statistics to the user
  - **Scope:** Company
  - **Level:** Summary
  - **Preconditions:** -
  - **Success end condition:** The correct result is returned to the user
  - **Failed end condition:** The result could not be returned to the user.

B. MAIN SUCCESS SCENARIO
  - The user enters criteria's for the search
  - The application computes the result
  - The result is returned to the user.

C. EXTENSIONS
  -

D. VARIATIONS
  -

E. RELATED INFORMATION
  - **Priority:** Medium
  - **Performance target:** Wait time over 10 seconds is not acceptable for a "normal" search.
  - **Frequency:** Used less then once a day.
  - **Superordinate use case:**
  - **Subordinate use case:**
  - **Channel to primary actor:** WEB.
  - **Secondary actors:**

F. SCHEDULE
  - **Due date:** Release 1.1

G. OPEN ISSUES
  - How advanced search's should be possible to search for.