Prepayment module for a Least Cost Routing VoIP telephony platform

Implementation and integration with Lancelot Telecom’s telephony platform

A thesis submitted to the

Rotterdam University of applied Sciences

for the degree of

Bachelor of ICT

Presented by

Matthias van der Vlies

born August 7, 1988
Vlaardingen, the Netherlands
Student number: 0785972

Exam committee

A. Chamani Foumenidana, primary lecturer
J. Grobben, secondary lecturer
J.J.M. Bruggemann, company mentor

version: final-2012-01-24-1-gf9f87ec

Bodegraven 2011-2012
Preface

Before you is the thesis for the graduation internship on the subject, “Prepayment module for a Least Cost Routing VoIP telephony platform” implemented at Lancelot Telecom B.V. by Matthias van der Vlies, student at Rotterdam University of applied Sciences.

The contents of this document will be used for assessing the graduation internship.

This document describes the technical and functional implementation of a pre-payment module into the existing VoIP telephony platform of Lancelot Telecom B.V. which is a small Telecoms operator based in the Netherlands. This document also describes scenarios that could cause the company to lose money and shows how to minimize impact of such scenarios.

The reader is expected to have knowledge on the following telephony subjects:

- VoIP
- Least Cost Routing
- Billing
- Wholesale termination

Please refer to the glossary for unclear terms and definitions.

Special thanks go out to:

- Jeroen Bruggemann and Willem Johan Bogert for providing me with the opportunity to work for an innovative company and giving me the chance to improve my skills in every possible way.
- Hossein Chamani Foumenidana for supporting me during my education and my graduation internship.
- John Grobben for supporting me during my graduation internship.
- My parents, Andre and Joke for teaching me how to survive in the big world and always believing in me.

Note: when mentioning the word ’current’, it refers to the old situation as the process to get to the actual current situation is being illustrated.

This document was written using LaTeX
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>I</td>
</tr>
<tr>
<td>Contents</td>
<td>II</td>
</tr>
<tr>
<td>Executive summary</td>
<td>V</td>
</tr>
<tr>
<td>Technical summary</td>
<td>VIII</td>
</tr>
<tr>
<td>Glossary</td>
<td>XII</td>
</tr>
<tr>
<td>List of Figures</td>
<td>XIV</td>
</tr>
<tr>
<td>Bibliography</td>
<td>XV</td>
</tr>
</tbody>
</table>

## 1 Background

1.1 Introduction                                 | 2
   1.1.1 Lancelot Telecom B.V.                      | 3
   1.1.2 Rotterdam University of Applied Sciences (Hogeschool Rotterdam) | 3
   1.1.3 Least Cost Routing                          | 3

1.2 Current situation                            | 4

1.3 Desired situation                            | 5

## 2 Assignment

2.1 Introduction                                 | 8

2.2 Research questions                           | 9

2.3 Process control                              | 10
   2.3.1 eXtreme Programming                      | 10
   2.3.2 Trac                                      | 10
   2.3.3 Quality assurance                         | 10

2.4 Deliverables                                 | 11

## 3 Process

3.1 Introduction                                 | 12

3.2 System analysis                              | 13
   3.2.1 Routing engine                            | 13
   3.2.2 Billing engine: call monitor              | 13
   3.2.3 Problem scenarios                         | 13
   3.2.4 Solutions and implementation              | 14
   3.2.5 Requirements                              | 15
   3.2.6 Remaining elements                        | 15
   3.2.7 Requirements                              | 16
   3.2.8 Implementation                            | 16
   3.2.9 New architecture                          | 16
   3.2.10 Requirements                             | 16
   3.2.11 Implementation                           | 17
Executive Summary

There is a complementary summary including extra technical information in the technical summary.

Note: when mentioning the word ‘current’, it refers to the old situation as the process to get to the actual current situation is being illustrated.

Background

Lancelot Telecom is using a so called LCR (Least Cost Routing) system to allow making routing decisions for wholesale telephony traffic termination based on the following parameters:

- Costs
- Revenue
- Quality
- Manual overrides

Currently, all customers are sending traffic on a prepayment basis, there is however no automatic way of monitoring account balances and stopping traffic when a balance drops to zero.

Lancelot Telecom would like to have a system that enables this functionality within their current system, via a prepaid module.

In the desired situation a new architecture will be implemented that is able to recover from hardware and software failures. To solve the issues regarding the risks taken when terminating traffic the following has to be implemented:

- Stopping traffic when an account balance hits zero
- Warning customers and personnel about the account balance status
- Real-time view of the customer’s traffic and balance
- Allow customers to make a deposit using Lancelot Telecom’s own payment system (PayPal) via a customer portal

Research questions

To be able to design and implement the desired situation, the following questions arise:

- What are possible scenarios that may cause money to be lost and what are possible solutions for these scenarios?
- Which warning levels should be defined for sending out notifications concerning account balances?
- Is it desirable to terminate calls currently in progress when the account balance hits zero?

The following non-technical requirements haven been set out:

- A prepayment module matching the criteria given in the desired situation.
- Answers to the research questions and implementation of the found solutions.
System analysis

Due to the complexity of the overall system it is wise to do a full analysis before starting work on the actual project. The reason for this is, that although there have been requirements set, new requirements are likely to come up. The full analysis is described in the 'System analysis' section of chapter two.

The following extra requirements came up:

Customer portal requirements

- Usable by non-prepaid customers as well.
- Authorization via a user log-in.
- Account status page showing account related details.
- Traffic page showing near-real-time traffic insights to the customer.
- Payments page allowing prepaid customers to see their payments, account balance and ability to 'buy' new balance.
- Settings page for setting warning thresholds and notification options.
- Administration accounts to manage users.

Results

The prepaid module has been successfully implemented and is in use by Lancelot Telecom for real traffic.

Conclusion

All requirements have been met, allowing Lancelot Telecom to roll out new services to their customers. See below for answers to research questions.

Which warning levels should be defined for sending out notifications concerning account balances?

We specify the following levels:

- warning
- critical
- zero

The thresholds are configurable by the customer. By company experience the following default thresholds are defined for 'regular' customers:

<table>
<thead>
<tr>
<th>Level</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning</td>
<td>500</td>
</tr>
<tr>
<td>critical</td>
<td>100</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
</tbody>
</table>

Is it desirable to terminate calls currently in progress when the account balance hits zero?

No customer should be able to get a negative balance or at least a minimal amount. This means that the policy should be that calls are terminated by the call monitor when the calls are still in progress.
Recommendations

By meeting all set requirements and intensive testing it is recommended to immediately deploy the prepaid module on the production systems. This way, Lancelot Telecom is able to utilize the module to its full potential and have the ability to extend their services based on this module.

Lancelot Telecom has existing customers which are already making manual prepayments. Lancelot Telecom should inform these customers of the availability of the customer portal so can give their customers a higher service level.
Technical Summary

This is the complementary summary to the executive summary including extra technical information. Please refer to the executive summary, since it’s required to fully understand this technical summary.

Technical research questions

To be able to implement the desired situation, the following questions arise:

- What are possible scenarios that may cause money to be lost and what are possible solutions for these scenarios?
- How can the billing engine be modified to do real-time prepaid billing?
- What should the hardware/software architecture look like to almost match the zero-fault-tolerance criteria?

The following technical requirements have been set out:

- Writing JUnit test cases to prove the found solutions are valid.
- Javadoc documentation for each newly implemented code.
- Using Trac to manage the project.
- Hosting code in an VCS system.
- Migrating the existing source code from SVN to GIT
- A customer portal implemented using the Play! framework

System analysis

Due to the complexity of the overall system it is wise to do a full analysis before starting work on the actual project. The reason for this is, that although there have been requirements set, new requirements are likely to come up. The full analysis is described in the 'System analysis' section of chapter two.

The following extra requirements came up:

Technical customer portal requirements

- Communication with SPoC APIs to be able to show and manipulate customer data.

Hardware requirements

- Hardware fail-over capabilities.
- Load sharing.
- Increased hardware capacity.

Technical Conclusion

All requirements have been met, allowing Lancelot Telecom to roll out new services to their customers. See below for answers to research questions.
What are possible scenarios that may cause money to be lost and what are possible solutions for these scenarios?

- System failures; impact is minimized by creating fail-over and load-sharing mechanisms.
- Data corruption; impact is minimized by using reliable protocols that use checksumming to verify data.
- Concurrency issues; By using a SPoC layer to ensure atomic operations.
- Security issues; By creating an additional layer between the SPoC layer and external services and provide security mechanisms for them.

How can the billing engine be modified to do real-time prepaid billing?

- Store call information in a database.
- Move to an asynchronous (multi-threaded) architecture to allow high performance call monitoring.
- Use the SPoC layer (Prepaid Billing Server) to debit credits.

What should the hardware/software architecture look like to almost match the zero-fault-tolerance criteria?

<table>
<thead>
<tr>
<th>Server</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>apollo.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>athena.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>ares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine and Prepaid Billing Server</td>
</tr>
<tr>
<td>antares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine, and fail-over Prepaid Billing Server</td>
</tr>
<tr>
<td>artemis.bb.lancelot-telecom.nl</td>
<td>Fail-over for atlas</td>
</tr>
<tr>
<td>atlas.bb.lancelot-telecom.nl</td>
<td>LCR web interface</td>
</tr>
<tr>
<td>astro.bb.lancelot-telecom.nl</td>
<td>Unused for LCR platform</td>
</tr>
<tr>
<td>ajax.bb.lancelot-telecom.nl</td>
<td>System monitoring</td>
</tr>
</tbody>
</table>

Recommendations

Implementing more test cases will prevent serious problems to arise in day-to-day business as well increase the code’s quality. Using cobertura we were able to retrieve the following statistics:

<table>
<thead>
<tr>
<th>Result type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>340</td>
</tr>
<tr>
<td>Line coverage</td>
<td>6% (702/11439)</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>1% (72/3864)</td>
</tr>
<tr>
<td>Complexity</td>
<td>2.107</td>
</tr>
</tbody>
</table>

Although the average McCabe cyclomatic code complexity \(^{[\text{Wiki11a}]}\) is low, there are some packages that really need refactoring to decrease code complexity, mainly:

- com.lancelot.asterisk.agi.scripts
- com.lancelot.lcr.processors
- com.lancelot.callgen

My advise would be to research what coverage is acceptable for the LCR system matched against to be defined company standards. My personal opinion is that important packages with a complexity above 2.0 should have 80% line coverage and about 100% branch coverage.
Most important parts of the code is documented, but lots of code is still missing Javadoc documentation. Writing more code documentation makes it easier for new developers to find their way in the code.

Using the stylecheck tool the following data was retrieved:

<table>
<thead>
<tr>
<th>Result type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of methods</td>
<td>1967</td>
</tr>
<tr>
<td>Number of attributes</td>
<td>872</td>
</tr>
<tr>
<td>Total methods/attributes</td>
<td>2839</td>
</tr>
<tr>
<td>Undocumented methods/attributes</td>
<td>907</td>
</tr>
<tr>
<td>Undocumented percentage</td>
<td>31.95%</td>
</tr>
<tr>
<td>Total lines of code</td>
<td>24398</td>
</tr>
</tbody>
</table>

This leaves room for improvement.
Figure 1: Overview of the new architecture of Lancelot Telecom’s telephony platform
AGI The Asterisk Gateway Interface (AGI) is an interface for adding functionality to Asterisk with many different programming languages. [inf11b]. 4

AMI The Asterisk Manager Interface (AMI) allows a client program to connect to an Asterisk instance and issue commands or read events over a TCP/IP stream. [inf11c]. 4

Asterisk Asterisk is a complete PBX. [inf11a]. 4, III

CDR A Call Detail Record (CDR) contains valuable information regarding telephone calls. 5, 11

dialcode Part of a telephone number, mostly assigned to a specific destination/carrier. III

GIT A distributed VCS created by Linus Torvalds. 8

HTTP Hyper Text Transfer Protocol for transferring data. 11

Javadoc Documentation standard for the Java programming language. 8

JUnit A Java library for writing and executing test cases. 7, 8, 27

LaTeX LaTeX is a macro package based on TeX created by Leslie Lamport. Its purpose is to simplify TeX typesetting, especially for documents containing mathematical formulae. [aut11]. 1

LCR A system that allows making routing decisions for wholesale telephony traffic termination. 3, 6, 13, 14, 17, 24, 27

Master Dialcode List A list matching dialcodes to a specific destination. 5

PBX A Private Branch eXchange (PBX) is a phone switch located at the customer’s premises. [inf11d]. III

Play! framework A Java framework used for building web applications. 8, 13, 19

POP A point-of-presence (POP) is an artificial demarcation point or interface point between communications entities. [Wik11c]. 3, 17

REST Representational State Transfer (REST) is a way of representing data in a software architecture using HTTP. 11

SoC Single point of Control. 10–13, 24

SVN A VCS. 8

TCP/IP An Internet protocol for reliable communications between two hosts. 11
**TeX** TeX is a low-level markup and programming language created by Donald Knuth to typeset documents attractively and consistently. [aut11]. III

**Trac** An open source tool for managing projects. 7, 8

**VCS** A Version Control System (VCS) allows version management for files, for example: source code files. 8, III

**VoIP** Voice over Internet Protocol (Voice over IP, VoIP) is one of a family of internet technologies, communication protocols, and transmission technologies for delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet. [Wik11d]. 4, 6, 10, 13, 14, 24

**XML** Easy to read file format used in communications. 11
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>CRM module</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Pricing module</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Current architecture</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Trac tickets</td>
<td>10</td>
</tr>
<tr>
<td>4.1</td>
<td>Prepaid options in LCR system</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Hosts in Nagios</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>GIT repositories</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Customer Portal - login</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Customer Portal - Account status</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Customer Portal - Traffic overview</td>
<td>25</td>
</tr>
<tr>
<td>4.7</td>
<td>Customer Portal - Traffic details</td>
<td>25</td>
</tr>
<tr>
<td>4.8</td>
<td>Customer Portal - Payments</td>
<td>25</td>
</tr>
<tr>
<td>4.9</td>
<td>Customer Portal - Settings</td>
<td>26</td>
</tr>
<tr>
<td>4.10</td>
<td>Customer Portal - Users management</td>
<td>26</td>
</tr>
<tr>
<td>4.11</td>
<td>Customer Portal - User management</td>
<td>26</td>
</tr>
<tr>
<td>4.12</td>
<td>Customer Portal - Dutch version</td>
<td>27</td>
</tr>
<tr>
<td>4.13</td>
<td>Customer Portal - Branding</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>New architecture</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Cobertura overview</td>
<td>34</td>
</tr>
<tr>
<td>6.2</td>
<td>Cobertura class overview</td>
<td>35</td>
</tr>
</tbody>
</table>
Bibliography

Chapter 1

Background
1.1 Introduction

This chapter illustrates the goals of the graduation project. The following details will be discussed:

- Involved parties and their backgrounds.
- Introduction to the current architecture.
- Description of the issues found on the current architecture.
- Solutions and boundaries (scope) for fixing these issues.

1.1.1 Lancelot Telecom B.V.

Lancelot Telecom B.V. is a small, young and innovative telecoms operator in the Netherlands focusing on hosting geographic and premium telephone numbers in the Netherlands. Their main business also consists of providing international telephony traffic termination to the market for use by business to business customers.

Lancelot Telecom’s network has several POPs in the following cities:

- Amsterdam
- London
- Frankfurt
- Zurich

All services and solutions are designed and by implemented by the company itself, leveraging great capabilities for the company to explore new and niche markets throughout the world without depending on external organizations.

1.1.2 Rotterdam University of Applied Sciences (Hogeschool Rotterdam)

Rotterdam University is one of the major Universities of Applied Sciences in the Netherlands. Currently almost 30,000 students are working on their professional future at our university.

The university is divided into eleven schools, offering more than 80 graduate and undergraduate programmes in seven fields: art, technology, media and information technology, health, behaviour and society, engineering, education, and of course, business. [Rot11]

1.1.3 Least Cost Routing

Lancelot Telecom is using a so called LCR (Least Cost Routing) system to allow making routing decisions for wholesale telephony traffic termination based on the following parameters:

- Costs
- Revenue
- Quality
- Manual overrides

This allows Lancelot Telecom to offer traffic termination based on the best price/quality to their customers. The LCR system was built internally by Lancelot Telecom. This leverages great possibilities to customize the system to the company’s needs.

The LCR system consists out of the following 'modules':

- Customer management
- Routing engine
- Pricing engine
- Billing engine
• Services engine

All of these modules are accessible via a web interface. The LCR application was built using the following application stack:

• GNU/Linux Debian
• Asterisk
• JBoss Application Server
• PostgreSQL
• Memcached
• Java
• ztemplates
• Hibernate

Customer management

The customer management module makes basic customer administration possible via a web interface. This includes configuration for VoIP connectivity and setting billing parameters for each customer, which are used by the routing engine for taking decisions by using the configured parameters, such as:

• IP addresses of VoIP gateways and other settings
• Allowed traffic directions (inbound v.s. outbound or both)
• Per second billing or per minute billing
• Static (forced) routing

Figure 1.1: Example view of managing a customer
Routing engine

The routing engine module provides high-level access to the telephony VoIP subsystems using the AGI and AMI protocols. The VoIP subsystems are controlled by the routing engine, which is also responsible for making routing decisions based on the parameters configured. By allowing these decisions to be made and controlled centrally, multiple subsystems are able to use the same routing information.

Pricing engine

To be able to keep all pricing information complete the pricing engine allows the process of importing rate sheets provided by suppliers, which include pricing broken down into destinations. It is also essential that customer pricing can be managed. This module provides an interface to do so. Further tasks are:

- Processing price changes
- Show pricing information for each destination.
- Managing the Master Dialcode List

Dialcodes

Filter by zone name:  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z all
Filter by dial code:  1 2 3 4 5 6 7 8 9

<table>
<thead>
<tr>
<th>Master dial code list:</th>
<th>Dialcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Zone (148)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Dialcodes</strong></td>
<td></td>
</tr>
<tr>
<td>AFGHANISTAN</td>
<td>93</td>
</tr>
<tr>
<td>AFGHANISTAN HERAT</td>
<td>9340</td>
</tr>
<tr>
<td>AFGHANISTAN KABUL</td>
<td>9302</td>
</tr>
<tr>
<td>AFGHANISTAN KANDAHAN</td>
<td>9330</td>
</tr>
<tr>
<td>AFGHANISTAN NAZAR E SHIRAF</td>
<td>9350</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE</td>
<td>9378, 9372, 9371</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE AWCC</td>
<td>9370</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE COHA</td>
<td>9375</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE ITHN</td>
<td>9377</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE TCHIA</td>
<td>9379</td>
</tr>
<tr>
<td>AFGHANISTAN MOBILE NASSEL</td>
<td>937502, 937501, 937503, 937500, 937504</td>
</tr>
<tr>
<td>AFGHANISTAN NANGARHAN</td>
<td>9360</td>
</tr>
<tr>
<td>AFGHANISTAN SPECIAL NAMSHER</td>
<td>93135603500</td>
</tr>
</tbody>
</table>

Figure 1.2: Example view of managing the Master Dialcode List

Billing engine

Billing customers is an essential part of a company’s back-office process. When a call is terminated by the system a so called CDR (Call Detail Record) is generated which contains valuable data regarding the call:

- duration
- calling party
- called party
- customer
- supplier

With this information the billing engine is able to make a billing record for the CDR, which can later be used to generate invoices for customers.

In the telephony business disputes regarding invoices occur regularly so the billing engine is also able to export the CDRs with pricing information found in the billing records so the disputing parties are able to use the CDRs for CDR mediation.
Services engine

This module offers various services that are used by retail customers. The goals of this module are not within the scope of this document and not a necessary part for understanding the thesis’ subject. There is however an indirect relation, which will be explained later on in this document.

1.2 Current situation

Currently, all customers are sending traffic on a prepayment basis, there is however no automatic way of monitoring account balances and stopping traffic when a balance drops to zero. This means Lancelot Telecom’s employees need to keep track of account balances manually, which can be an issue since this process can never be real-time due to a limited amount of resources. In theory this also means that the company is taking huge risks when terminating traffic as they can not be a 100% sure that they will get paid for traffic above the set limit.

Lancelot Telecom also has the wish to create some new services for their retail customers which they want to make prepayments as well. This is currently not possible, thus the requirement for implementing a prepayment module for their LCR system.

Implementing such a prepayment module requires some more hardware resources to run these modules on. Currently Lancelot has three servers in use for their LCR system.

<table>
<thead>
<tr>
<th>Server</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>apollo.bb.lancelot-telecom.nl</td>
<td>Complete LCR system and VoIP subsystem</td>
</tr>
<tr>
<td>astro.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine</td>
</tr>
<tr>
<td>ajax.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem and system monitoring</td>
</tr>
</tbody>
</table>

If one of the above servers fails it affects the entire VoIP platform and traffic will not be possible anymore.

Putting all above together gives us the following diagram:

![Diagram of the current architecture of Lancelot Telecom's telephony platform](image)

Figure 1.3: Overview of the current architecture of Lancelot Telecom’s telephony platform
1.3 Desired situation

In the desired situation a new architecture will be implemented that is able to recover from hardware and software failures. To solve the issues regarding the risks taken when terminating traffic the following has to be implemented:

- Stopping traffic when an account balance hits zero
- Warning customers and personnel about the account balance status
- Real-time view of the customer’s traffic and balance
- Allow customers to make a deposit using Lancelot Telecom’s own payment system (PayPal) via a customer portal
Chapter 2

Assignment
2.1 Introduction

This chapter describes the complete assignment that has been given to the student by Lancelot Telecom.

- Involved parties and their backgrounds.
- Introduction to the current architecture.
- Description of the issues found on the current architecture.
- Solutions and boundaries (scope) for fixing these issues.

2.2 Research questions

To be able to implement the desired situation, the following questions arise:

- How can the billing engine be modified to do real-time prepaid billing?
- What are possible scenarios that may cause money to be lost and what are possible solutions for these scenarios?
- Which warning levels should be defined for sending out notifications concerning account balances?
- Is it desirable to terminate calls currently in progress when the account balance hits zero?
- What should the hardware/software architecture look like to almost match the zero-fault-tolerance criteria?

2.3 Process control

Lancelot Telecom has some internal control mechanisms for projects which is mainly based on an agile development process.

2.3.1 eXtreme Programming

Lancelot Telecom is using the Agile development method, eXtreme programming.

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles (timeboxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting changes in the customer’s requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers. [Wik11b]

As with agility, there comes making choices, for this subject, no pair-programming is required.

Alternatives

An alternative could be to use Scrum, however this requires various roles. Since I’m the only developer working on this project and the company is the Scrum owner it would cause too much overhead on project management which we don’t need. We want to be able to make fast decisions and don’t have any roles interfering.

Lancelot telecom’s ethos doesn’t match to this methodology as well. Lancelot wants to move fast and not been held back by overhead caused by project management. XP simply offers us the flexibility we need.
Personal view

I fully agree using XP as the methodology to use for this project. I have a 3 year history of using this methodology and feel very comfortable in using its elements. As from a project perspective, my experience with Lancelot Telecom is that requirements change very often and priorities are often shifted. This means that the development methodology needs the same kind of flexibility. By splitting the project into various smaller chunks it’s easy to prioritize certain functionality and emphasize on them very quickly. These smaller chunks also make it much more easy to delegate chunks to other developers (if required.)

This methodology matches the company's ethos and flexibility very well.

2.3.2 Trac

Trac is an open source software product that allows:

- Issue tracking
- Road-maps for collecting issues in milestones
- Browsing source code
- Writing wiki-style documentation

This tool will be used to keep track of all the features of the prepayment system so that other employees have an overview on the status of each project.

<table>
<thead>
<tr>
<th>Status: closed</th>
<th>Ticket</th>
<th>Summary</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#294</td>
<td>Auto re-rating on immediate price changes</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#296</td>
<td>Store current calls in database for automated rating when updating application</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#493</td>
<td>Prepaid API</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#453</td>
<td>Create balance mechanism</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#454</td>
<td>Deduct credits</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#456</td>
<td>Stop traffic when balance is 0</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#457</td>
<td>Notify when balance is going to 0</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#459</td>
<td>Prepaid Billing Server</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#461</td>
<td>Prepaid Billing Client</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#462</td>
<td>Add SHA-1 checksum to server requests and check them</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#464</td>
<td>Implement TOPUP command</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#464</td>
<td>[Prepaid API] implement resource for requesting balance</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#465</td>
<td>[Prepaid API] implement resource for credit debit</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
<tr>
<td></td>
<td>#466</td>
<td>Prepaid UI in CRM</td>
<td><a href="mailto:matthias@lancelot-telecom.nl">matthias@lancelot-telecom.nl</a></td>
</tr>
</tbody>
</table>

Figure 2.1: Example tickets for a single milestone

2.3.3 Quality assurance

In telecoms it is absolutely vital to make no mistakes when billing customers and making cost calculations due to the large volumes being handled. Therefore all code must have automated tests that should test each of the requirements previously set-up.

Lancelot Telecom currently uses a combination of JUnit and Hudson Continuous Integration server to automatically run test cases when a developer modified or has written new code.

2.4 Deliverables

- A prepayment module matching the criteria given in the desired situation.
- Answers to the research questions and implementation of the found solutions.
- Writing JUnit test cases to prove the found solutions are valid.
- Javadoc documentation for each newly implemented code.
• Using Trac to manage the project.
• Hosting code in an VCS system.
• Migrating the existing source code from SVN to GIT
• A customer portal implemented using the Play! framework
Chapter 3

Process
3.1 Introduction

This chapter illustrates the process taken to get to the final results. The following details will be discussed:

- An analysis of the complete system to get the full grasp and understanding of it. Also solutions are given to solve issues that stand in the way for meeting the requirements.
- Approaches for tackling issues described in the analysis

Before we begin on this project, we must make a full analysis of the current situation to discover possible culprits and faults in the existing architecture and provide ways to improve these. Due to the complexity of the overall system it is wise to do such an analysis before starting work on the actual project. The reason for this is, that although there have been requirements set, new requirements are likely to come up. The analysis is described in the 'System analysis' section of this chapter. The reason for not putting this analysis in an appendix is to ease reading of this document as the content of the analysis is used in the next chapters.

3.2 System analysis

Integrating the prepayment module has the following dependencies with other modules.

3.2.1 Routing engine

The routing engine needs to know how a calling customer will be billed. This means the following information should be available:

- Is the customer using the prepayment mechanism?
- What is the customer's balance?
- How long will the customer be able to call with this balance?

3.2.2 Billing engine: call monitor

The call monitor is part of the billing engine. It has the following tasks:

- Monitoring active calls
- Generating billing records when a call completes

To allow prepaid billing it should have the following extra capabilities:

- Near-real-time billing
- Deducting credits from the account balance during a call

3.2.3 Problem scenarios

There are several scenarios that should be taken into account before implementing the module.

Concurrency and data integrity

In the current situation there are multiple call monitors, this means that in theory concurrency issues may arise when two call monitor instances make changes to the call balance. Normally this should be handled by the back-end database systems, however in a telecoms system there is zero-fault-tolerance policy, which for this system implicates that there should be a SPoC that handles all account balance related operations. There are also some more advantages to this approach:

- More control over data.
• Standardized manner of handling data.
• Caching possibilities.
• All operations will be ensured to be atomic.

Concurrency and performance
Currently the call monitor polls the VoIP subsystems every second to check the status of monitored calls. The monitoring process is a synchronous process, so with many calls this could take more than a second to complete. This means that with an increasing call load, it may be possible that account balance is not debited on time and a financial risk is exposed.

Data integrity
Due to the nature of the data it is very important that all data is transferred with high integrity. If data is corrupted, this means a risk that may cause financial resources to be lost or even worse customers being billed incorrectly.

Data exposure and security
The SPoC layer should not be directly exposed to external services like the customer portal. There should be a layer in between that does additional authentication, checks and bypasses some of the functionality that the SPoC layer provides to increase performance. Extra features that don’t require atomic operations can also be implemented in this layer.

Fault recovery
Currently, when a call monitor is unexpectedly closed down or experiences other problems, billing records for CDRs may not be created. This is fine for non-prepayment calls as there is a manual mechanism existing to resolve these issues, and thus rating the CDRs manually. When doing near-real-time billing this can not be done anymore, since credit may already have been deducted and the customer can not be billed for the entire call anymore.

3.2.4 Solutions and implementation
The following components need to be implemented or improved

Improved call monitor
To allow the call monitor to handle more calls simultaneously an asynchronous process should be used. This means multiple processes have to be running at the same time. By using a multi-threaded programming model we are able to ensure calls are being handled asynchronously.

Call data persistence
As said before we have incomplete information about calls when a call monitor crashes. This means we will need to store call information into a database, so when restarting a monitor it is able to recover the call and continue the billing process.
Prepaid Billing Server

The Prepaid Billing Server should:

- Act as a SPoC to ensure data integrity when using concurrency.
- Implement a network protocol using TCP/IP to ensure data integrity.
- Have a checksumming mechanism in the protocol to further ensure data integrity.

The Prepaid Billing Server’s protocol should have the following capabilities:

- Request current balance
- Top-up of balance
- Debits on balance

See appendix B for the protocol design.

Prepaid Billing API

The Prepaid Billing API should:

- Act as a layer between external services and the SPoC layer using HTTP, REST and XML
- Support authentication
- Provide more information concerning accounts and payments

See appendix A for the API design.

3.2.5 Requirements

Summing up all above leads to the following requirements:

- Modify routing engine to check call balance and duration when routing calls.
- Modify call monitor to do near-real-time billing.
- Improve call monitor performance by using a multi-threaded model.
- Implement Prepaid Billing Server to act as a SPoC for balance related actions.
- Implement Prepaid Billing API to limit data exposure and provide a security layer.
- Store current calls in database for failure recovery.

3.2.6 Remaining elements

There are still some open issues to be discussed.

Warning levels

When an account balance is hitting zero, the customer is not able to make any more calls. If this is realized when the balance has already been zero for a while, Lancelot Telecom will lose business and thus financial resources due to the customer not being able to generate traffic.

To minimize the impact of this issue, notifications have to be sent out via email. However there should be certain levels, as customers would like to know when their balance is getting low or has reached zero. We specify the following levels:

- warning
- critical
- zero
These levels should have certain thresholds. They could be fixed, however no customer is alike. The user should be able to decide what threshold should be used for the warning and critical level. The zero level is always '0'.

By company experience the following default thresholds are defined for 'regular' customers:

<table>
<thead>
<tr>
<th>Level</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning</td>
<td>500</td>
</tr>
<tr>
<td>critical</td>
<td>100</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
</tbody>
</table>

**Terminate calls**

As discussed internally at Lancelot Telecom no customer should be able to get a negative balance or at least a minimal amount. This means that the policy should be that calls are terminated by the call monitor when the calls are still in progress. Due to the warning notifications customers should realize in time that they should deposit more credit so in progress calls are not terminated. This minimizes financial risks for Lancelot Telecom.

**Customer portal**

The customer portal is a vital part of interacting with the customers. This allows Lancelot Telecom to build a trustworthy image and show their capabilities to the customer. As described earlier the warning levels for balance notification should be able to be managed in the customer portal. Lancelot Telecom also has non-prepaid customers which need to be able to login as well.

### 3.2.7 Requirements

- Usable by non-prepaid customers as well.
- Authorization via a user log-in.
- Account status page showing account related details.
- Traffic page showing near-real-time traffic insights to the customer.
- Payments page allowing prepaid customers to see their payments, account balance and ability to 'buy' new balance.
- Settings page for setting warning thresholds and notification options.
- Administration accounts to manage users.
- Communication with SPoC APIs to be able to show and manipulate customer data.

### 3.2.8 Implementation

Lancelot Telecom prefers using Play! framework for implementing new applications. This framework provides all the necessary features to implement above requirements.

### 3.2.9 New architecture

All above software changes also rely on hardware, of which the architecture needs to be redesigned.

### 3.2.10 Requirements

- Hardware fail-over capabilities.
- Load sharing.
- Increased hardware capacity.
3.2.11 Implementation

Lancelot Telecom will be buying five new servers to increase the hardware capacity:

- Two servers for running the LCR Routing engine in load sharing configuration and the Prepaid Billing Server in fail-over configuration.
- One server to replace ajax.bb.lancelot-telecom.nl as a VoIP subsystem
- Two servers for running the LCR web interface in fail-over configuration
- All new servers should be installed with GNU/Linux Debian

HeartBeat

To allow fail-over configurations to work there is a requirement to have a piece of software that manages shared resources between two hosts. If one hosts fails the other hosts knows about the resources running on the failing hosts and is able to recover them. HeartBeat is a software daemon that implements such functionality.

Routing engine (call monitor) and Prepaid Billing Server

These components are the most crucial components in the LCR system. This means that these services should be configured redundantly. The routing engine can be used in a load sharing configuration since there is no SPoC requirement. For the Prepaid Billing server there should be a SPoC this means that both hosts share a virtual IP address and in case a host fails the other will take over the service so the SPoC remains.

LCR web interface

The LCR system also provides a web interface which also runs some various background processes. This means only one such service can run at a time. This implies using a fail-over configuration.

New equipment and tasks

This translates into the following equipment and tasks:

<table>
<thead>
<tr>
<th>Server</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>apollo.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>athena.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>ares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine and Prepaid Billing Server</td>
</tr>
<tr>
<td>antares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine, and fail-over Prepaid Billing Server</td>
</tr>
<tr>
<td>artemis.bb.lancelot-telecom.nl</td>
<td>Fail-over for atlas</td>
</tr>
<tr>
<td>atlas.bb.lancelot-telecom.nl</td>
<td>LCR web interface</td>
</tr>
<tr>
<td>astro.bb.lancelot-telecom.nl</td>
<td>Unused for LCR platform</td>
</tr>
<tr>
<td>ajax.bb.lancelot-telecom.nl</td>
<td>System monitoring</td>
</tr>
</tbody>
</table>

3.2.12 Migration from SVN to GIT

Lancelot Telecom is currently using a single SVN repository for version controlling their source code. This needs to be split up into separate GIT repositories. There is no requirement for Lancelot Telecom to have this documented as this is a one-time operation.

3.3 Issue tackling

Now that we have a full analysis we are able to provide more details to the actual development process.
3.3.1 Setting up the development environment

To be able to work using short iteration, the development platform must be well set-up. In the current situation Lancelot Telecom has an eclipse project, called lancelot-lcr. Using ant build files modules are built:

- Manager module
- Routing module

We can integrate most of the functionality into the existing modules, however for the SPoC we need an additional module by creating a new ant build file. This is based on the existing build files. This script will build an ear file which we are able to deploy to a Jboss Application Server. This is very much the same as for the Routing and Manager module except its configuration:

```
src-static/lancelot-prepaid-server.ear/META-INF/application.xml:

<?xml version="1.0" encoding="UTF-8"?>
<application
  xmlns:javae="http://java.sun.com/xml/ns/javae"
</application>
```

```
src-static/lancelot-prepaid-server.ear/META-INF/jboss-app.xml:

<!DOCTYPE jboss-app PUBLIC "-//JBoss//DTD J2EE Application 4.2//EN" "http://www.jboss.org/j2ee/dtd/jboss-app_4_2.dtd">
<jboss-app>
  <loader-repository>
    Lancelot:loader=prepaid
  </loader-repository>
</jboss-app>
```

Furthermore we must create a so called MBean that starts the prepaid billing server:

```
@Management(PrepaidManagement.class)
@Service(objectName = "Lancelot:service=PrepaidMBean")
public class PrepaidMBean implements PrepaidManagement {
  private static final Logger log = Logger.getLogger(PrepaidMBean.class);
  private PrepaidServerThread thread = null;
  class PrepaidServerThread extends Thread implements Runnable {
    private PrepaidBillingServer server;
    public PrepaidServerThread(PrepaidBillingServer server) {
      this.server = server;
    }
    public void run() {
      log.info("Configuring Prepaid Billing server");
      if (server != null) {
        // nothing we can do about that and exceptions have already been
        // logged by start().
        log.error("StartUp failed in MBean PrepaidMBean -- attempting to continue", ex);
      }
    }
    public PrepaidBillingServer getServer() { return server; }
    public void start() throws Exception {
      log.info("StartUp failed in MBean PrepaidMBean -- attempting to continue", ex);
    }
  }
}
```

Note that on JBoss 4.0.5, this method is accidentally called twice by JBoss. It is a known bug: reports can be found at:

- http://jira.jboss.org/jira/browse/EJBTHREE-711
- http://jira.jboss.org/jira/browse/EJBTHREE-786
- http://jira.jboss.org/jira/browse/EJBTHREE-789

```
public void stop() throws Exception {
  if (thread != null & thread.getServer() != null) {
    thread.getServer().shutdown();
  }
}
```
The JBoss Application Server will auto-detect this MBean and start it when deploying the module. This allows us to implement the prepaid billing server according to the defined Prepaid API. By simply redeploying the jar file we can run the implemented software.

The actual implementation was built with Java NIO, which allowed us to build a high performance server and SPoC

The Prepaid Billing API must be implemented in the existing manager module. This module provides web services to other modules. This module makes use of ZTemplates and we need a controller for each API request. An example can be found in appendix A. Since we already have the server implemented we are now able to proxy the incoming requests to the SPoC.

3.3.2 Preparing for testing

For testing test cases must be defined, for each API method. Since the existing project already supports JUnit we can easily add new test cases. For an example test case see Appendix D. Since the API has been well documented on the internal wiki, it was easy to extract valid test cases.

3.3.3 Implementing and configuring the new equipment

We have a list of things to do when implementing the new situation:

- Specify hardware specifications (these will be not disclosed in this document, due to company policy)
- Compare pricing from various suppliers
- Order hardware
- Installation of the operating system (this has been delegated to another employee)
- Configuration
- Installing of the LCR system.

The biggest issue was ordering the hardware that took 3 weeks. Also the placement of the hardware on-site had to be planned in front. This was done together with another employee. Together we first made a plan, this included:

- Checking whether the current co-location site offered enough power
- Designing rack space
- Dividing tasks
  - Rack-mounting
  - Cabling
  - Labeling
  - Configuration testing

3.3.4 GIT migration

As I had only basic personal experiences with GIT. I’ve decided to buy a book on GIT, called Pro Git, by Scott Schacon. This book describes various scenarios very well in the migration process as it provided with clear examples. It also described how to leverage the maximum of using GIT versus Subversion, which also meant a lot of improvements to the current source tree could be made.
Chapter 4

Results
4.1 Introduction

This chapter illustrates the results of the service/module development that has taken place based on the set requirements.

4.2 Prepaid module

The prepaid module has the following components that were implemented:

- Prepaid Billing Server
- Prepaid Billing API
- LCR UI (User Interface)

The results for these components will be discussed per component in the next sections.

4.2.1 Prepaid Billing Server

By using a Jboss MBean we were able to start the prepaid billing server on deployment, and stop it on removal. The server has been installed on {ares, antares} by deploying them on the installed JBoss instances. Using HeartBeat configured for sharing an IP address, only one prepaid server is available (SPoC) on the shared IP address. If one host fails the other is able to take over the responsibility, without losing valuable data.

On the client side a retry mechanism was implemented to make sure hosts are able to switch the shared IP address resource to another resource, without causing issues in the upper layer call management.

Through logging the actions we are able to debug issues very easily.

Example logging of the Prepaid server:

```
[PrepaidBillingServer] - Client: /10.207.0.21:40414 connected
[CallMonitor] - [SIP 217.xx.xx.xx-apollo - 1325347780.801857] Call completed, customer: XXX supplier: XXXX from: 91XXXX to: 4076XXXX duration: 379 (0) costs = 0.0606
```

Explanation of what is happening:

- A call comes in from customer XXX
- It is determined by the Routing Engine that customer XXX is a prepaid customer and has account number 3
- A call is made to the prepaid billing server do request the balance.
- When the call completes, the Call Monitor sends a DEBIT request to the prepaid billing server.

4.2.2 Prepaid Billing API

The API has been implemented according to its specifications and is now being used by the customer portal and various applications that use the Services Engine. An example product is 3eurobellen.nl which allows customer to call to a set of destinations for 3 euro a month.

4.2.3 LCR UI (User Interface)

Some basic UI features have been implemented into the existing LCR application to allow setting the account type (prepaid vs non-prepaid) and making deposits manually.
4.3 Architectural changes

Various architectural changes were required to support the prepaid module.

- The installation of new hardware
- Installing and configuring software on the new hardware
- Implementing a fail-over mechanism on fail-over host groups.

4.3.1 Hardware installation

All hardware has been placed at the co-location site at Lancelot’s Amsterdam POP. This took around 2 hours with two employees.

The following tasks have taken place:

- Moving old hardware inside the racks to make space for the new equipment
- Installing rack sliding rails
- Mounting the machines using the sliding rails
- Power and ethernet cabling
- Testing of equipment

4.3.2 Software installation

All hardware was installed with GNU/Linux Debian as the Operating System and successfully configured for LCR platform use.

Another employee configured all new equipment to be monitored by Nagios.
4.3.3 HeartBeat

As shown by the architecture diagram heartbeat was to be configured on the groups {ares, antares} and {artemis, atlas}. This was successfully implemented on all servers. During set-up a reference document was created for future HeartBeat installations, which can be found in appendix C.

Various tests have been executed in a test-environment when the hardware was still at the office to test whether all fail-over mechanisms operated properly.

4.4 Migration from SVN to GIT

All code was successfully transferred into GIT repositories for each company project as can be seen in the Trac environment:

<table>
<thead>
<tr>
<th>Repository Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>belgipod.git</td>
</tr>
<tr>
<td>billing-platform.git</td>
</tr>
<tr>
<td>configuration.git</td>
</tr>
</tbody>
</table>
| customer-portal.git | 118956 | 31 | hours | marcus | Added PostgreSQL library Signed-off-by: Martin Ziglbauer ...
| freesim-api.git | 109546 | 6 | months | marcus | re #260 (VAG settings) |
| lancelot-lcr.git | 71700B | 8 | days | marcus | always force a reboot Signed-off-by: Matthias van der Vlies ...
| matthias-bachelor-thesis.git | 11360B | 5 | hours | marcus | minor changes Signed-off-by: Matthias van der Vlies ...
| twacg.git        | 1154570 | 2 | months | simon | Pay application |
| utilities.git    | 24670B | 7 | days | marcus | patch for cluster config causing out of date Signed-off-by: Matthias van der ...
| vopex.git        | 1129567 | 12 | months | marcus | re #460 (Reintroduce SIP/A02 peers from database) - Moved callset patch ...
| voicemail-ui.git | 19834B | 10 | months | marcus | re #234 (V1 of voicemail UI) fixed uri and wrong audit calculation due ...
| websites.git     | 183854 | 2 | weeks | marcus | use "Signed-off-by: Matthias van der Vlies ...

Figure 4.3: GIT repositories for Lancelot Telecom
4.5 Customer portal

Below are figures that show each implemented requirement. During the development the following additional requirements came up. Due to the agile development and use of a very flexible framework (Play! framework) this was very easy to implement.

![Login screen](image1)

Figure 4.4: Login screen

![Account status](image2)

Figure 4.5: Account status displaying data gathered via the Prepaid Billing API
Figure 4.6: Overview of the customer’s traffic

Figure 4.7: Details for the customer’s traffic

Figure 4.8: Show payments, account balance and provide option to make payments
Figure 4.9: Notification settings

Figure 4.10: Users management

Figure 4.11: User management
Figure 4.12: Extra: Localization for dutch customers

Figure 4.13: Extra: Branding by implementing different CSS template
Chapter 5

Conclusions
5.1 Introduction

This chapter shows conclusions based on the given assignment and its resulting output.

5.2 Overall conclusion

By doing extensive research on the existing system and setting out clear requirements, we were able to meet all requirements.

This allows Lancelot Telecom to roll out new services to their customers and provide a more flexible configuration for any future product.

5.3 Research questions

The research questions defined have extended answers which are displayed below. On most questions, an extensive answer will be given followed by a summarized version.

5.3.1 How can the billing engine be modified to do real-time prepaid billing?

Since the system needs to be able recover any given call it was a prerequisite to store the call information available into a database for further processing, and fault-recovery.

Due to the amount of traffic on any given LCR system the current architecture did not support high volumes of call billing as it was a synchronous process. This process had to be a-synchronized.

The billing engine should not be responsible for the actual debit of credits, since operation can never be assured to be atomic. This is why a SPoC was put in place to allow a gateway for this functionality.

Summarized answer

- Store call information in a database.
- Move to an asynchronous (multi-threaded) architecture to allow high performance call monitoring.
- Use the SPoC layer (Prepaid Billing Server) to debit credits.

5.3.2 What are possible scenarios that may cause money to be lost and what are possible solutions for these scenarios?

There are various dangerous scenarios that add risk to the company’s operation. Almost all of them are equally dangerous for day-to-day operation.

System failures

A system failure means customers are not able to send any traffic and existing traffic may be interrupted. Also billing data may be lost causing discrepancies in the billing scheme.

To avoid such issues the system should be suited redundantly using fail-over and load-sharing mechanisms. Each of the following critical services should be available at least twice:

- LCR Routing module
- Prepaid module
- VoIP gateways

Which can be achieved by hosting these services on at least two servers that implements a fail-over/load-sharing scheme.
Data corruption

Another major issue is data corruption. Again discrepancies in the billing scheme may cause money to be lost and poses a financial risk. By using reliable protocols we are able to minimize such risks.

Concurrency issues

Imagine a program requesting a customer’s balance, and another does the same simultaneously. The first program debits, but the second still has the old data. This may/or may not result in corruption of the data, because the operation were not atomic and did provide a locking scheme. By providing a single layer that controls access to resources we are able to ensure atomic operations. This layer is called an SPoC.

Security issues

Although not an immediate threat, this is very real when external services are connected to the platform. Since the SPoC does not provide any security scheme an API must be implemented that provides a secured proxy to such services. Access to resources can then be managed individually or revoked at any time.

Summarized answer

- System failures; impact is minimized by creating fail-over and load-sharing mechanisms.
- Data corruption; impact is minimized by using reliable protocols that use checksumming to verify data.
- Concurrency issues; By using a SPoC layer to ensure atomic operations.
- Security issues; By creating an additional layer between the SPoC layer and external services and provide security mechanisms for them.

5.3.3 Which warning levels should be defined for sending out notifications concerning account balances?

A very desirable functionality is to warn customers automatically when their balance hits critical/warning levels. We specify the following levels:

- warning
- critical
- zero

The thresholds should be configurable by the customer. However not all customers have experience yet. By company experience the following default thresholds are defined for ‘regular’ customers

<table>
<thead>
<tr>
<th>Level</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>warning</td>
<td>500</td>
</tr>
<tr>
<td>critical</td>
<td>100</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
</tbody>
</table>

5.3.4 Is it desirable to terminate calls currently in progress when the account balance hits zero?

No customer should be able to get a negative balance or at least a minimal amount. This means that the policy should be that calls are terminated by the call monitor when the calls are still in progress.
5.3.5 What should the hardware/software architecture look like to almost match the zero-fault-tolerance criteria?

To provide the fail-over/load-sharing mechanisms more hardware has to be bought based on the given requirements. Also old hardware has to be investigated for their capacity.

This translates into the following equipment and tasks:

<table>
<thead>
<tr>
<th>Server</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>apollo.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>athena.bb.lancelot-telecom.nl</td>
<td>VoIP subsystem</td>
</tr>
<tr>
<td>ares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine and Prepaid Billing Server</td>
</tr>
<tr>
<td>antares.bb.lancelot-telecom.nl</td>
<td>LCR Routing engine, and fail-over Prepaid Billing Server</td>
</tr>
<tr>
<td>artemis.bb.lancelot-telecom.nl</td>
<td>Fail-over for atlas</td>
</tr>
<tr>
<td>atlas.bb.lancelot-telecom.nl</td>
<td>LCR web interface</td>
</tr>
<tr>
<td>astro.bb.lancelot-telecom.nl</td>
<td>Unused for LCR platform</td>
</tr>
<tr>
<td>ajax.bb.lancelot-telecom.nl</td>
<td>System monitoring</td>
</tr>
</tbody>
</table>

Figure 5.1: Overview of the new architecture of Lancelot Telecom’s telephony platform
Chapter 6

Recommendations
6.1 Production deployment

By meeting all set requirements and intensive testing it is recommended to immediately deploy the prepaid module on the production systems. This way, Lancelot Telecom is able to utilize the module to its full potential and have the ability to extend their services based on this module.

6.2 Make existing customers aware of customer portal

Lancelot Telecom has existing customers which are already making manual prepayments. Lancelot Telecom should inform these customers of the availability of the customer portal so can give their customers a higher service level.

6.3 More test cases

There is a lot of functionality in the LCR that don’t have test cases associated. A lot of vital parts of the system are not automatically tested using JUnit test cases. Implementing more test cases will prevent serious problems to arise in day-to-day business as well increase the code’s quality.

6.3.1 Cobertura

Using Cobertura [Dol11] it is possible to check code coverage of test cases. To illustrate the above the Ant build script for the JUnit test cases has been altered.

The modifications that were required are explained in appendix E.1.

6.3.2 Results

The results of the Cobertura run are the following:

<table>
<thead>
<tr>
<th>Result type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>340</td>
</tr>
<tr>
<td>Line coverage</td>
<td>6% (702/11439)</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>1% (72/3864)</td>
</tr>
<tr>
<td>Complexity</td>
<td>2.107</td>
</tr>
</tbody>
</table>

Cobertura also allows to see what happens with the actual code, see figure 6.2.

Although the average McCabe cyclomatic code complexity [Wik11a] is low, there are some packages that really need refactoring to decrease code complexity, mainly:

- com.lancelot.asterisk.agi.scripts
- com.lancelot.lcr.processors
- com.lancelot.callgen

My advise would be to research what coverage is acceptable for the LCR system matched against to be defined company standards. My personal opinion is that important packages with a complexity above 2.0 should have 80% line coverage and about 100% branch coverage. The reason for defining a higher branch coverage ratio is that the LCR is relying heavily on decisions made in these packages, all decisions should be assumed to be correct and thus this requirement. The value 2.0 is based purely on analyzing all the packages individually and the conclusion was that currently all important functionality has a complexity above 2.0. As stated before important functionality is part of the decision making packages. By raising the branch coverage to 100%, line coverage will most likely rise.

I also investigated on what other developers are saying [htt11] and discovered that aiming for 100% code coverage says nothing about the actual quality of the code. In that sense it’s a useless metric, however my point concerning the big number of missing tests is easily illustrated by this metric, making it a useful tool to investigate whether a project requires more testing, not if it’s tested enough.
To solve above issues a code review would be a good idea for the LCR system. This makes the code clearer to read and increases the fault-tolerance of the application.

### 6.4 Documentation

Most important parts of the code is documented, but lots of code is still missing Javadoc documentation. Writing more code documentation makes it easier for new developers to find their way in the code.

Using the checkstyle API \cite{Tea11} in combination with a metrics plugin for Eclipse IDE I was able to derive documentation coverage and illustrate above point.

The modifications that were required are explained in appendix E.2.

This resulted into the following results:

<table>
<thead>
<tr>
<th>Result type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of methods</td>
<td>1967</td>
</tr>
<tr>
<td>Number of attributes</td>
<td>872</td>
</tr>
<tr>
<td>Total methods/attributes</td>
<td>2839</td>
</tr>
<tr>
<td>Undocumented methods/attributes</td>
<td>907</td>
</tr>
<tr>
<td>Undocumented percentage</td>
<td>31.95%</td>
</tr>
<tr>
<td>Total lines of code</td>
<td>24398</td>
</tr>
</tbody>
</table>

Note that this is only the percentage of documentation used on methods/attributes. The actual code within the methods are not checked by stylecheck, but this should give us a rough idea on what is going on. I included the total lines of code to illustrate that 70% documentation looks pretty good, however assuming most of the code is in the actual methods this value will actually be a lot lower. Therefore it’s not really possible to show complete code coverage at the moment. This is also practically impossible since some code doesn’t require documentation where other code does. By at least providing Javadoc on all methods/attributes the API documentation quality is increased, currently (December 2011) 907 methods/attributes should still be documented.
An example of undocumented code in the core functionality:

```java
public static TrafficDistributionElement<Carrier> calculateCarrierDistributionElement(
    final List<TrafficDistributionElement<Carrier>> elements, int upper) throws Exception {
    final int random = getRandom(upper);
    for (final TrafficDistributionElement<Carrier> d : elements) {
        if (random > d.lower && random <= d.upper) {
            return d;
        }
    }
    throw new Exception("No distribution element found + random");
}
```

Example of the documented version:

```java
/**
 * Calculate the carrier element using a random number mechanism.
 * @param elements The carrier elements
 * @param upper The upper limit (i.e. 100 for 100%)
 */
public static TrafficDistributionElement<Carrier> calculateCarrierDistributionElement(
    final List<TrafficDistributionElement<Carrier>> elements, int upper) throws Exception {
    final int random = getRandom(upper);
    // Find first element that has a range which matches the random number
    for (final TrafficDistributionElement<Carrier> d : elements) {
        if (random > d.lower && random <= d.upper) {
            return d;
        }
    }
    throw new Exception("No distribution element found");
}
```

Another essential part is more documentation on how the LCR system is actually working. This should be
recorded in Lancelot Telecom’s wiki to allow other developers to easily understand the processes in the LCR system. A start with this documentation has been made on the internal wiki.
Chapter 7

Appendices
.A Prepaid API documentation

.A.1 Example API controller

```java
@ZSecure("api-rw")
@ZMatch("/api/prepaid/account/${accountId}"")
public class AccountController extends AbstractAction {
    private Boolean validRequest = false;
    private Boolean validParameters = true;
    private Integer accountId;

    @Override
    public void after() throws Exception {
        final HttpServletRequest request = getRequest();
        if (validRequest) {
            if (validParameters) {
                final Account account = PrepaidDAO.getAccountById(accountId);
                if (account == null) {
                    error(404, "Not found");
                } else if ("GET".equals(request.getMethod())) {
                    try {
                        account.balance = PrepaidBillingClient.getBalance(account);
                        renderXml(account);
                    } catch (Exception e) {
                        error(503, e.getMessage());
                    }
                } else if ("PUT".equals(request.getMethod())) {
                    try {
                        final Account xmlAccount = XMLUtils.deserialiseFromXML(request.getInputStream(), Account.class);
                        if (xmlAccount.notificationsEnabled != null) {
                            account.notificationsEnabled = xmlAccount.notificationsEnabled;
                        }
                        if (xmlAccount.ceilingBalanceThreshold != null) {
                            account.ceilingBalanceThreshold = xmlAccount.ceilingBalanceThreshold;
                        }
                        if (xmlAccount.floorBalanceThreshold != null) {
                            account.floorBalanceThreshold = xmlAccount.floorBalanceThreshold;
                        }
                        if (xmlAccount.fromNotificationAddress != null) {
                            account.fromNotificationAddress = xmlAccount.fromNotificationAddress;
                        }
                        if (xmlAccount.toNotificationAddress != null) {
                            account.toNotificationAddress = xmlAccount.toNotificationAddress;
                        }
                        new GenericDAO().update(account);
                        renderXml(account);
                    } catch (Exception e) {
                        error(400, e.getMessage());
                    }
                }
            } else {
                error(500, "Bad request");
            }
        } else {
            error(405, "Method not allowed");
        }

    @Override
    public void before() throws Exception {
        final HttpServletRequest request = ZTemplates.getService().getRequest();
        if (request.getMethod().equals("GET") || request.getMethod().equals("PUT")) {
            validRequest = true;
        }
    }

    @ZSetter("accountId")
    public void setAccountId(final String accountId) {
        try {
            this.accountId = Integer.valueOf(accountId);
        } catch (Exception e) {
            validParameters = false;
        }
    }
}
```
### A.2 API resources

<table>
<thead>
<tr>
<th>URI</th>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/prepaid/account/accountId</td>
<td>GET</td>
<td>accountId</td>
<td>Get account information</td>
</tr>
<tr>
<td>/api/prepaid/account/accountId</td>
<td>PUT</td>
<td>accountId</td>
<td>Modify account information</td>
</tr>
<tr>
<td>/api/prepaid/account/accountId/payments</td>
<td>GET</td>
<td>accountId</td>
<td>Get list of payments for an account</td>
</tr>
<tr>
<td>/api/prepaid/account/accountId/payment</td>
<td>POST</td>
<td>accountId, amount, expiry</td>
<td>Topup credits</td>
</tr>
<tr>
<td>/api/prepaid/account/accountId/debit?amount=amount</td>
<td>PUT</td>
<td>accountId, amount</td>
<td>Debit credits</td>
</tr>
</tbody>
</table>

### A.3 GET /api/prepaid/account/accountId

Request example:

```
GET /api/prepaid/account/1
```

Success example response:

```
200 OK
<account enable-notifications="false" id="1">
  <notification-from-address>
    lcr@lancelot-telecom.nl
  </notification-from-address>
  <notification-to-address>
    matthias@lancelot-telecom.nl
  </notification-to-address>
  <balance-floor-threshold>100</balance-floor-threshold>
  <balance-ceilinng-threshold>500</balance-ceilinng-threshold>
  <balance>9.4331</balance>
</account>
```

Failure example:

```
404 Not found
```

### A.4 PUT /api/prepaid/account/accountId

Request example:

```
PUT /api/prepaid/account/1
```

Success example response:

```
200 OK
```

Failure example:

```
404 Not found
```
.A.5 GET /api/prepaid/account/accountId/payments

Request example:

GET /api/prepaid/account/1/payments

Success example response:

200 OK
<payments id="1">
  <payment id="10">
    <amount>10.00</amount>
    <balance>8.12</balance>
    <date>2011-01-01</date>
    <expiry>2011-06-06</expiry>
  </payment>
</payments>

Failure example:

404 Not found

.A.6 POST /api/prepaid/account/accountId/payment

Request example:

POST /api/prepaid/account/1/payment?amount=18.00&expiry=2011-07-07

Success example response:

200 OK

Failure example:

404 Not found

Testing:


.A.7 PUT /api/prepaid/account/accountId/debit

Request example:

PUT /api/prepaid/account/1/debit?amount=18.00

Success example response:

200 OK
<account enable-notifications="false" id="1">
  <notification-from-address>
    lcr@lancelot-telecom.nl
  </notification-from-address>
  <notification-to-address>
    matthias@lancelot-telecom.nl
  </notification-to-address>
  <balance-floor-threshold>100</balance-floor-threshold>
  <balance-ceiling-threshold>500</balance-ceiling-threshold>
  <balance>9.4331</balance>
</account>

Failure example:

404 Not found

Testing:

.B  Prepaid Billing Server protocol

.B.1  Requests

Getting current balance:

\texttt{BALANCE \{accountId\} \{checksum\}}

Debit credits:

\texttt{DEBIT \{accountId\} \{amount\} \{(resource)\} \{checksum\}}

Topup credits:

\texttt{TOPUP \{accountId\} \{amount\} \{(expiry)\} \{checksum\}}

.B.2  Responses

Account not found:

\texttt{NOTFOUND \{checksum\}}

Failure:

\texttt{ERROR \{checksum\}}

Debit success:

\texttt{SUCCESS \{checksum\}}

Balance success example:

\texttt{10.00 \{checksum\}}

Checksum error:

\texttt{CHKSMERR}

.B.3  Checksumming

The checksum consists out of the following hash string:

\texttt{\{command\}==\{length\}}

Example:

\texttt{BALANCE 1234=12}

Notice that 12 is the length of:

\texttt{BALANCE 1234}

To generate the checksum using a command line (for example testing via telnet):

\texttt{\$ echo "BALANCE\_1234\=12" | sha1sum | awk \{'print\_\$1'\}}

Result:

\texttt{513ae6194b75edca5b8cc2cfc1277385daa3b84}

.B.4  Expiry date format

\texttt{yyyy-MM-dd}
.C HeartBeat configuration manual

.C.1 Installation

Install the debian package on both servers using:

$ apt-get install heartbeat

.C.2 Configuration

In this example we have the following two hosts:

- ares: 10.36.84.105 (eth1)
- antares: 10.36.84.106 (eth1)

They will be used as nodes for the 10.36.84.104 address.

On both machines do the following:

$ echo "net.ipv4.ip_nonlocal_bind=1" >> /etc/sysctl.conf
$ sysctl -p

Create the file /etc/ha.d/authkeys on both hosts with the following content:

auth 3
3 md5 somerandomstring

And perform some security:

$ chmod 600 /etc/ha.d/authkeys

Check host name ($hostname in next example) with:

$ uname -n

This should be ares or antares depending on the host you are on. Make sure it matches the next configuration files.

On ares edit /etc/ha.d/ha.cf:

```bash
# keepalive: how many seconds between heartbeats
keepalive 2
# deadtime: seconds to declare host dead
deadtime 10
# What UDP port to use for udp or ppp–udp communication?
udpport 694
bcast eth1
mcast eth1 225.0.0.1 694 1 0
ucast eth1 10.36.84.106
# What interfaces to heartbeat over?
udp eth1
# Facility to use for syslog()/logger (alternative to log/debugfile)
logfacility local0
# Tell what machines are in the cluster
node ares... -- must match uname -n
node antares
```

On antares edit /etc/ha.d/ha.cf:

```bash
# keepalive: how many seconds between heartbeats
keepalive 2
# deadtime: seconds to declare host dead
deadtime 10
# What UDP port to use for udp or ppp–udp communication?
udpport 694
bcast eth1
mcast eth1 225.0.0.1 694 1 0
ucast eth1 10.36.84.105
```
# What interfaces to heartbeat over?
udp eth1
#
# Facility to use for syslog()/logger (alternative to log/debugfile)
# logfacility local0
#
# Tell what machines are in the cluster
# node nodename ... -- must match uname -n
node antares
node antares

Now on both machines create the file /etc/ha.d/haresources with the following contents:

```
ares IPaddr::10.36.84.104/24/eth1
```

And run the heartbeat services:

```
$ /etc/init.d/heartbeat start
```
The following example shows a test suite containing test cases for testing the Prepaid Billing Server.

```java
public class TestPrepaidBillingServer {
    final Account account = new Account(1000);

    @Before
    public void setupConfiguration() {
        Configuration.getProperties().setProperty("module.lcr.prepaid.server", "127.0.0.1");
    }

    @Test
    public void testBalance() throws Exception {
        assertEquals(new Double(0.0), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testDebit() throws Exception {
        assertEquals(new Double(-1.1204), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testDebitWithResource() throws Exception {
        PrepaidBillingClient.debit(account, 1.1204);
        final Debit debit = (Debit) new GenericDAO().get(Debit.class, 1);
        assertEquals(new Double(1.1204), debit.amount);
        assertEquals("R1", debit.resource);
    }

    @Test
    public void testTopup() throws Exception {
        PrepaidBillingClient.topup(account, 2.2408);
        assertEquals(new Double(0.0), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testDebitSecondPass() throws Exception {
        PrepaidBillingClient.debit(account, 1.1204);
        assertEquals(new Double(8.8796), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testTopupWithExpiry() throws Exception {
        final Payment payment = (Payment) new GenericDAO().get(Payment.class, 4);
        assertEquals(new Double(1.1204), payment.amount);
        assertEquals(new Double(1.1204), payment.balance);
        assertEquals(new SimpleDateFormat("yyyy-MM-dd").parse("2020-01-01"), payment.expiryDate);
    }

    @Test
    public void testTopupWithExpiryPassTwo() throws Exception {
        // expiry in past, balance should be unmodified
        PrepaidBillingClient.topup(account, 1.1204, "2000-01-01");
        assertEquals(new Double(10.0), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testInvalidChecksum() throws Exception {
        final String data = "BALANCE:1000" + "INVALID" + PrepaidBillingServer.EOL;
        final RemoteCommand command = new RemoteCommand(Configuration.getPrepaidBillingServerAddress(), 8989, data);
        assertFalse(command.execute());
        assertTrue(command.getLastResponse().startsWith("CHKSMERR"));
    }

    @Test
    public void testNotFoundError() throws Exception {
        PrepaidBillingClient.topup(new Account(12345678), 1234.0, "INVALID");
    }

    @Test
    public void testStillSameBalance() throws Exception {
        assertEquals(new Double(10.0), PrepaidBillingClient.getBalance(account));
    }

    @Test
    public void testNotFound() throws Exception {
        PrepaidBillingClient.getBalance(new Account(12345678));
    }
}
```
E Complementary technical data for recommendations chapter

E.1 Integrating cobertura

The following code was added to/modified in the build script (GIT diff output):

diff --git a/build-test.xml b/build-test.xml
index 0ca63c2..4f0b9e8 100644
--- a/build-test.xml
+++ b/build-test.xml
@@ -1,6 +1,7 @@

<project default="test" names="lancelot-test" basedir="."/>

++<property file="${basedir}/etc/ant/build.properties"/>
++<property name="lib.dir" location="${{basedir}}/lib"/>
++<property name="classes.dir" location="${{basedir}}/classes"/>
++<property name="cobertura.classes.dir" location="${{basedir}}/instrumented-classes"/>
++<property name="cobertura.documentation.dir" location="${{basedir}}/cobertura"/>
++
++<target name="init">-
++  <mkdir dir="${{basedir}}"/>
++  <mkdir dir="${{classes}}"/>
++  <copy file="${{src-dir}}/hibernate.cfg.xml" todir="${{classes}}" overwrite="true"/>
++  <copy file="${{src-dir}}/import.sql" todir="${{classes}}" overwrite="true"/>
++</target>

-<target name="clean" description="Clean the build directory">-
-  <delete dir="${{build-dir}}"/>
-<target>
++
++<target name="test" depends="compile">+
++  <taskdef classpathref="cobertura.class.path" resource="tasks.properties"/>
++  <target name="test" depends="init,compile">+
++    <Instrument classes/>
++    <fact file="${{cobertura.classes.dir}}" classpathref="cobertura.class.path"/>
++    <filesset dir="${{classes}}"/>
++    <fileset/>
++    <cobertura.instrument/>
++    <junit failonerror="true" failonfailure="false" haltonerror="true" haltonfailure="false"/>
++    <classpath refid="test.classpath" />+
++    <formatter type="brief" usefile="false"/>
++</target>
++
++<target name="test" depends="init,compile">+
++  <taskdef classpathref="cobertura.class.path" resource="tasks.properties"/>
++  <target name="test" depends="init,compile"></target>
++  <taskdef classpathref="cobertura.class.path" resource="tasks.properties"/>
++  <target name="test" depends="init,compile"></target>
++  <taskdef resource="checkstyle\task.properties" classpath="\lib\checkstyle-5.5/\checkstyle-5.5-all.jar"/>
++  <target name="test" depends="init,compile">+
++</target>

E.2 Integrating stylecheck
I also had to configure stylecheck with a configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//PuppyCrawl/DTD_Check_Configuration_1.3//EN" "http://www.puppycrawl.com/dtds/configuration_1.3.dtd">

<module name="Checker">
  <module name="JavadocPackage"/>
  <module name="TreeWalker"/>
</module>
```

I also had to configure stylecheck with a configuration file: