Functional Reactive Programming as programming model for telecom server software

by

Klervie Toczé

LiTH-IDA/ERASMUS-A--14/003—SE

2014-12-18
Final Thesis

Functional Reactive Programming as programming model for telecom server software

by

Klervie Toczé

LiTH-IDA/ERASMUS-A--14/003—SE

2014-12-18

Supervisor: Maria Vasilevskaya
Examiner: Simin Nadjm-Tehrani
Abstract

This thesis studies the use of the functional reactive programming (FRP) framework reactive-banana in a prototype which simulates a part of a Long-Term Evolution (LTE) base station: the Radio Resource Control connection setup procedure. The investigated problem is to determine whether using this FRP framework leads to an implementation with suitable performance and improved maintainability compared to the current implementation. Enhancing the maintainability of the base station software enables quicker and more efficient maintenance activities, which lead to an improved customer satisfaction. Moreover, it means that less programmers need to work on maintenance, so they can work on developing new products instead.

In order to compare the use of the FRP paradigm to the one currently used in the base station implementation, the object-oriented programming (OOP) paradigm, a second prototype using this paradigm was also implemented. Having two prototypes implementing the same designed reference model (which is a simplified version of the Radio Resource Control connection setup procedure) enables a relevant comparison of the two paradigms. The two prototypes were then compared in terms of performance and maintainability. The maintainability evaluation consisted in using both software metrics and experts’ assessment, as this has been proven to be the most efficient way to evaluate software maintainability. Four experts were asked to fill in a questionnaire after reviewing the code of the two implementations.

The comparison of the two prototypes indicates that the FRP prototype is more maintainable than the OOP one, but the OOP prototype has better performances than the FRP one. Moreover, the performance of the FRP prototype during the conducted tests indicates that such an implementation of the FRP paradigm is not suitable for a real base station.
Acknowledgements

First I would like to thank my thesis examiner, professor Simin Nadjm-Tehrani, for letting me work on this thesis topic and for her useful insights given along the thesis work. I also thank my university supervisor, Maria Vasilevskaya, for our productive discussions and her useful pieces of advice. Thanks also for your comments on this report. Furthermore, I thanks Kristian Sandahl for his advice on the different questionnaires used in the thesis work.

Then, I would like to thank my supervisor at Ericsson, Patrik Sandahl, particularly for being always available and very helpful when a problem or question arose. I also want to thank all the Ericsson experts who found some time in their very busy schedules for contributing to my thesis. Thanks a lot as well to all my colleagues at 3Gsim for being very welcoming and helping me a lot to improve my Swedish. *Tack så mycket!*

I would also like to thank Michael for his great support and understanding, particularly during the report writing period.

Finally, my studies in a whole and this thesis work in particular would have been a lot more difficult to achieve without the support and encouragement of my family. I will always be immensely grateful for that.

*Klervie Točzé*
*Linköping, Sweden*
*January, 2015*
# Contents

1 Introduction ................................. 1  
1.1 Motivation ................................ 1  
1.2 Problem definition ...................... 2  
1.3 Goals .................................. 2  
1.4 Methodology ............................. 2  
1.5 Intended Audience ...................... 3  
1.6 Limitations ............................. 3  
1.7 Thesis outline ......................... 4  

2 Telecom application background .... 5  
2.1 Long-Term Evolution architecture .... 5  
  2.1.1 Long-Term Evolution network overview 6  
  2.1.2 eNodeB architecture ................. 6  
2.2 Radio Resource Control Protocol .... 7  
  2.2.1 Location in the protocol stack .... 7  
  2.2.2 States of the User Equipment ...... 8  
  2.2.3 Connection setup procedure ...... 8  

3 Functional Reactive Programming .... 10  
3.1 Functional Programming ............... 10  
3.2 Reactive Programming ................ 11  
3.3 Functional Reactive Programming .... 12  
  3.3.1 Evolution of the FRP paradigm .... 12  
  3.3.2 Advantages and disadvantages of FRP 15  
3.4 Alternative approaches for reactive programming 15  
  3.4.1 Akka ................................ 15  
  3.4.2 Erlang ............................. 16  
  3.4.3 ReactiveML ......................... 16  
3.5 FRP and the telecom servers .......... 16  

4 Survey of the FRP frameworks ....... 18  
4.1 Methodology ......................... 18  
4.2 Presentation of the preliminary selected frameworks .... 19  
  4.2.1 Scala.React ..................... 19  
  4.2.2 Reactive-banana ................. 20  
  4.2.3 Yampa ........................... 20
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.4</td>
<td>TimeFlies</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>Result of the survey</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>Prototype design and implementation</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>Methodology</td>
<td>23</td>
</tr>
<tr>
<td>5.2</td>
<td>Reference model</td>
<td>24</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Simplified RRC connection setup procedure</td>
<td>24</td>
</tr>
<tr>
<td>5.2.2</td>
<td>System architecture</td>
<td>26</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Deployment details</td>
<td>26</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Simulation parameters</td>
<td>27</td>
</tr>
<tr>
<td>5.3</td>
<td>Implementations</td>
<td>27</td>
</tr>
<tr>
<td>5.3.1</td>
<td>FRP implementation</td>
<td>27</td>
</tr>
<tr>
<td>5.3.2</td>
<td>OOP implementation</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Performance evaluation</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Key performance indicators</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>Methodology</td>
<td>38</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Testing environment</td>
<td>38</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Testing procedure</td>
<td>39</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Testing scenarios</td>
<td>39</td>
</tr>
<tr>
<td>6.3</td>
<td>Results</td>
<td>39</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Results for Scenario 1</td>
<td>40</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Results for Scenario 2</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>Maintainability evaluation</td>
<td>49</td>
</tr>
<tr>
<td>7.1</td>
<td>Expert evaluation - Initial thoughts</td>
<td>49</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Methodology</td>
<td>49</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Outcome</td>
<td>51</td>
</tr>
<tr>
<td>7.2</td>
<td>Expert evaluation - Assessment</td>
<td>52</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Methodology</td>
<td>52</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Outcome</td>
<td>55</td>
</tr>
<tr>
<td>7.3</td>
<td>Software Metrics</td>
<td>62</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Methodology</td>
<td>62</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Outcome</td>
<td>63</td>
</tr>
<tr>
<td>7.4</td>
<td>Conclusion of the maintainability evaluation</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions and future work</td>
<td>67</td>
</tr>
<tr>
<td>8.1</td>
<td>Conclusions</td>
<td>67</td>
</tr>
<tr>
<td>8.2</td>
<td>Future investigations</td>
<td>68</td>
</tr>
<tr>
<td>8.2.1</td>
<td>About performance</td>
<td>68</td>
</tr>
<tr>
<td>8.2.2</td>
<td>About maintainability</td>
<td>69</td>
</tr>
<tr>
<td>8.2.3</td>
<td>About the FRP paradigm in general</td>
<td>69</td>
</tr>
<tr>
<td>8.2.4</td>
<td>About the prototypes</td>
<td>69</td>
</tr>
<tr>
<td>8.3</td>
<td>Personal reflection on the thesis work</td>
<td>69</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Survey of the frameworks</td>
<td>76</td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Messaging sequences</td>
<td>78</td>
</tr>
<tr>
<td>C</td>
<td>Content of the UE context</td>
<td>80</td>
</tr>
<tr>
<td>D</td>
<td>Interview questionnaires</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>D.1 Preliminary questionnaire</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>D.2 Interview questionnaire</td>
<td>82</td>
</tr>
<tr>
<td>E</td>
<td>Evaluation questionnaire</td>
<td>85</td>
</tr>
<tr>
<td>F</td>
<td>Analysis of the maintainability assessment</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>F.1 Answers frequencies</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>F.2 Descriptive statistics</td>
<td>94</td>
</tr>
</tbody>
</table>
List of Figures

2.1 High-level overview of a LTE network ........................................ 6
2.2 High-level overview of a LTE base station .................................. 7
2.3 Uu control-plane protocol stack ................................................... 7
2.4 Messaging sequence of a successful connection setup procedure in the reference model .................................................. 9

5.1 Overview of the reference model .................................................... 26
5.2 Deployment view of the reference model ....................................... 27
5.3 Module coupling in the FRP prototype ......................................... 28
5.4 Class diagram of the OOP prototype ............................................ 33

6.1 Average total time (with standard deviation) elapsed on the UE side for powering on given numbers of UEs (FRP prototype) ........................................ 40
6.2 Average time (with standard deviation) elapsed on the UE side for attaching a UE in the FRP prototype for 100, 500 and 1000 UEs powered on .................................................. 41
6.3 Average productivity (with standard deviation) of the FRP prototype programs according to the number of UEs powered on .................................................. 42
6.4 Average productivity (with standard deviation) of the FRP prototype programs (with classic functional UE program) according to the number of UEs powered on .................................................. 43
6.5 Average total time elapsed on the UE side for powering on given numbers of UEs (FRP prototype with classic functional UE program) ........................................ 43
6.6 Average time (with standard deviation) elapsed on the UE side for attaching a UE in the FRP prototype (with classic functional UE program) for 100, 500 and 1000 UEs powered on .................................................. 44
6.7 Average total time elapsed on the UE side for powering on given numbers of UEs (OOP prototype) ................................................. 44
6.8 Average time (with standard deviation) elapsed on the UE side for attaching a UE in the OOP prototype for 100, 500 and 1000 UEs powered on .................................................. 45
6.9 Average time elapsed on the UE side for attaching a UE in the FRP prototype .................................................. 46
6.10 Average percentage (with standard deviation) of CPU time (CPU) and available physical memory (MEM) used by the eNodeB program in the FRP prototype .................................................. 47
6.11 Average time elapsed on the UE side for attaching a UE in the OOP prototype .................................................. 47
6.12 Average percentage (with standard deviation) of CPU time (CPU) and available physical memory (MEM) used by the eNodeB program in the OOP prototype .................................................. 48

7.1 Average score on each question for both prototypes ................................................. 58
7.2 Frequencies of the answers to question 32 ....................................... 59
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>Frequencies of the answers to question 33</td>
<td>60</td>
</tr>
<tr>
<td>7.4</td>
<td>Frequencies of the answers to question 34</td>
<td>60</td>
</tr>
<tr>
<td>7.5</td>
<td>Frequencies of the answers to question 35</td>
<td>61</td>
</tr>
<tr>
<td>7.6</td>
<td>Frequencies of the answers to question 36</td>
<td>61</td>
</tr>
<tr>
<td>B.1</td>
<td>Messaging sequence during a failed connection setup procedure: rejection of the RRC connection request</td>
<td>78</td>
</tr>
<tr>
<td>B.2</td>
<td>Messaging sequence during a failed connection setup procedure: failure of the security mode</td>
<td>79</td>
</tr>
<tr>
<td>B.3</td>
<td>Messaging sequence during a failed connection setup procedure: failure of the RRC connection reconfiguration</td>
<td>79</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Content of two successive messages .............................................. 25
5.2 Data stored in each UE ......................................................... 25

7.1 Types of question examples from the interview questionnaire ............. 50
7.2 Table explaining the meaning of the different scale items .................. 54
7.3 Maintainability levels according to the score on a question ................. 56
7.4 Mean and standard deviation for the OOP prototype ....................... 56
7.5 Mean and standard deviation for the FRP prototype ........................ 57
7.6 Questions for the second task .................................................. 59
7.7 Code size metrics in the FRP implementation ................................ 63
7.8 Code size metrics in the OOP implementation ................................ 64

A.1 Framework list summarizing the preliminary selection outcome .......... 76
A.2 Characteristics needed for the framework to be chosen .................... 77

C.1 Data stored for each UE in the MME ...................................... 80
C.2 Data stored for each UE in the eNodeB ..................................... 80

F.1 Frequencies of the answers for the OOP prototype ....................... 93
F.2 Frequencies of the answers for the FRP prototype ........................ 94
F.3 Descriptive statistics for the OOP prototype ................................ 94
F.4 Descriptive statistics for the FRP prototype ................................ 95
List of Snippets

4.1 Currency converter example implemented in Scala.React
4.2 Currency converter example implemented in reactive-banana
4.3 Currency converter example implemented in Yampa
4.4 Currency converter example implemented in TimeFlies
5.1 Creating and powering up several UEs
5.2 Set up of the event network
5.3 Message reception decoding and sending to the event network
5.4 Handling of the RRC Connection Request message
5.5 Partial set up of the epoll instance
5.6 Reception, decoding and handling beginning of one UE message
5.7 Extract of the handleUeMessage function
5.8 Function handleRrcConnectionRequest
List of Acronyms

2G second generation.
3G third generation.
3GPP 3rd Generation Partnership Project.
4G fourth generation.
AFOTEC Air Force Operational Test and Evaluation Center.
AFRP Arrowized FRP.
API Application Programming Interface.
C-RNTI Cell-Radio Network Temporary Identifier.
CPU Central Processing Unit.
E-FRP Event-driven FRP.
EPC Evolved Packet Core.
FRP Functional Reactive Programming.
GCC GNU Compiler Collection.
GHC Glasgow Haskell Compiler.
GUI Graphical User Interface.
IMSI International Mobile Subscriber Identity.
JVM Java Virtual Machine.
LTE Long-Term Evolution.
MAC Medium Access Control.
Acronyms

MME  Mobility Management Entity.
OOP  Object-Oriented Programming.
P-FRP Priority-based FRP.
PDCP Packet Data Convergence Protocol.
PHY  physical layer.
RAM  Random-Access Memory.
RLC  Radio Link Control.
RNTI Radio Network Temporary Identifier.
RRC  Radio Resource Control.
RT-FRP Real-time FRP.
SRB  Signalling Radio Bearer.
TCP  Transmission Control Protocol.
UE  User Equipment.
UEH  User Equipment Handling.
UMTS Universal Mobile Telecommunication System.
Chapter 1

Introduction

This thesis work (30 ECTS) was done in partial fulfilment of a French engineering degree in Computer Science at the University of Technology of Compiègne and of a master degree in Computer Science at Linköping University. It was examined at Linköping University, Department of Computer and Information Science, as part of a dual degree programme. The work was carried out at Ericsson in Linköping.

Nowadays, telecommunication systems involve lots of complex software components. The telecom protocols themselves are complex but the fact that they are programmed using traditional software engineering techniques adds even more complexity, called accidental complexity [20], to the whole program. Indeed, the current programming model used in the telecom servers doesn’t handle events and other reactive behaviours, which are the core of the telecom protocols, without a lot of additional code dedicated to them. This creates a program that is more complex than strictly needed and thus, less readable and maintainable. The aim of this thesis is to investigate a new programming model which could lead to a less complex code for the telecom servers.

Functional Reactive Programming (FRP) is a quite new programming model [28] which is dedicated to deal with reactive systems. It has already successful applications in different domains such as Graphical User Interfaces (GUIs), video games, and robotics. The aim of this thesis is to determine whether this programming model can be used in the telecom servers as well.

1.1 Motivation

Reducing as much as possible the accidental complexity of the telecom servers is very important as it is closely associated to increased maintainability. First, this is beneficial because the resources (and the associated cost) that are no longer needed to be spent on maintenance tasks can be used to create new products with added value. Secondly, if the maintenance tasks are easier to perform for the programmer, it will lead to a more efficient and quicker maintenance which will increase customer satisfaction.
CHAPTER 1. INTRODUCTION

1.2 Problem definition

The question addressed in this thesis is whether using Functional Reactive Programming as a programming model leads to a suitably performing and more maintainable program compared to the current Object Oriented Programming paradigm. To address the problem in the telecom context, the implementation of the Radio Resource Control protocol in a Long-Term Evolution base station will be considered.

1.3 Goals

The goals which were set for the thesis were the following:

- To perform a theoretical investigation of Functional Reactive Programming (FRP) to find out how this programming model relates to reactive behaviours typical for telecom server software.
- To survey the existing FRP frameworks, in particular, the Haskell framework reactive-banana [6] and the Scala framework Scala.Reactive [39] in order to select the most appropriate for implementing a prototype.
- To develop one prototype showing how a typical reactive pattern from telecom software can look like when implemented in a FRP framework.
- To develop one prototype showing how the same reactive pattern can look like when implemented with the currently-used object-oriented programming paradigm.
- To compare the performance of both prototypes, together with insights about the performance of the current telecom server software, in order to determine whether the FRP prototype performs suitably.
- To obtain insights about the current telecom server software maintainability in order to know which maintainability characteristics are the most important.
- To compare the maintainability of the two prototypes using both expert assessment and software metrics.

1.4 Methodology

The thesis began by a study of the connection setup procedure of the Radio Resource Control (RRC) protocol, the part of the Long-Term Evolution (LTE) base station which is modelled in the thesis. The aim was to get familiar with the procedure and also to identify its typical reactive behaviours.

The thesis continued with a literature study of FRP. The aim was to get to know this programming paradigm and to select a suitable framework for implementing the reference model. The selection was made according to criteria such as the amount of documentation and example available and the type of applications already using the framework. It was done first using the literature specific to the frameworks and then completed by small experimentations for four pre-selected frameworks. This approach was chosen due to time limitation which made experimenting with every framework impossible. Moreover, practical experimentations were
1.5. INTENDED AUDIENCE

This thesis is intended to be understandable by anyone without prior knowledge of telecom applications but with basic programming knowledge. Being familiar with object-oriented and functional programming is an advantage but already knowing the FRP paradigm is not required as it will be presented.

1.6. LIMITATIONS

The intent of this thesis was not to reimplement the whole LTE base station using functional reactive programming, but to focus on the parts of this implementation dealing with event handling. Thus, the prototypes are implementing only a simplified version of a procedure which is part of one of the protocols used in the base station: the RRC connection setup procedure.
1.7 Thesis outline

This thesis is organized as follows:

- The chapter 2 will describe the telecom application background of the thesis, including a presentation of the LTE base station and of the RRC connection setup procedure.
- The chapter 3 will present the functional reactive programming paradigm, as well as related works.
- The chapter 4 will explain how the survey of the FRP frameworks was conducted and present its outcome.
- The chapter 5 will present the reference model used for the design of the two prototypes and details about their implementations.
- The chapter 6 will explain how the performance evaluation of the two prototypes was performed and describe its outcomes.
- The chapter 7 will describe the maintainability evaluation of the two prototypes and its outcomes.
- The chapter 8 will present the conclusions of the thesis work and ideas for future investigations.
Chapter 2

Telecom application background

This thesis was carried out within a specific field of application: the telecom servers. In this chapter, basic background about these telecom servers will be given.

2.1 Long-Term Evolution architecture

The telecom server architecture used in the thesis is the LTE architecture. LTE is a standard for wireless data communications technology. It is the latest evolution of the Universal Mobile Telecommunication System (UMTS) standard, commonly called third generation (3G).

The LTE standard is developed by the 3rd Generation Partnership Project (3GPP), which is a collaboration between six telecommunications standard development organizations based in different parts of the world. One of the primary goals of this collaboration is to have globally applicable standards for mobile communications, on the contrary to the first and second generations where several technologies were deployed nationally or regionally [1].

Driven by the growing popularity of smartphones and other mobile devices, the aim of the LTE was to increase the capacity and speed of wireless data networks, as well as to simplify the network architecture. It uses a larger wireless spectrum than the precedent second generation (2G) and 3G standards, meaning that both the users and the operators need to invest in compatible equipments to use the new features of the LTE.

The first LTE standard (Release 8) was finalized in December 2008. It has been greatly improved with the release 10, finalized in March 2011, thus taking the name of LTE-Advanced from this release. This improved version of the standard complies with the requirements set by the International Telecommunication Union in order to belong to the fourth generation (4G). However, this version of the standard is backwards compatible so LTE and LTE-Advanced terminals can both use either LTE or LTE-Advanced networks [1]. Some improvements of LTE-Advanced are [62]:

- an increased peak data rate beyond 1 Gbps
- a higher spectral efficiency
- an increased number of simultaneously active subscribers
- improved performance at cell edges.

This thesis considers the release 12 of the 3GPP standard.


### 2.1.1 Long-Term Evolution network overview

The LTE network architecture is composed of three main high-level components [57]:

- the **User Equipment (UE)**
- the **Evolved UMTS Terrestrial Radio Access Network (E-UTRAN)**
- the **Evolved Packet Core (EPC)**

The **UE** is the device that the end-user uses to communicate over the network. It can be a mobile phone but also a mobile broadband adapter used by a computer, a tablet or any other device.

The **E-UTRAN** is the access part of the network. It handles the radio communications between the **UEs** and the **EPC**. The **E-UTRAN** is composed of base stations, called eNodeBs, physically spread all over the area where the telecommunication service is accessible.

The **EPC** is the core part of the network, which handles the communication with packet data networks in the outside world, like the internet. It is itself composed of several components. During the thesis, we will only consider the **Mobility Management Entity (MME)** of the **EPC** which processes the non-radio signalling between the **UE** and the core network. We will more specifically focus on its functions related to connection management [9].

Figure 2.1 presents these three main components of the LTE architecture, as well as the three interfaces (Uu, X2 and S1) which enable the communication between them.

### 2.1.2 eNodeB architecture

Figure 2.2 shows a high-level overview of the hardware organization in a LTE-Advanced base station. A base station is composed of an antenna which will receive all the radio signalling coming over the air from the different **UEs** that are located in its range. Then, this radio signalling is handled by several digital signal processors depicted as dark-blue rectangles gathered in the Radio Units box in Figure 2.2. Those Radio Units then transmit information to the Digital Unit part of the eNodeB, using some real-time signalling equivalent to the signalling used in an ordinary network application. The interface used between a radio unit and a digital unit is called the Common Public Radio Interface. The role of the Digital Unit is to take care of everything which is not related to the radio signalling, like traffic handling, communications with the **MME** or **UE** handling. The Digital Unit is an ordinary Linux-based computer. The number of radio units and digital units depends on the type of base station.
2.2 RADIO RESOURCE CONTROL PROTOCOL

The RRC protocol that was implemented during the thesis is part of the User Equipment Handling (UEH) software layer of the eNodeB. This software layer is processed by the Main Processor, a part of the Digital Unit.

2.2 Radio Resource Control Protocol

Since a telecom server consists of a lot of protocols and procedures, it was chosen for the thesis to only implement a part of the RRC protocol, the connection setup procedure. In this part, some information will be given about the RRC protocol in general and about the connection setup procedure in particular.

2.2.1 Location in the protocol stack

The LTE radio protocols over the Uu and S1 interfaces are separated into two categories: the user plane protocols and the control plane protocols. On the one hand, in the user plane, the protocols implement the actual E-UTRAN radio access bearers which will carry the user data. On the other hand, in the control plane, the protocols are responsible for the controlling of those bearers and for several different aspects of the connection between the UE and the network [9].

As it is shown in Figure 2.3, the RRC protocol is on top of the Uu control-plane protocol.
CHAPTER 2. TELECOM APPLICATION BACKGROUND

stack. Lower layers are: Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC), Medium Access Control (MAC) and the physical layer (PHY) that handles the actual transmission.

The RRC protocol is thus a high-level protocol, mainly responsible for the establishment, maintenance, handover to another eNodeB and release of a RRC connection between the UE and the E-UTRAN. It also, among other functions, broadcasts system information for both the non-access stratum (between the UE and the EPC) and the access stratum (between the UE and the eNodeB) and performs integrity protection of the RRC messages [9].

2.2.2 States of the User Equipment

The radio functionalities handled by the control plane depend on the state of the considered UE. A UE can be in two states: idle or connected [9].

An idle UE is not connected to the network. In order to become connected, it performs uplink synchronization and the RRC connection setup procedure.

A connected UE transmits and receives control messages and user data to and from the network. In this state, the network controls the UE mobility, which means that it decides when the UE must move to another cell.

2.2.3 Connection setup procedure

The connection setup procedure occurs when a UE wants to attach itself to an eNodeB in order to be able to communicate over the network. This situation occurs when someone powers on a smartphone for example. The connection setup procedure involves three actors:

- a UE which initiates the connection setup procedure towards an eNodeB.
- an eNodeB, which will handle the connection setup procedure.
- a MME which will handle the core network part of the connection setup procedure.

During the connection setup procedure, those three actors will exchange information in order to establish a secured connection between a UE and an EPC. The messaging sequence of a successful connection setup procedure can be found in Figure 2.4.

The two first messages are the initial contact between a UE and an eNodeB. Temporary connection identifiers and configuration are exchanged. Then, the two next messages enable the establishment of a dedicated connection between the UE and the eNodeB, with permanent identifiers and configuration. The eNodeB also selects a MME for the next steps of the procedure. The following two messages convey the attach request from the UE to the MME through the eNodeB. Then, the core network asks for further configuration using the SIAP Initial Context Setup Request message. The next four messages configure the encryption of the messages (Security Mode messages) and the network technologies that the UE can handle (UE Capability messages). Finally, the last four messages perform the last steps needed for the attach to be completed. The RRC Connection Accept message, which is not part of the standard, is used to define when a UE is attached on the UE side in our reference model. At the end of this procedure, the UE is able to concretely communicate using the network, for example to call another UE.
Figure 2.4: Messaging sequence of a successful connection setup procedure in the reference model
Chapter 3

Functional Reactive Programming

In Section 3.1, the functional programming paradigm will be presented. Then, Section 3.2 will explain the need to use more reactive programming. After that, in Section 3.3, the FRP paradigm, which derives from the two previous ones, will be introduced, including its evolution, its advantages and its disadvantages. Section 3.4 will present other approaches for programming in a reactive way. Finally, Section 3.5 emphasizes how the FRP paradigm can be useful for the telecom servers.

3.1 Functional Programming

Functional programming is a programming paradigm which, as its name indicates, uses functions (either pre-defined or user-defined) as base elements. On the contrary to an imperative program, where a sequence of instructions tells the computer how to solve a problem, a functional program consists of an expression telling the computer what it should compute [38].

Functions in a functional language are thus similar to the mathematical concept of "function": they can be be passed around like every other objects, and even to other functions (in which case it is called function composition by analogy with mathematics). They also always give the same result when called with the same argument(s), without side effects: they are "pure". In addition to the latter, another important property of a functional language is that the values are immutable: they never change after instantiation, thus enabling an easier reasoning about the program. Those two properties contribute to making a functional program more reliable, predictable and maintainable [26]. Functional programming thus promotes code reuse and is often considered as an elegant way of writing programs. [38]

Examples of programming languages using the functional paradigm are Haskell, Scala, Common Lisp, Scheme, Erlang and OCaml. The majority of functional languages are almost purely functional (like Haskell or Common Lisp) but some of them allow the use of imperative constructs as well (like Scala, which also uses the object-oriented paradigm). Languages that are almost purely functional restrict the ways that a program can use to interact with other programs or systems.

Some other concepts that are widely used in functional programming are lists, recursion (which usually replaces loops), partial application of functions and strong type systems (enabling the use of pattern matching and type inference for example).
3.2 Reactive Programming

Nowadays most applications need to interact with their environment and to be able to react to events occurring around them. Indeed, there are new requirements on applications: higher number of servers needed, lower expected response time, different kinds of maintenance, and data size which is rapidly growing. Some of the most concerned domains are robotics and GUI applications.

The reasons why we need to strive for programming using more and more reactive functionalities and the definition of a reactive system have been summarized in the Reactive Manifesto [3]. It describes four characteristics that a program needs in order to be reactive. Thus, there is currently a need for systems that are:

- **event-driven** (react to events)
- **scalable** (react to load)
- **resilient** (react to failure)
- and **responsive** (react to users)

Anyone who wants to design a reactive system (and especially a reactive program) should keep those four interconnected aspects in mind.

Reactive programming is a quite new area of programming whose aim is to coordinate the asynchronous data streams that go between the different actors of the system. Those reactive behaviours (like changing the display according to what the user is typing) are difficult to program using the conventional ways (which favour a sequential approach of programming) because it is then impossible to predict the order of the arrival of events. Moreover, it is the responsibility of the programmer to check that all data dependent on one state is updated if this state changes and to ensure that the asynchronous callbacks (which traditional programming techniques rely on to react on events, like in the Observer design pattern [29]) are correctly implemented [17]. Doing so uses a lot of the programmer’s time and is often error-prone.

The reactive programming paradigm aims at tackling those issues in event-driven programs by providing abstractions to express the program in terms of its reactions to the events, thus freeing the programmer from having to manage the order of events and data dependencies, since the language takes care about that automatically [17]. This programming paradigm is based on the synchronous dataflow paradigm described by Lee and Messerschmitt in [37], but with less strict real-time constraints [17]. Indeed, synchronous languages are languages specifically developed for real-time systems, which forbid constructs like recursion in order to avoid space and time leaks. A space leak occurs when a program unexpectedly uses a very large amount of memory thus leading to a very long computation time (time leak), which makes the concerned program unsuitable for certain application domains, particularly in real-time systems. The key ingredients of the synchronous dataflow paradigm are **continuous time-varying values** and **propagation of changes**. The paradigm takes care that when a change occurs, all dependent values and computations are updated automatically as well. The abstractions provided thus enable the programmer to focus on what to do, as the language will take care of when to do it [17]. Two key abstractions of reactive programming are **behaviours** and **events**. Behaviours refer to continuously time-varying values whereas events refer to streams of value changes. A basic example of a behaviour is time itself (it has a value at every instant) and an example of an event can be keyboard button presses (which occur at discrete instants) [17].
3.3 Functional Reactive Programming

FRP is a quite new (around 20 years) area of programming whose aim is to combine the strengths of the functional approach with the need to use reactive programming. FRP uses the declarative paradigm [26]. It means that a programmer only needs to focus on what to program, leaving the compiler decide how the computer will do it. This enables a higher level of abstraction and makes the code clearer, because a lot of details do not need to be expressed any more in order for the program to work correctly. Moreover, FRP expresses the mutable values as time-varying values, which makes programming time-related values easier.

This programming approach has already been used with success to implement applications in various domains such as graphical user interface [26, 24], animation [28], video games [25, 22], robotics [61] and music synthesis [30]. All of these areas indeed need a lot of interaction with external systems, ranging from a human user to the mechanical components of a robot and are thus good representatives of reactive domains.

3.3.1 Evolution of the FRP paradigm

The name and the idea of FRP originated from the work of Conal Elliott and Paul Hudak [28] who worked on a more reactive animation framework, called Fran (for Functional Reactive ANimation). Their aim was to create higher-level abstractions for constructing interactive multimedia animations in a less complex way. They wanted to achieve a clear separation of modelling and presentation, so that the programmers only care about what the animation should do, with Fran handling how to present it.

Fran uses the programming language Haskell because they found that the animation modelling benefits from using properties such as higher-order functions, strong typing and non-strict semantics [28]. With Fran, Elliott and Hudak introduced the two key abstractions of FRP: the behaviour and the event. They defined the behaviours as "time-varying, reactive values" and the events as "sets of arbitrarily complex conditions, carrying possibly rich information" [28]. They also provided semantics for both abstractions.

However, Fran suffers from space and time leaks [61, 60]. Therefore, other frameworks were created in order to solve those issues, keeping the key abstractions of Fran that would become the key abstractions of the FRP paradigm. Because Fran was developed using Haskell, most of the FRP frameworks use Haskell as the support language. However, the paradigm has also been implemented using other languages like Java [23] or Scala [10] and even with languages dedicated to FRP, like Elm [26].

Until now, two main different semantic frameworks, which will be presented in the next subsections, have been defined theoretically through research on the FRP paradigm. They basically use many common concepts but differentiate themselves in the treatment of signals and the representation of events. [10]

Classic FRP

The original semantic framework used for FRP is now referred to as "Classic FRP" [27]. It stays close to the semantics of Fran, with some modifications in order to avoid space and time leaks. In Classic FRP signals (most often called behaviours) and events are first-class concepts. In FRP these are referred to as "first-class values" [17]. It means that the programmer can directly manipulate them, using the different language constructs available [10]. Behaviours
are continuously time-varying values. They are represented as functions from a time to a value \[20\]. Events represent a discrete sequence of values associated with a time-stamp.

As mentioned before, the semantics used in the Classic \texttt{FRP} frameworks were initially outlined for Fran \[28\]. This first formulation of \texttt{FRP} introduced the behaviour and the event, the two types of value that can be found in every \texttt{FRP} framework. These semantics were later formally defined by Wan and Hudak \[59\]. In this article, the authors gave a denotational semantics to \texttt{FRP} and tested the match of a stream-based implementation of \texttt{FRP} towards its formal semantics. The results were that the implementation is faithful to the semantics under a set of sufficient conditions, thus leaving to the programmer an important role in ensuring that the behaviours will respect the semantics.

The space and time leaks that Fran suffers from are due to the fact that Fran is implemented in Haskell \[26\]. Indeed, Haskell evaluates expressions in a lazy manner, meaning that it delays the computation until it is really needed. This kind of evaluation makes the use of infinite data structures very convenient. However, if an accumulated very big expression (taking a lot of memory and creating a space leak) suddenly needs to be evaluated, it can take a very long time (creating a time leak). Subsequent implementations of \texttt{FRP} thus focused on creating a leak-free implementation of \texttt{FRP}. Some implementations changed the language but kept the semantics, like FrTime (implemented in Racket) or Frappé (implemented in Java), whereas some other implementations made some changes in the approach to the semantics of \texttt{FRP}. Those latter implementations are presented in the next subsections.

**Classic FRP - second approach**

As described by Conal Elliott, one of the developers of Fran, despite being "simple and powerful", the first version of \texttt{FRP} semantics did not succeed in being efficiently implemented \[27\]. Therefore, Elliott presented a new version of the semantics \[27\] in 2009. These semantics combine the data- and demand-driven evaluations (hence the name of the article, Push-Pull \texttt{FRP}) to obtain both the advantages of pull-based evaluation (simplicity of functional implementation and applicability to temporal continuity) and those of the push-based approach (efficiency and minimal latency). This new version of the semantics solved the memory leaks problems of the previous one \[40\]. Frameworks like reactive-banana \[6\] and Lula-FRP \[53\] implement this modification of the Classic \texttt{FRP} semantics.

**Signal function FRP**

In Signal function \texttt{FRP}, signals are conceptually there but are not considered as first-class values. Instead, functions on signals can be manipulated and are made reactive. Events are considered as a special case of signals and can be manipulated with specific signal functions. \[10\]. Signal functions were introduced in the semantics of \texttt{FRP} with the aim to avoid space and time leaks and to simplify the use of external inputs as signals \[44\].

Real-time FRP (RT-FRP) introduced by Hudak et al. in 2001 \[60\], managed to solve the space and time leaks problems of the first version of Classic \texttt{FRP} semantics. In RT-FRP the semantics of \texttt{FRP} were also simplified, with behaviour and event being represented with a common type called signal. Indeed an event is considered as a behaviour of a special type that can either represent an occurrence of an event or \texttt{Nothing} (a Haskell constructor for an empty optional value). However, this implementation of \texttt{FRP} is less expressive because the signals are not higher-order functions and can only be manipulated by a limited language distinct from
the base language. This is done in order to have better control over signals and to ensure a resource-bounded execution \[26\].

The same research team also published in 2002 the Event-driven FRP (E-FRP) \[61\]. It is the continuation of the work with RT-FRP but introduces a discrete time model with discrete signals, that only change when an event occurs. Conal Elliott indicates in the second version of the Classic FRP semantics \[27\] that E-FRP has similar goals, with the system being re-evaluated only if necessary. However, he points out the lower expressiveness of E-FRP and some deviation from the original semantics.

These two approaches (RT-FRP and E-FRP), despite the fact that they have solved memory and space leaks, are not really developed any more. Indeed, the research teams which created them think that it is possible to achieve better expressiveness if going another way. However, a recent FRP implementation in Java uses this approach \[34\].

Arrowized FRP (AFRP) \[44\], created in 2002, aims at giving Haskell programmers some of the expressive capabilities of synchronous dataflow languages (such as Lustre, Esterel, Signal), basic hybrid modelling functionality and at solving the time and space leaks problems of the first version of the semantics of Classic FRP. Building upon the results of RT-FRP and E-FRP there are no events in AFRP. Instead, it uses signal functions, which are functions from a signal to another signal. A signal has the same definition as in RT-FRP \[26\]. In order to avoid the dreaded space and time leaks, the programmer cannot directly use signals. He or she has to manipulate them using signal functions. Thus, contrary to dataflow languages, the signal functions of AFRP (the equivalent to a dataflow processing element) are first-class values. AFRP is named after the arrow notation of Ross Paterson \[46\], which enables the programmer to conveniently use the arrow model, which increases expressiveness among other benefits. The AFRP has been implemented in frameworks like Yampa \[25\] or Fruit \[24\].

**Other evolutions**

Neil Sculthorpe and Henrik Nilsson \[51\] proposed a new conceptual model whose aim is to overcome some limitations of Yampa, such as a Haskell-based type system which is not safe or limited possibilities to enable optimisations like change propagation. It introduced a further abstraction called signal vector, with signal functions being redefined to be functions on signal vectors. In 2013, a new FRP framework using this model was designed by Edward Amsden: Timeflies \[12\]. It uses a push-pull evaluation system, like in the second version of the Classic FRP semantics. A part of its aim was to have better performance than Yampa and to provide a much easier way to implement optimizations than what is possible in Yampa.

Priority-based FRP (P-FRP) is a variant of E-FRP which guarantees that the resource usage is bounded and where different events can have different priorities. Belwal et al. \[19\] present a technology that reduces the energy consumption of programs using P-FRP, thus making it more suitable for embedded and real-time systems which are battery-powered.

Monadic FRP \[58\], which was presented in 2013, uses another approach to FRP based on the notion of reactive computation: "a monadic computation which may require the occurrence of external events to continue". It differentiates itself from other FRP formulations in two main aspects. First, the signal computations can end, thus providing a convenient interface for expressing reactive behaviours which are only needed for a limited period of time. Second, Monadic FRP can avoid needless recomputations by using a straightforward and purely functional implementation.
3.3.2 Advantages and disadvantages of FRP

The FRP has several assets, like a high level of abstraction, a more concise and elegant produced code, and a native handling of events (which means no need for the observer pattern any more). As it is derived from the functional paradigm, same pieces of code copy pasted at different places can be replaced by a function that will be passed around, thus promoting code reuse and removing the errors created when these same pieces of code are modified only in one place. The latter can also be done in OOP but as the functional paradigm is based on functions, it is easier and more flexible to do it using this paradigm.

However, considering the number of successive frameworks that were developed, it is rather difficult to go from the semantics to an efficient abstraction that will avoid memory and time problems. FRP has also known efficiency problems such as global delays and needless recomputations [26]. Global delays occur when a system cannot immediately process an incoming event and is thus less responsive. This is due to the fact that most of the FRP implementations maintain a strict ordering of incoming events. Events are handled one after the other and if one event takes particularly long to be processed, all the following have to wait, even if they are not related. A needless recomputation occurs when a function is recomputed even if its input has not changed. In purely functional languages, like Haskell or Elm, a function always produces the same output for a given input. Therefore, there is no need to recompute a function if its input has not changed. The Elm language, specifically created for FRP, was designed with those issues in mind, as well as the space and time leaks problems [26]. However, this language focuses mainly on GUIs.

One other current issue is that few functional reactive languages support multidirectionality and distributed programming [17]. Multidirectionality refers to the fact that when a value changes, the changes can be propagated to the dependent values (in one direction) but also back to the value from which they came from (in another direction). Distributed programming enables the programmer to spread dependent computations or data among several nodes (for example on different machines or on different cores of a multi-core processor). This area is currently being researched by Salvaneschi et al. [49].

3.4 Alternative approaches for reactive programming

Designing a language to express in an easier way reactive behaviours has not been the focus of research only in the FRP paradigm. Other approaches are also developed in order to make programs more concise, elegant and easier to read. These approaches which tackle the same kind of issues are presented in this section.

3.4.1 Akka

Akka [33] is an open-source toolkit and runtime developed since 2009. It aims to simplify the development of concurrent, distributed, fault-tolerant event-driven applications running on the Java Virtual Machine (JVM). It uses mainly the actor model to deal with concurrency, taking inspiration from Erlang. It is written in Scala and can be used in both Scala and Java.

The actor model is based on small, concurrent and independent entities called actors. Actors communicate with each other using messages and an actor can perform some basic actions on the messages it receives: sending messages, creating new actors or changing its internal state.
Those actions may be concurrent but the actor must be able to finish all its actions. Messages between the actors are sent asynchronously, without any preamble. Additionally, actors do not share data with each other. These characteristics makes systems using the actor model very scalable and easily distributed over several machines.

Akka implements this model with a few subtleties. Each actor has an event driven message box that keeps incoming messages before they are processed. Akka also implements ordering of the messages for some of them. However, since Akka runs above the JVM, a strict separation of data between actors is not completely achieved. In addition to the actor model, Akka also implements supervision, with each actor being supervised by its parent actor, which will handle problems that the actor itself can not solve. Akka has been used in numerous applications in various domains like Bitcoin exchange, photo manipulation or data analysis.

3.4.2 Erlang

Erlang is an open-source programming language and runtime system. Its development was initiated by Joe Armstrong in 1986 at the Ericsson Computer Science Laboratory. It aims to be used to build scalable, concurrent, distributed and fault-tolerant soft real-time systems that require high-availability.

This functional language also implements the actor model and focuses on enabling highly concurrent programs. In order to ensure high availability for the programs using it, Erlang is a safe language with a specific error philosophy: if an error occurs, only the affected process will fail and this failure will be taken care of by another process. From its first use to manage phone switches, Erlang has since been used for example in distributed Internet server applications or GUIs.

3.4.3 ReactiveML

ReactiveML is a functional programming language developed by Mandel et al. since 2005 at the Université Paris 6. It is specialized in the implementation of reactive systems and aims at combining the expressiveness of synchronous languages with the strengths of functional programming.

The ReactiveML language is built atop OCaml, a dialect of the ML language, with additional constructs to express the time-related part of the program. Its semantics includes a part influenced by the synchronous language ESTEREL and another one to manage the interactions between the reactive constructs and the classic ML constructs. ReactiveML is based on Boussinot’s reactive model, which combines the fundamental ideas of synchronous languages and features of asynchronous models, like dynamic creation of processes. ReactiveML was used for example in a simulation of power consumption in sensor networks or in a sequencer for mixed music.

3.5 FRP and the telecom servers

The telecom server programs consist of receiving messages, updating stored information if necessary and sending messages depending on the content or type of the message received previously. Those programs are very complex and could obviously benefit greatly from using a paradigm dedicated to reactive programs.
3.5. FRP AND THE TELECOM SERVERS

In this thesis, we will investigate one of the approaches to implement such programs, which is the FRP paradigm. As we have seen in this chapter, this paradigm aims at abstracting away all the pieces of code needed to handle events in traditional programming techniques. It claims to produce clearer programs, which are also easier to reason about. In a nutshell, it appears to be very promising for reducing the complexity of telecom servers applications.

However, the various problems (particularly those linked to memory and execution time) that have been spotted in various FRP implementations could prevent it for being concretely usable in this kind of real-life application. As far as we know, the FRP paradigm has never been used for this kind of large scale applications, being only used in research or basic programs.

In the following chapters, we will select a FRP framework that will be used to develop a prototype simulating a LTE base station. We will then compare its performance and maintainability with another prototype that uses traditional programming techniques.

---

1Erlang is not considered here because despite being functional, telecom oriented and having been used for large applications, it is not using the semantics of the FRP paradigm and thus falls out of the thesis scope.
Chapter 4

Survey of the FRP frameworks

In this chapter, the approach used to select one framework for the prototype development will be described and the outcome of the resulting survey will be presented.

4.1 Methodology

Having a good framework is important for the thesis work: it will make the comparison between the two paradigms more relevant. Therefore, an investigation of the existing FRP frameworks was performed in order to select one for the development of a LTE base station prototype. Indeed, more than twenty different FRP frameworks have been developed since FRP was created. However, they often target a specific application domain (like GUIs or music synthesis), which may not be adapted to a LTE base station prototype. Moreover, some of them are still experimental and cannot be used for real applications. The survey was conducted in two phases: first a preliminary selection and then a final selection.

For the preliminary selection we considered only the FRP frameworks which were mentioned in the research papers discussed in Section 3.3. Each framework was studied with respect to the following criteria:

- What language does the framework use? The different frameworks should indeed not use too many new languages in order to minimize the time spent learning them.

- Is the framework enough developed, stable and maintained? The framework should be usable to develop the prototype and it should be maintained, as a proof that it is not an abandoned project.

- Has the framework identified flaws? Frameworks with identified space and time leaks should be avoided in order to have good performance in the prototype.

- For what kind of applications has this framework already been used? If the application domain is similar to the thesis’ one, the framework is considered as a good candidate.

The complete list of the 21 frameworks that were investigated during this phase can be found in Appendix A.1.

At the end of the preliminary selection, four frameworks were selected. They are presented in the next section. In order to make the final choice, a more detailed list of criteria was
elaborated. It can be found in Appendix A.2 and also contains brief explanations of why the criteria are important. The different frameworks were studied more in detail so as to check these criteria. This included more literature and documentation studies but also some concrete experimentation in order to determine if they were easy enough to use and to understand in the limited time-frame of the thesis.

4.2 Presentation of the preliminary selected frameworks

The four selected frameworks (Scala.React, reactive-banana, Yampa and TimeFlies) are implemented in either Scala or Haskell. Haskell is a purely functional language, whereas Scala also uses concepts from the object-oriented paradigm.

In order to have comparable examples of code, each framework has been used to implement a very simple currency converter which takes a price in euros (EUR) and converts it to Swedish crowns (SEK). This example represents a typical reactive scenario: one value is updated by a user (here the price in EUR) and this automatically triggers the update of other related value(s) (here the price in SEK). Reacting to an external event and performing some actions related to it are the core ideas used in the connection setup procedure. The implementation process of examples like the currency converter was used to investigate criteria such as the ease of understanding the framework and the support provided with it.

4.2.1 Scala.React

Scala.React [39] is a FRP library for Scala, which has been developed since 2010 at the Ecole Polytechnique Fédérale de Lausanne by Ingo Maier and Martin Odersky (who is also the designer of the Scala language itself). The concepts of Scala.React and its implementation are presented in [40]. In this report, it is explained why and how the observer pattern could be replaced by reactive programming abstractions. Moreover, in [41], the authors present in detail one feature of Scala.React which improves the efficiency and the expressiveness of programs: a higher-order functional reactive data structure. However, they also mention that there is room for optimization of the current implementation of Scala.React, which they are working with. In Scala.React, the concept of behaviour is called signal. It is used to create dependencies between variables. The changes are propagated using a push-based model [17].

The currency converter example can be implemented in Scala.React as follows:

```scala
val valueEUR = Signal{ NewAmount() }
val valueSEK = Signal{ valueEUR() * 9.19} /*change value on the 29/07/2014*/
observe(valueEUR) {EUR =>
  //nicely print the value
}
observe(valueSEK) {SEK =>
  //nicely print the value
}
```

In order to keep the example simple, we must assume that the signal NewAmount holds at any time the amount of money to be converted. The function Signal is used to create the
depending signals `valueEUR` and `valueSEK`. The `observe` method contains a closure that is invoked at every change of the signal value. In the example, the value will be printed again each time there is a new amount of money to be converted.

### 4.2.2 Reactive-banana

Reactive-banana [6] is a FRP library for Haskell which is developed since 2011 by Heinrich Apfelmus. The concepts and implementation of reactive-banana are described in a series of articles on the maintainer’s blog [14]. Reactive-banana implements the second version of the Classic FRP semantics [27], with some adjustments. Since reactive-banana follows the Classic FRP semantics, it implements both behaviours and events.

The currency converter example can be implemented in reactive-banana as follows:

```haskell
snippet 4.2: Currency converter example implemented in reactive-banana

eValueEUR <- fromAddHandler newAmount

let
  eValueSEK :: Event t Double
  eValueSEK = (\euro -> euro * 9.19) <$> eValueEUR

reactimate $ printNicelyEUR <$> eValueEUR
reactimate $ printNicelySEK <$> eValueSEK
```

In this example, `newAmount` contains the updated value. This value is stored in the event `eValueEUR` using the reactive-banana function `fromAddHandler` and the Haskell operator `<-`, which means "draw from". The `<$>` operator is used to apply the function on its left to the value of the event on its right. The `$` operator is a Haskell operator used to avoid parenthesis. What is on its right takes precedence over what is on its left. Finally, the reactive-banana function `reactimate` is used to output results.

### 4.2.3 Yampa

Yampa [2] is a domain-specific embedded language in Haskell, which means that this language builds abstractions dedicated to a specific domain (like animation, GUIs or networking). It uses the concepts of FRP and is based on Fran. It is developed at Yale University by the Yale Haskell Group since 2003. This group presented the Yampa language and its underlying concepts in detail in [32]. This article also contains an example of the use of Yampa for a mobile robot. Robotics is not the only domain where Yampa has been used: in [25], the same research group described an implementation of the game "Space Invaders" using it.

In order to diminish the risks of introducing space and time leaks into the systems developed with it, Yampa is structured using arrows. Its abstractions for reactive behaviours are signal functions and events. Signal functions are a way of encapsulating time-varying values, but they are first class functions so the programmer can directly manipulate them [17]. Additionally, Yampa provides a set of primitive combinators for event composition and a set of switching

---

1For example, it does not implement the `switcher` function and the continuous behaviours

2[http://haskell.cs.yale.edu/](http://haskell.cs.yale.edu/)
combinators to enable the use of the dynamic dataflow structure. The changes are propagated using a pull-based model [17].

The example of the currency converter can be implemented in Yampa as follows:

```haskell
Snippet 4.3: Currency converter example implemented in Yampa

currencyConverter = proc -> do
  valueEUR <- newAmount
  valueSEK <- valueEUR * 9.19
returnA <- valueSEK
```

In the previous code snippet, `currencyConverter` is a signal function, defined using the keyword `proc`. `newAmount` is a predefined signal function holding the updated value. The converted value is sent as an input to `returnA` (the identity arrow, which plays the role of the `return` function in the arrow notation) using the arrow notation `->`. The converted value is returned each time the signal function is accessed.

### 4.2.4 TimeFlies

TimeFlies [11] is a FRP library for Haskell which was developed at the Rochester Institute of Technology by Edward Amsden during his master thesis in 2013. Thus, the concepts and the description how the framework was designed and implemented can be found in Edward Amsden’s thesis [12]. The three main goals behind TimeFlies were to implement a framework that has an efficient evaluation, a good composability of events and a simple integration with other Haskell libraries, like GUI libraries.

TimeFlies uses the signal function abstraction. It also uses the concept of signal vectors in order to be able to better control the inputs and outputs of the signal functions in a push-based evaluation. According to its designer, it provides a semantic guarantee that no event will be lost during evaluation. The language contains a whole set of combinators for manipulating the signal functions. TimeFlies uses a push-pull evaluation model to propagate changes [12].

The currency converter example can be implemented in TimeFlies as follows:

```haskell
Snippet 4.4: Currency converter example implemented in TimeFlies

currencyConverter :: SVEvent (Double) :~> SVEvent (IO())
currencyConverter = pureEventTransformer convertAndPrint
  where
    convertAndPrint :: Double -> IO()
    convertAndPrint valueEUR = do
      let valueSEK = 9.19 * valueEUR
      printNicely valueEUR
      printNicely valueSEK
```

In order to keep the example concise, the setup code for the TimeFlies framework is not considered here. It is thus assumed that the value to be converted is to be handled by the function `currencyConverter`. The function `pureEventTransformer` will then apply `convertAndPrint` to each new value of `newAmount` to obtain the result of the conversion.
4.3 Result of the survey

After the further investigation of the preceding four FRP frameworks, the final selection was made: it was decided to use reactive-banana for developing the LTE base station prototype.

Implementing small examples like the currency converter enabled a deeper comparison of the syntax used by all frameworks. For instance, the Yampa example is the shortest one whereas the Scala.React and the Timeflies examples are the longest ones. Secondly, all examples use specific syntax elements and reactive-banana is the framework where the most specific syntax elements were needed to write the example whereas Scala.React needed the least number of specific syntax elements. However, other aspects of the frameworks such as documentation, examples and help available were prioritized over syntactic sugar in the final choice of the framework to be used.

Indeed, the amount of documentation and help available for Scala.React was not sufficient to manage the development of the prototype in the time-frame of the thesis work. Two additional insights about Scala.React were also obtained during this survey. First, as Scala.React uses Scala, which mixes the object-oriented and the functional paradigm, it may not be the most suitable frameworks to emphasize the differences between FRP and traditional programming techniques like object-oriented programming. Secondly, even in the Scala.React enthusiast community, there are some interactions on whether the framework is still under development and whether the released version is ready to be used in real-world projects.

Choosing a framework using Haskell as its host language was also motivated by the fact that there were experienced people available at Ericsson, which could help if something was going wrong during the prototype development. At the time of the choice of the framework, Yampa had not been updated on Hackage since 2012. This is why TimeFlies was also investigated, as it was recent and closely related to Yampa (its aim was to improve Yampa’s performance and effectiveness). However, TimeFlies has not sufficient amount of examples and documentation available, probably because it is a very recent framework. Last but not least, it uses a recent evolution of FRP which implements signal vectors. The new syntax introduced with this evolution made the few examples available quite complicated to understand.

On the contrary to Yampa, reactive-banana is actively maintained and although its maintainer has not published academic papers about his framework, he follows the semantics of the new Classic FRP and discusses its implementation with Conal Elliott. Moreover, it is indicated that future development of reactive-banana will focus on performance, which is something important for the prototype. Finally, given the limited experience of the author in FRP or telecom programming, the choice among the frameworks tested was somewhat arbitrary and affected by the experienced ease and subjective judgement.

\[3\] But two updates were released later in 2014
Chapter 5

Prototype design and implementation

In this chapter, the approach that was used to define the reference model and to implement the prototypes will be described. Details about the reference model and the implementations will then be presented.

5.1 Methodology

In order to be able to determine whether the FRP paradigm leads to a less complex and more maintainable implementation of the connection setup procedure, the initial approach was to develop a prototype using FRP and to compare it to the code implementing the connection setup procedure in the real base station implementation. However, this approach had several downsides:

- Isolating the part of the implementation that would only deal with the connection setup procedure would have been extremely time-consuming and maybe impossible. The base station implementation is indeed very large and contains the code of numerous procedures that are sometimes interwoven with some others.

- The code extracted from the real implementation would not be executable without spending a lot of time creating stubs for all the needed functions. Therefore, comparing the performance of both implementation would imply relying on external tests that may be difficult to have access to and/or not have been carried in relevant conditions for this thesis.

- The prototype and the real implementation are not completely comparable programs. Indeed, they differ in some ways, such as:
  - the complexity of the procedure. The real implementation realises the complete procedure, including handling for various problems that can appear. Due to the limited time-frame allocated to the thesis work, it was not possible to implement a complete version of the connection setup procedure in the prototype: only a simplified version is implemented.
  - the conditions of execution. Procedures in the real base station include real radio-signalling, which is not the case of the prototype: it will only be implemented on a personal computer, for practical reasons. Therefore, comparing the performance of those two programs would be irrelevant.
how the code is written. The real implementation contains parts of code that are
automatically generated, which means that the programmer has no control over how
the paradigm is used in those parts. In the prototype, all the code will be written
by hand. Such a difference could affect the maintainability comparison of the two
pieces of code.

In order to perform a relevant comparison between the FRP paradigm and the traditional
paradigm that is used in the base station, both in terms of complexity and performance, it was
decided to improve this approach: instead of only creating one FRP prototype, two prototypes
implementing the same design but using different paradigms will be developed. Thus, the only
significant difference between the implementations will be the paradigm they use and not some-
thing else which cannot be controlled. The second prototype, which represents the traditional
programming techniques, will be implemented using the object-oriented programming language
C++, as it is the one used mostly in the real LTE base station implementation.

5.2 Reference model

The two prototypes will implement a simplified version of the connection setup procedure in
a LTE base station (see Figure 2.4). This simplified version is our reference model. The
simplifications made and the design decisions taken at the reference model level are presented
in the following subsections.

5.2.1 Simplified RRC connection setup procedure

The reactive behaviours which are investigated during this thesis are those typical of the telecom
servers (mentioned in Section 3.5). Therefore, the simplified RRC connection setup procedure
focuses on two aspects: receiving/sending messages and modifying stored information if needed.
Those two aspects are presented in the following subsections.

Messaging

The message sequence used in the reference model is almost the same as the one defined in the
RRC standard [8] and is depicted in Figure 2.4 for a successful connection. Apart from this
case of the success of the procedure, the reference model also contains three scenarios where
the connection setup procedure fails and the UE is not attached. A UE is rejected if:

- the eNodeB rejects its RRC connection request.
- the security mode fails to complete. This happens if the decryption of the security mode
  command message failed.
- the RRC connection reconfiguration fails to complete.

The sequence diagrams for those three scenarios can be found in Appendix B.

The content of the messages exchanged during all possible scenarios are most of the time
simplified, in order to only keep the pieces of information that are needed for modifying the
stored information or for sending the response. The other pieces of information, for example
those needed by other procedures or for configuring the radio signalling, were discarded. The
5.2. REFERENCE MODEL

The original full specification of the connection setup procedure messages can be found in Section 6.2.2 of the RRC standard [8].

Table 5.1 shows two examples of simplified messages with their fields. The first one is the RRC connection setup message, which is sent from the eNodeB to one UE. It contains three fields. The two first ones describe the Radio Network Temporary Identifier (RNTI) used to identify this UE in the eNodeB: its type and its value. Since the random access was completed, the type is Cell-Radio Network Temporary Identifier (C-RNTI). The third field contains the Signalling Radio Bearer (SRB) identity which identifies the specific bearer created by the eNodeB to communicate with this UE. When the UE receives this message, it stores the SRB identity in its context in order to use it for communicating with the eNodeB. Then, the UE creates the second message presented in Table 5.1, the RRC connection setup complete message. It has two fields: one containing the UE C-RNTI and one containing the UE public land mobile network identity, which enables correct billing for the service that the UE will use.

**Table 5.1: Content of two successive messages**

<table>
<thead>
<tr>
<th>Message</th>
<th>Field</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRC connection setup</td>
<td>UE id</td>
<td>Changes according to the progress of the procedure</td>
</tr>
<tr>
<td></td>
<td>RNTI type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UE id RNTI value</td>
<td>UE identifier in the eNodeB</td>
</tr>
<tr>
<td></td>
<td>SRB identity</td>
<td>Radio identifier</td>
</tr>
<tr>
<td>RRC connection setup complete</td>
<td>Public land mobile network identity</td>
<td>Mobile country code and mobile network code</td>
</tr>
<tr>
<td></td>
<td>UE C-RNTI</td>
<td>UE identifier in the eNodeB</td>
</tr>
</tbody>
</table>

**User Equipment context**

Each actor involved in the RRC connection setup procedure needs to store UE information in order to ensure a successful connection and later on, a successful communication with the network. In the reference model, these pieces of information are gathered in the UE context. The eNodeB and the MME store various data for every UE that attaches to the base station whereas the UE only stores data about itself.

At the beginning of the procedure, most of the fields on the eNodeB and MME sides contain default values. The accurate values will be stored when they are received in the corresponding message or created. Table 5.2 presents the UE data that is stored in the UE. The data stored in the MME and in the eNodeB can be found in Appendix C.

**Table 5.2: Data stored in each UE**

<table>
<thead>
<tr>
<th>UE context</th>
</tr>
</thead>
<tbody>
<tr>
<td>International Mobile Subscriber Identity (IMSI)</td>
</tr>
<tr>
<td>Security key (for the decryption)</td>
</tr>
<tr>
<td>SRB identity</td>
</tr>
</tbody>
</table>
5.2.2 System architecture

Figure 5.1 presents an overview of the system architecture for the reference model.

A prototype consists of three components, which implement the three types of actors involved in the connection setup procedure: the MME, the eNodeB and the UEs. There are always one MME and one eNodeB, whereas the number of UEs can vary. However, all the UEs are within the same UE component.

The messages exchanged between the three components are of two types: RRC messages (for communication between the UE and the eNodeB) or S1 messages (for communication between the eNodeB and the MME). The messages implement the different fields defined in the simplified RRC connection setup procedure. The various identifiers used in the different messages are either derived from the UE number (attributed when it is created) for the UE identifiers or generated randomly for the other identifiers.

A prototype also contains logging tools which register the content of each message arriving in one of the three components. Information about the number of messages received is registered in the eNodeB and MME logs. The log for the UE component indicates how many UEs were powered on. When a UE completes the procedure in one component (either with success or fail) a specific log entry is printed. Each log entry has a timestamp and there is one log file created for each of the three components.

5.2.3 Deployment details

The three components run in the same Linux computer and they communicate with each other using the Transmission Control Protocol (TCP) protocol (which is similar to the real-time signalling used in the real base station), as depicted in Figure 5.2. Since the programs are running in the same computer for practical reasons, another simplification was made: the prototypes are not handling packet loss in the network. Moreover, every component runs in one operating system thread, including the UE component (meaning that all the UEs are in the same thread).
5.3. Implementations

The complete source code of both prototypes is available online\footnote{https://github.com/Ilydocus}. In this Section, the particularities of each implementation will be presented.

5.3.1 FRP implementation

The FRP implementation of the LTE base station prototype is coded in Haskell using the reactive-banana framework. It is structured using the Haskell Cabal\footnote{http://www.haskell.org/cabal/}, which is a system enabling the use of a common architecture for building Haskell applications and libraries.

Program design

In order to implement the reference model using FRP, the three main components were split in smaller design modules (depicted as light blue boxes in Figure 5.3). The arrows indicate that a module uses function(s) from another module. Each design module is implemented using one or several Haskell modules (depicted as dark blue boxes in Figure 5.3). Using Haskell modules is the way used to structure programs in Haskell. Each Haskell module corresponds to one source code file with the same name (except from the main modules which must be named "main", whatever the name of the source code file).

---

5.2.4 Simulation parameters

In order to be as close to the real situation as possible, some RRC connection setup procedures must fail to complete, according to the scenarios described in 5.2.1. Therefore, two failure modes were implemented.

The first failure scenario occurs after the RRC connection request message. In the reference model, 3% of the UEs are rejected at this stage. The second failure scenario occurs if the security mode is not completed. It was chosen not to implement a failure mode for this scenario in the prototypes. The last failure scenario occurs if the RRC connection reconfiguration fails. In the reference model, 1% of the UEs which initiated the reconfiguration are rejected at this stage.

---

Figure 5.2: Deployment view of the reference model
CHAPTER 5. PROTOTYPE DESIGN AND IMPLEMENTATION

The design modules are organized as follows. First, there are three main modules which contain the main functions of each three components (i.e. all the FRP setup and the message handling logic). Then there are six other modules which contain functions that are used in at least two of the main modules. These functions belong to different areas. The functions that are found in the logging module (implemented in the Haskell module LogTools) handle all the logging-related actions of the prototype. Then, the FRP module contains setup functions used by all three components. The random identifiers generation module contains the functions used to generate randomly the different fields of the UE context. The context module contains the definition of the three different context, as well as the functions used to modify or print those contexts. Finally the messages modules contains the message definitions and the functions related to them. It is implemented using two Haskell modules, one for each type of messages (RRC or S1).

![Module coupling in the FRP prototype](image)

The following subsections present the interesting particularities of the FRP prototype implementation. This includes a description of how the handling of a message is performed in this prototype, focusing on the important parts for the telecom servers: receiving/sending messages and modifying stored information when needed. The same description will also be made for the OOP prototype in order to be able to compare both implementations.

Haskell light-weight threads

Even if each of the three programs runs in one operating system thread, some parts of the programs need to use concurrency, i.e. to execute different actions at the same time. For example, the UE program has to implement several UEs that will simultaneously try to attach to the base station.

In the FRP prototype, concurrency is achieved using Haskell light-weight threads. This kind of thread can be created using the function forkIO, which belongs to Concurrent Haskell. They are called light-weight because both thread creation and context switching overheads are very low. They are thus very cheap to use and more efficient than operating system threads.
5.3. IMPLEMENTATIONS

The scheduling of those threads is done internally by the Haskell runtime system. Haskell threads are non-deterministic, meaning that the order in which they will execute cannot be specified.

Snippet 5.1 shows how the different UEs are created and powered on simultaneously in the prototype:

```haskell
ues <- mapM (async . powerOn logh) [1..nbOfUes]
mapM_ wait ues
```

Here the function `async` is used. It is part of a Haskell module which is a thin layer over the basic concurrency operations providing functions for performing I/O operations asynchronously and waiting for their results.

Setting up the event network

The reactive-banana framework uses an event network to implement FRP. This event network contains the event logic of the program: what type of events the event network will handle and how to react to them. Implementing the event network means describing these events and reactions in a function in the source code. The event network can execute actions outside of itself using a specific instruction.

Snippet 5.2 shows the main function of the eNodeB program. It presents how the event network is set up. We can see that the setup of an FRP network (with the data structure needed to store the UE contexts and a log system) does not demand a lot of code: only 6 lines. Moreover, since FRP provides high-level abstractions for the setup, it is concise and the different parts are rather easy to understand.

```haskell
main :: IO ()
main = do
  sources <- makeSources
  enbMap <- newTVarIO Map.empty
  logh <- openFile "log_enb.txt" WriteMode
  network <- compile $ setupNetwork sources (return enbMap) logh
  actuate network
  eventLoop sources
```

The function `eventLoop` receives the messages sent to the eNodeB using the TCP protocol and fires the corresponding events to the event network in order for them to be handled.

Receiving a message and reacting to it

As explained in the previous paragraph, the messages are received in the `eventLoop` function and fired to the event network as soon as they are received. Snippet 5.3 shows how the messages coming from the UEs are received, decoded and then sent to the event network in the eNodeB. Thanks to the high-level abstractions of both Haskell and FRP, this takes only two lines of code: one for receiving and decoding the message and one for sending it to the event network. Those lines do not contain any byte operation. We can also notice that the FRP prototype
does not discriminate the different messages arriving from the UEs before sending them to the
event network. It does however separate the messages coming from the UEs and from the MME
(since they arrive on different sockets and have different types) but both will be handled in the
same event network.

Snippet 5.3: Message reception decoding and sending to the event network

\[
\text{messDecUe <- decode <$> recv ueSock 1024} \\
\text{(fire messageUe (messDecUe,ueSock,mmeSock))}
\]

Once the message arrives in the event network, it will be handled according to its type. In
order to understand how a message is handled in the FRP prototype, we take the example of
an incoming RRC Connection Request message.

Snippet 5.4 contains the parts of the event network function that are related to the handling
of this message. The handling of an event is composed of different steps:

1. The event is retrieved from the event source (line 4)
2. Depending on the type of message which arrived, a specific event is created, here
eRrcConnectionRequest (line 24)
3. When the eRrcConnectionRequest event is fired, the createRrcCS function is
executed (line 29). This function creates a new event containing the response message,
which is either RRC Connection Reject or RRC Connection Setup, depending on the
C-RNTI contained in the event eRrcConnectionRequest.
4. Different response messages are aggregated in the event (line 41) eResponseMessage,
including eResponseRrcCRequest created during the previous step.

Different outputs are also defined in the function describing the event network:

- the received message (contained in the event eMessageUe) is written to the log file (line
49).
- the IMSI identifier contained in the received message (once it is identified as an RRC
connection request) is added to the UE context corresponding to the sending UE (line
47).
- the response message (contained in the aggregated event eResponseMessage) is sent
to the UE (line 45).

Snippet 5.4: Handling of the RRC Connection Request message

\[
\text{setupNetwork :: forall t. Frameworks t} => \\
\text{(EventSource (RrcMessage,Socket,Socket), EventSource(S1ApMessage,}
\text{Socket,Socket))} \rightarrow \text{IO(TVar EnbMap)} \rightarrow \text{Handle} \rightarrow \text{Moment t ()}
\]

\[
\text{setupNetwork (messageUe, messageMme) state logh = do} \\
\text{eMessageUe <- fromAddHandler (addHandler messageUe)} \\
\text{(--(...) } \\
\text{let } \\
\text{--(...) } \\
\text{--Behavior holding the state}
\]
bUeContexts :: Behavior t (IO (TVar EnbMap))
bUeContexts = pure state

addImsi :: IO(TVar EnbMap) -> (RrcMessage,Socket,Socket) -> IO ()
addImsi state (message,_,_) = do
  liftedState <- liftIO state
  atomically $ modifyTVar liftedState (\ueMap -> Map.adjust (addImsi_enb (ueIdentity message)) (ueIdRntiValue message) ueMap)

--Incoming messages
incomingMessageTypeUe :: (RrcMessage,Socket,Socket) -> RrcMessageType
incomingMessageTypeUe event = case event of
  (RaPreamble _ _,_,_) -> RaP
  (RrcConnectionRequest _ _, _,_) -> RrcCRequest
  --(...) 23

  eRrcConnectionRequest = filterE (\t -> (incomingMessageTypeUe t) == RrcCRequest) eMessageUe
  --(...) 24

--Responses
eResponseRrcCRequest =
  createRrcCS <$> eRrcConnectionRequest
  where
    createRrcCS (message, ueSocket, _) =
      if reject
      then (RrcConnectionReject crnti ((crnti 'mod' 15) + 1),ueSocket)
      else (RrcConnectionSetup C_RNTI crnti srbId,ueSocket)
      where
        crnti = ueIdRntiValue message
        srbId = genRandId 8 (crnti *4)
        reject = (crnti 'mod' 30) == 0
      --(...) 38

  eResponseMessage = eResponseRaP 'union' eResponseRrcCRequest
  --(...) 42

--Output
reactimate $ (sendResponse logh <$> bUeContexts) <$> eResponseMessage
  --(...) 46
reactimate $ (addImsi <$> bUeContexts) <$> eRrcConnectionRequest
  --(...) 48
reactimate $ writeToLog logh . showMessageUe <$> eMessageUe
  --(...) 50

Snippet 5.4 thus presents a very interesting feature of FRP inside the event network, everything is an event (or a behaviour). It means that one or several action(s) can be triggered at every step of the handling of an event. For example, on line 24, the incoming event is recognized as a **RRC Connection Request** message and is stored in the variable eRrcConnectionRequest. This will trigger two different actions (and could have triggered more if needed): the update of the **UE** context (line 47) and the creation of the response message (line 29). The programmer
CHAPTER 5. PROTOTYPE DESIGN AND IMPLEMENTATION

does not have to take care of the order in which those actions will be performed: both will be performed when the event is fired. Other features from Haskell and FRP are also present in Snippet 5.3 pattern matching, event filter and event union. Pattern matching is found in several functional programming languages (including Haskell). On line 20 and 21, it enables a quick filtering of the messages. There is no need to have an additional attribute to discriminate which message it is, only the type of the message is enough. The filtering function is thus kept very brief and easy to read. Event filter is a FRP feature. On line 24, it enables a specific event to be fired only when a certain message is received. It is thus possible to create different sub events all deriving from the same main event. The reverse operation is event union and is used on line 41. This enables the responses to be sent using only one line for all similar messages. Finally, we can notice that creating a response message is very concise, as it takes only one line of code (line 33 for example).

Updating the User Equipment context

The eNodeB and the MME need to store the UE context of several UEs whereas each UE only stores its own information. The UE context is stored in a record which needs to be updated several times during the RRC connection setup procedure.

However, since having a variable that is mutable is not in the spirit of functional programming, it was needed to use a specific type of variable: a transactional variable (TVar). Those variables can be modified using specific primitives (such as modifyTVar). In order to avoid concurrency problems, all the modifications or accesses to the TVar variables are performed atomically.

Encoding

In order to be sent to the TCP network, the messages (which are values of custom algebraic data types) need to be encoded into byte strings. This is done in the FRP prototype by using the Put and Get monads. Each message is encoded by creating a byte string that contains two parts: first a number which indicates what type of message it is and then the different fields of the message. The details of the message encoding and decoding can be found in the modules RrcMessages and S1Messages, which were developed during the thesis but are not included in this report.

Encryption

The message Security Mode Command (see Figure 2.4) contains a field which is encrypted. The UE which receives this message needs to decrypt it and compare it to its reference value in order to imitate (on a smaller scale) the encryption process which takes place in the real base station implementation. In the FRP prototype, this encryption and decryption is performed using the Haskell package AES.

5.3.2 OOP implementation

The second prototype, which represents the traditional programming techniques, is implemented using the object-oriented programming paradigm and the language C++.

\[^{3}\text{http://hackage.haskell.org/package/AES}\]
5.3. IMPLEMENTATIONS

Program design

The class diagram presenting how the reference model is implemented in the OOP prototype can be found in Figure 5.4.

Figure 5.4: Class diagram of the OOP prototype

The main function for each component creates an event handler. The event handler powers up the UEs (in the UE program) and receives the different messages. The handling of those messages is the task of an appropriate UeContext. There is one UeContext per UE in each of the three components and it also contains the UE context for this UE.

The following subsections present the interesting particularities of the OOP prototype implementation. This includes a description of how the different steps of the handling of a message are performed in this prototype and a comparison with how the FRP prototype performs them.

Handling of events

There is no equivalent in C++ to the Haskell light-weight threads. Therefore, another approach was used to enable the concurrent treatment of different messages: the use of the epoll facility. This facility regroups three Linux kernel system calls which provide I/O event notifications. It uses a file descriptor (created using epoll_create) which registers interest for other file descriptors (registration is done using epoll_ctl). The epoll instance then waits (using epoll_wait) for an event to occur in one of the registered file descriptors. Snippet 5.5 shows an example of a set up of the epoll instance (from the file EnbEventHandler.cc). The epoll instance is first created (line 5), then the socket that is used to communicate with the MME is added to the file descriptor of the instance in order for it to receive a notification when a new message arrives on this socket (line 14). Finally, the epoll instance waits in an infinite loop until an event arrives in one of its registered file descriptors (line 22). The code which follows this line will handle the incoming event. In the prototype, the epoll function is used in edge-triggered mode on the receiving sockets, which means that epoll_wait will return only when a new message is detected on those sockets.

Snippet 5.5: Partial set up of the epoll instance

```cpp
void EnbEventHander::run () {
  epoll_event ev, events[MAX_EVENTS];
  int connSock, nfds, epollfd;
```
Snippets 5.5 illustrates well the differences between the FRP and the OOP prototypes when it comes to setting up the handling of events (which is a very important part in the telecom servers). First, the number of lines needed for the setup is higher in the OOP prototype. Then, the abstractions used in the OOP prototype are on a much lower level. This has several implications: first the programmer needs more time to understand how the setup works, then he or she also has to read up on the different options to be sure that the epoll instance is correctly configured and finally because of the previous aspects, it is easier to make mistakes, hence the different error handlers that are in the snippet. During the prototypes’ implementation, it was more time-consuming to setup the epoll instance than the FRP event network. However, the epoll instance allows more configuration options, as it is on a lower level of abstraction.

Receiving a message and reacting to it

As it was stated in the previous paragraph, when an event occurs on a registered file descriptor, the function epoll_wait returns. If this event corresponds to a message arriving from a UE which is currently performing the RRC connection setup procedure, the following code is executed:

```cpp
ssize_t bytesReceived;
char incomingDataBuffer[1000];
memset(incomingDataBuffer, 0, 1000);
bytesReceived = recv(events[n].data.fd, incomingDataBuffer, 1000, 0);
if (bytesReceived == 0) {std::cout << "host shut down." << std::endl;}
if (bytesReceived == -1) {std::cerr << "receive error!" << std::endl;}
if (bytesReceived != -1 && bytesReceived != 0) {
    incomingDataBuffer[bytesReceived] = '\0';
    GOOGLE_PROTOBUF_VERIFY_VERSION;
    std::string strMessage(incomingDataBuffer, bytesReceived);
    RrcMessage rrcMessage;
    rrcMessage.ParseFromString(strMessage);
```
5.3. IMPLEMENTATIONS

Snippet 5.6 implements one of the important reactive behaviours for the telecom servers: the reception of a message. The higher abstraction level of the FRP paradigm can also be seen when comparing this snippet to Snippet 5.3. Instead of the two elegant lines of the FRP prototype, the OOP prototype needs 15 lines which include very low level actions, such as on line 8, were the "end of string" character needs to be manually added to the received message.

At the end of Snippet 5.6, the message is received (line 4), decoded (line 12) and the UeContext object corresponding to the relevant UE has been found (line 14 and 15). In the handleUeMessage function (line 16), each message will be handle according to its type. We take the example of the reception and handling of a RRC Connection Request message.

Snippet 5.7 shows the part of handleUeMessage related to this message. This snippet shows the OOP equivalent to the pattern matching of the FRP prototype. Here the RRC message needs to carry a specific attribute to indicate which messages it carries. Then the real message needs to be retrieved in a variable in order to be able to call the right handler. This is more low level and less readable than the pattern matching performed in the FRP prototype.

Snippet 5.7: Extract of the handleUeMessage function

```cpp
void EnbEventHandler::handleUeMessage(RrcMessage rrcMessage, UeContextEnb *ueContext){
   //(...)
   switch (rrcMessage.messagetype){
      //(...)
      case RrcMessage_MessageType_TypeRrcCRequest :
         {RrcConnectionRequest rrcCRequest;
          rrcCRequest = rrcMessage.messagerrccrequest();
          ueContext->handleRrcConnectionRequest(rrcCRequest);
          break;
      }
      //(...)
      default:
         std::cerr << "Unexpected message type from Ue in eNodeB" << std::endl;
   }
}
```

The next part of the code related to the handling of this message is found in the file UeContextEnb.cc (see Snippet 5.8). It consists of the function handleRrcConnectionRequest, called on line 8 in Snippet 5.7.

Snippet 5.8: Function handleRrcConnectionRequest

```cpp
void UeContextEnb::handleRrcConnectionRequest(RrcConnectionRequest message){
   std::ostringstream messageLog;
   messageLog << "Message received from Ue: RrcConnectionRequest {Rnti Type: " << 
                message.ueidrntitype() << " Rnti Value: " << message.ueidrntivalue() << " UE identity is : " << 
                (message.ueidentity()).mcc() << "-" << 
                (message.ueidentity()).mnc() << "-" << (message.ueidentity()).msin() << "}" << 
```
std::endl;
mLog->writeToLog(messageLog.str());

mState->imsi = message.ueidentity();

RrcMessage rrcMessage;
bool reject;
reject = (message.ueidrntivalue() % 30) == 0;
if (reject) {
    RrcConnectionReject *rrcCReject = new RrcConnectionReject;
    rrcCReject->set_uecrnti(message.ueidrntivalue());
    rrcCReject->set_waitingtime((message.ueidrntivalue() % 15)+1);
    rrcMessage.set_messagetype(RrcMessage_MessageType_TypeRrcCReject);
    rrcMessage.set_allocated_messagerrccreject(rrcCReject);
    printState();
}
else {
    std::string * srbId = new std::string;
    genRandId(srbId, 8);
    mState->srbIdentity = *srbId;
    delete srbId;
    RrcConnectionSetup *rrcCS = new RrcConnectionSetup;
    rrcCS->set_ueidrntitype(C_RNTI);
    rrcCS->set_ueidrntivalue(message.ueidrntivalue());
    rrcCS->set_srbidentity(mState->srbIdentity);
    rrcMessage.set_messagetype(RrcMessage_MessageType_TypeRrcCS);
    rrcMessage.set_allocated_messagerrccs(rrcCS);
}
sendRrcMessage(mUeSocket,rrcMessage);

Snippet 5.8 shows the different steps of the handling of the RRC connection request:

1. The UE context is updated according to the C-RNTI of the received message (line 7).
2. Depending on whether the request is rejected or not, the response message is created (lines 13-18 or 27-33).
3. If the request is accepted, the SRB id is added to the UE context (line 24).
4. Finally, the response message is sent (line 36).

This snippet shows important differences between the prototypes on aspects related to the important reactive behaviours in the telecom servers. First creating the response message takes 5 or 6 lines (depending on the number of fields in the message) in the OOP prototype whereas it takes 1 line in the FRP prototype. Moreover, the logging instructions, the storing of the IMSI instructions, the creation of the response message and the sending of the response are in every message handler in the OOP prototype. In the FRP prototype, those parts are
more clearly separated and common code is only written once in the event network function. However, the storing of the IMSI is more straightforward in the OOP prototype and requires 1 line. In the ERPP prototype, it requires an additional function and 5 lines in total. Moreover, as storing a value in a data structure which can be modified is not really in the spirit of functional programming, it requires the use of the specific type of variable TVar which demands a bit of learning to handle.

**Updating the User Equipment context**

In the C++ implementation of the prototype, the UE context is contained in a data structure. The relevant UeContext object (UeContextUe, UeContextEnb or UeContextMme) which represents the UE has a pointer to this structure in its attributes. Updating the UE context is thus done by using this pointer.

**Encoding**

The message encoding in the C++ implementation is taken care of by the Google Protocol Buffers Application Programming Interface (API). The structure of all the messages is declared using a specific language in separate files. Then, these files are used by the API to generate C++ files which contain all the functions needed to encode/decode and use the messages.

**Encryption**

In the C++ prototype, the encryption and decryption of the Security Mode Command field are performed using the Crypto++ library.

---

4 https://developers.google.com/protocol-buffers/
5 http://www.cryptopp.com/
Chapter 6

Performance evaluation

In this chapter, the performance evaluation part of the thesis work will be presented. The aim of this part was to investigate the first aspect of the thesis’ hypothesis: Using Functional Reactive Programming as a programming model leads to a suitably performing program compared to the current implementation of the RRC protocol in an LTE base station.

6.1 Key performance indicators

The performance of both prototypes is going to be evaluated according to different key performance indicators. The indicators used were thus:

- time needed for the three components to attach X UEs (where X is a placeholder for a fixed number which can take different values)
- time needed for the UE component to attach for each UE
- memory footprint of each of the three programs
- CPU footprint of each of the three programs

Those indicators were selected because they are the ones which are of interest in the real base station implementation.

6.2 Methodology

In order to carry out the performance evaluation and to measure the different key performance indicators, several tests were done using both prototypes. The following sections present the tests which were performed and how they were performed.

6.2.1 Testing environment

All the tests presented in this chapter were performed on the same test machine, which is equivalent in terms of hardware and operating system to what is used in the real base station. This test machine is a Dell Inspiron 7537 laptop which is running Ubuntu 14.04 LTS 64-bit. This computer is equipped of an Intel Core i7-4500U Haswell CPU and of 8Gb of DDR3
6.3. RESULTS

The programs where compiled using the version 4.8.1 of the GNU Compiler Collection (GCC) (for the OOP prototype) and the version 7.6.3 of the Glasgow Haskell Compiler (GHC) (for the FRP prototype).

6.2.2 Testing procedure

Performing a test consists in executing the three programs [UE], eNodeB and MME] which compose each prototype. The CPU usage, the memory usage and other statistics are measured for each program using the Linux commands `top` and `pidstat`. In addition, for the FRP prototype programs, the Haskell runtime system is used to obtain additional information, like time spent in garbage collection. Time information for the execution of the UEs is obtained with the help of specific lines of the log files which are generated during the execution of the programs. Each test is executed automatically several times (five times for each of the ten fixed numbers of UEs in Scenario 1 and three times in Scenario 2 below) with the help of bash scripts.

6.2.3 Testing scenarios

Two different scenarios were thus investigated to evaluate the performance of both prototypes. They simulate different situations which the base station encounters during its use.

Scenario 1: A fixed number of UEs is powered on

In this scenario, the [UE] program is asked to power on a fixed number of UEs. Those UEs are powered on in a limited amount of time, resulting in a bursty behaviour. This scenario thus corresponds to a peak load test. Such a situation, where a large number of UEs will try to connect to the base station almost simultaneously corresponds for example to a train entering the base station area when moving or to the end of a movie, where all the users begin to use their smartphones again.

Scenario 2: A random number of UEs is powered on each second

However, most of the time, the base station is not experiencing a bursty behaviour. Therefore, it was also important to test the behaviour of both prototypes in a scenario which simulates a spread load. In this second scenario, the [UE] program was slightly modified to power on a random number of UEs every second, those random numbers being Poisson distributed.

6.3 Results

The two preceding scenarios were thus implemented and tested. The results of those tests are presented in this section.

In the prototypes, the total time to power on a given number of UEs is calculated from the beginning of the program execution until the moment where all UEs have completed the RRC connection setup procedure, either with success or not. However, when the average time for one UE to attach is presented, only the successful ones are taken into account (those which received the RRC connection accept message), as the rejected ones do not have to send and receive the same amount of messages. The proportion of rejected UEs were defined in the reference model (see Section 5.2.4).
CHAPTER 6. PERFORMANCE EVALUATION

6.3.1 Results for Scenario 1

In order to test the first scenario, both prototypes were executed ten times with an increasing number of UEs to power on (from 100 to 1000). The test was performed five times for each given number of UEs to power on. The values presented show the average and standard deviation of these five test runs.

**FRP prototype**

Figure 6.1 presents the average total time (with standard deviation) needed to power on a given number of UEs. It reminds the exponential curve since the average total time grows faster with the increase of the number of UEs powered on.

![Figure 6.1: Average total time (with standard deviation) elapsed on the UE side for powering on given numbers of UEs (FRP prototype)](image)

Figure 6.2 presents the average time for a UE to attach according to the total number of UEs which were powered on. From those three graphs, it can be observed that the average time for one UE to attach depends on the total number of UEs which were powered on. Indeed, when 100 UEs are powered on, the average time to attach one UE varies between 0.25 and 0.36 seconds, whereas for 1000 UEs powered on, it varies between less than 1 and more than 8 seconds.

On this figure, an interesting behaviour can also be observed: the time needed for attaching a UE first increases, then stays stable and finally decreases for an increasing number of UEs attached. When looking at the CPU usage of the programs, it follows the same pattern: first it increases, then it stays stable at a high level and finally it decreases towards the end of the total time needed to attach all the UEs. Therefore the following situation may happen: when the number of UEs to attach is small (at the very beginning and at the end), the program attaches them with good performance (low CPU use and fast attach). However when there are a lot of UEs asking for attach, the performance degrades, meaning more time to attach and higher CPU use. This might consequently create a queuing phenomenon which could explain the aspect of Figure 6.2. This behaviour needs further investigation but the high CPU usage...
may be caused by a part of the reactive-banana implementation which has been identified by its maintainer as decreasing the framework performance [13].

Moreover, a problem appeared during the testing: the productivity of the UE program was a lot lower than the productivity of the eNodeB and MME programs. The productivity, which is calculated by the Haskell runtime system, is the percentage of execution time spent doing productive work (i.e. not spent doing garbage collection). A low productivity therefore indicates that the program spends a lot of time in garbage collection, meaning that the program generates a lot of heap allocated data. A satisfactory productivity is above 90% [13].

Figure 6.3 shows clearly that the UE program has a much lower productivity than the eNodeB and the MME programs. The only noticeable difference between the UE program and the other ones is that several FRP event networks are concurrently created and used in the UE program whereas the eNodeB and the MME use only one FRP event network each. This suggests that creating a lot of FRP event networks using Haskell’s concurrency does not result in good performance.

Since the UE program was experiencing performance troubles which could affect the eNodeB program, it was decided to develop a second version of this program (with the same messaging sequence and actions on the messages) which would not use FRP but classic functional programming. It means that we don’t use an FRP event network anymore to handle the messaging. Instead, the messages are received and treated in a sequential order. By doing so, we lose the flexible event handling system of FRP but we hope to solve the performance issue. Scenario 1 was then conducted again using this second version of the program.

First, Figure 6.4 shows that the productivity of the UE program using classic functional programming is now comparable to the productivity of the eNodeB and MME programs using FRP. Then, the average total time elapsed for a given number of UEs to complete the RRC connection setup procedure is presented in Figure 6.5. Two interesting conclusions can be drawn from this graph:

- The average total time is around five times lower than when using the FRP UE program (see Figure 6.1).
CHAPTER 6. PERFORMANCE EVALUATION

Figure 6.3: Average productivity (with standard deviation) of the FRP prototype programs according to the number of UEs powered on

- The average total time is now increasing close to linearly with the number of powered on UEs. However, there are some outliers (it is particularly visible for 700, 900 and 1000 UEs) which are also the values with the highest standard deviation.

During those tests, another interesting behaviour occurred: when testing the prototype with 500 UEs or more, a minority of test runs (on average 1.5 with a standard deviation of 0.76) would not complete in a reasonable amount of time (indicating a potential deadlock situation) and are thus not included in the values presented in the graphs. This may explain the outliers observed on Figure 6.5. It also suggests that this configuration of the FRP prototype (with a classic functional UE program and a FRP eNodeB and MME programs) is not always reliable when more than 500 UEs simultaneously try to attach. This behaviour was not present when the FRP prototype was tested with the three FRP programs.

The graphs of Figure 6.6 confirm that having the UE program implemented using classic functional programming impacts positively the performance of the eNodeB program using FRP. Indeed, the average time needed for a UE to attach is reduced in comparison to the values observed in Figure 6.2.

Since the purpose of this thesis was to investigate the use of FRP in a LTE base station (and not in the UEs) and since the UE program using FRP is affecting the performance of the prototype, it was decided to only use the UE program using classic functional programming for the tests of the second scenario. Thus, when comparing the OOP prototype to the FRP one, it is the values of the tests using the classic functional UE program which are taken into account.

OOP prototype

On Figure 6.7 we can see the average time needed for a given number of UEs to complete the RRC connection setup procedure. First, we can observe that this average time is around ten times less than the time needed in the FRP prototype. Then, we can see that the increase
6.3. RESULTS

Figure 6.4: Average productivity (with standard deviation) of the FRP prototype programs (with classic functional UE program) according to the number of UEs powered on.

Figure 6.5: Average total time elapsed on the UE side for powering on given numbers of UEs (FRP prototype with classic functional UE program).
Figure 6.6: Average time (with standard deviation) elapsed on the UE side for attaching a UE in the FRP prototype (with classic functional UE program) for 100, 500 and 1000 UEs powered on can be separated in three phases: from 100 to 200 UEs powered on, from 300 to 600 and from 700 to 1000. Within those phases, the increase is close to be linear but in between the phases there is a bigger increase of the average time needed and an increase in the standard deviation. This may be due to the fact that different test runs give very different values for the total time elapsed when more than 200 UEs are involved. Since only five test runs were performed, outliers have a great impact on the average total time elapsed, which may create those bigger increases in between the phases. This is not satisfactory but running more test runs was left out due to limitation in time.

Figure 6.7: Average total time elapsed on the UE side for powering on given numbers of UEs (OOP prototype)

Figure 6.8 presents the average time for a UE to attach according to the total number of UEs which were powered on. Those graphs also show that the OOP prototype is about ten times faster than the FRP prototype. It is also noticeable that the average time for one UE
to attach increases linearly with the number of UEs, apart from the last ones powered on (this may be a consequence of the use of epoll). On the contrary, in the FRP prototype (Figure 6.6), the average time increases rather quickly for the first fourth of the powered on UEs (for 100, 500 and 1000 UEs powered on) and a lot more slowly (between around 3 to around 8 times more slowly depending on the number of UEs powered on) afterwards.

Figure 6.8: Average time (with standard deviation) elapsed on the UE side for attaching a UE in the OOP prototype for 100, 500 and 1000 UEs powered on

**Conclusion of Scenario 1**

The first scenario, which compared the behaviour of the two prototypes in a peak load situation, showed that under those conditions, the OOP prototype performs better than the FRP prototype, since the time needed for attaching the UEs is smaller.

6.3.2 Results for Scenario 2

The second scenario simulates a base station in a spread load situation with a mean arrival rate of 100 UEs per second. In order to test it, both prototypes were executed during a given duration (10 min). During that time, a random number of UEs (following a Poisson process of mean 10) were powered on every $\frac{1}{10}$ seconds to simulate the spread load.

**FRP prototype**

Figure 6.9 presents the average attach time for the 5000 first UEs which successfully attached to the base station. We consider 5000 UEs here as, even if Scenario 1 only considered up to 1000 UEs, it is interesting to notice that the average time for attaching a UE is rather similar if it is the 1000th or the 5000th to attach. In Scenario 1, there was already a big difference when testing with 100 and with 1000 UEs and this difference increased even more when testing with more than 1000 UEs (those tests are not presented in this report).

Figure 6.9 shows that in a spread load scenario, the average time for a UE to attach (0.046s with a standard deviation of 0.01) is up to 50 times lower than in a bursty scenario, if we take the results of the bursty scenario where 1000 UEs are powered on. It is also noticeable that this
average time for a UE to attach does not increase a lot with the total number of UEs attached, contrary to what happens in the peak load scenario. However, when looking at the rest of the UEs attached (those results are not presented in this report), it appears that the average time for attaching a UE does increase with time.

![Graph showing average time elapsed for attaching a UE in the FRP prototype](image)

Figure 6.9: Average time elapsed on the UE side for attaching a UE in the FRP prototype

Moreover, Figure 6.10 shows the evolution of the eNodeB program shares in terms of CPU time and available physical memory during the ten minutes of the tests. Thus, we can see that the share of available physical memory is increasing linearly over time. This is due to the fact that to each attached UE corresponds a stored UE context in the eNodeB, so the more UEs are attached, the more memory the program is using for storing the contexts. This figure also shows that the share of CPU time used by the eNodeB program is very high at the beginning (on average 51.06% with a standard deviation of 5.45 during the first minute), then decreases to about 30% after one minute (with a high standard deviation), decreases again at around 200s and then increases with up and downs. This behaviour is the result of a deadlock occurring in the prototype after a random time following the first minute. This deadlock is probably the result of an ill-formed message failing to be decoded in the eNodeB program. This shows that, although it was chosen not to implement comprehensive error-handling in the prototypes due to time limitation, this is necessary to do in a real implementation to avoid such deadlock situations. Moreover, some formal analysis of designs would be needed to exclude deadlocks, both for the FRP and for the OOP prototypes. Due to this deadlock situation, only the first minute of Figure 6.10 can be used to compare the CPU and memory use between the two prototypes.

**OOP prototype**

Figure 6.11 presents the average attach time for the 5000 first UEs which successfully attached to the base station. We can see that the average time needed for a UE to attach in the OOP prototype is a lot lower (around 8 times) than the one needed in the FRP prototype. The two outliers occur when only one UE is powered on during a tenth of second. In this case, a lot
6.3. RESULTS

Figure 6.10: Average percentage (with standard deviation) of CPU time (CPU) and available physical memory (MEM) used by the eNodeB program in the FRP prototype.

of time (around 0.035s) is spent between the reception of the S1APInitialUeMessage and the reception of the S1APInitialContextSetupRequest. Moreover, when looking at the rest of the test results (which are not presented in this report), the average time for attaching a UE does not increase with time, contrary to what happens for the FRP prototype.

Figure 6.11: Average time elapsed on the UE side for attaching a UE in the OOP prototype.

Figure 6.12 shows how the eNodeB program shares CPU time and available physical memory during the ten minutes of the tests. On this figure, we can observe that the average shares of CPU time (6.75% with a standard deviation of 0.35) and available physical memory needed by the OOP prototype are a lot lower (around 7.5 times lower for the share of CPU time and up to around 36 times lower for the share of memory after one minute) than the ones needed by
CHAPTER 6. PERFORMANCE EVALUATION

the FRP one.

Figure 6.12: Average percentage (with standard deviation) of CPU time (CPU) and available physical memory (MEM) used by the eNodeB program in the OOP prototype

Conclusion of Scenario 2

The second scenario, which compared the behaviour of the two prototypes in a spread load situation, showed that under those conditions as well, the OOP prototype performs better than the FRP prototype. Indeed it has a lower average time needed for a UE to attach, as well as lower shares of CPU use and memory use. Moreover, it is interesting to notice that the average time needed for a UE to attach is much lower for both prototypes in the spread load scenario than in the peak load situation.
Chapter 7

Maintainability evaluation

In this chapter, the maintainability evaluation part of the thesis work will be presented. The aim of this part was to investigate the second aspect of the thesis hypothesis: Using Functional Reactive Programming as a programming model leads to a more maintainable program compared to the current implementation of the RRC protocol in a LTE base station.

This maintainability evaluation has been carried out in three different parts. In the first part, we investigated the maintainability aspects of the current base station code base. The second part uses the output obtained in the first part for investigating maintainability aspects of the two developed prototypes. Finally, in the third part, we compute software metrics to complement the investigation of the two prototypes’ maintainability carried in the second part. These three parts are presented in the following sections.

7.1 Expert evaluation - Initial thoughts

The first part of the maintainability evaluation was carried out with the help of three experts from Ericsson. The aim of this part was to obtain information about the maintainability of the LTE base station current implementation. Those pieces of information were then used during the next two maintainability evaluations of the prototypes.

7.1.1 Methodology

The approach is composed of five steps:

1. Elaboration of the interview questionnaire.
2. Meeting with the experts to present the thesis work and the evaluation process.
3. Sending of a preliminary questionnaire so that the experts can think about the topics which are going to be taken up during the interviews.
4. One-hour interview with each of the three experts.
5. Analysis of the interviews.

In the following subsections, we will outline the structure of the elaborated questionnaire and give some background about our experts, as well as brief information about the interview setup.
Interview questionnaire

The interview questionnaire (which can be found in Appendix D) was designed as a support for the interviewer. It is structured in four big parts:

- Questions about the expert.
- Questions about maintainability in general.
- Questions about the maintenance of the base station implementation.
- Questions about more specific aspects of maintainability (such as understandability, simplicity, readability, etc.).

The aim of this questionnaire was to give directions to the experts during the interview and to gather their experience and thoughts about topics that were thought to be of interest. Indeed, the experts know which aspects of maintainability particularly need to be emphasized and which ones may be less important for the LTE base station implementation. However, this questionnaire was only a guideline, which means that the experts did not have to answer all the questions if they did not want to. Moreover, they were encouraged to talk about other maintainability aspects which are not present in the questionnaire. Table 7.1 presents some of the different types of questions which are present in the interview questionnaire.

Table 7.1: Types of question examples from the interview questionnaire

<table>
<thead>
<tr>
<th>Question type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open question</td>
<td>4. What is maintainability for you ?</td>
</tr>
<tr>
<td>Yes/no question</td>
<td>8a. Would you say that a clear separation of concerns is achieved in the implementation ? (protocol logic, state handling, event handling, concurrent execution)</td>
</tr>
<tr>
<td></td>
<td>□ No</td>
</tr>
<tr>
<td></td>
<td>□ Rather no</td>
</tr>
<tr>
<td></td>
<td>□ Rather yes</td>
</tr>
<tr>
<td></td>
<td>□ Yes</td>
</tr>
<tr>
<td></td>
<td>8b. Why ?</td>
</tr>
<tr>
<td>Evaluation of a situation</td>
<td>16a. Please evaluate the following task using a number between 1 and 10 (1 means it’s extremely difficult and 10 that it’s extremely easy):  You have to add a new message in the middle of the current message sequence.</td>
</tr>
<tr>
<td></td>
<td>16b. Check all the following words/sentences that would apply to the task :</td>
</tr>
<tr>
<td></td>
<td>□ time-consuming</td>
</tr>
<tr>
<td></td>
<td>□ quick</td>
</tr>
<tr>
<td></td>
<td>□ will probably introduce errors (error-prone)</td>
</tr>
<tr>
<td></td>
<td>□ won’t introduce errors</td>
</tr>
<tr>
<td></td>
<td>□ hard to locate where to modify the code</td>
</tr>
<tr>
<td></td>
<td>□ easy to locate where to modify the code</td>
</tr>
<tr>
<td></td>
<td>□ hard to test</td>
</tr>
<tr>
<td></td>
<td>□ easy to test</td>
</tr>
<tr>
<td></td>
<td>□ Other: __________</td>
</tr>
<tr>
<td></td>
<td>16c. Could you explain why ?</td>
</tr>
</tbody>
</table>
Expert panel

This part of the maintainability evaluation was conducted with the help of three experts from Ericsson. Those experts work in different areas related to the base station implementation such as software architecture, testing, future hardware/software and the implementation itself. They can thus provide initial thoughts coming from different areas which would benefit from a higher maintainability of the code. All three of them have been working with the LTE base station implementation for at least five years.

Interviews

Three interviews were conducted, one with each expert. Because the scheduled duration for those meeting was only of one hour, the experts received in advance a preliminary questionnaire (which can be found in Appendix D) so that they could think about the different topics before the actual interview.

During the interviews, all the questions from the interview questionnaire were asked by the interviewer, as well as additional questions related to what the expert was saying. The thoughts which were gathered during those interviews are presented in the next section.

7.1.2 Outcome

The interviews provided initial thoughts about several aspects of the base station implementation maintainability. First, the experts gave their definition of maintainability. Thus, a software product is maintainable according to them if:

- it is flexible: it is possible to add new functionalities to the code and to correct existing bugs without introducing unwanted behaviours and destroying the architecture.
- it can be very often modified according to new requirements (both on software and hardware).
- it is easy to determine how a function or a module works, what are its side effects, how it interacts with the other components and how it uses the hardware resources.
- it is portable: it can be run on different types of hardware.
- it is easy for a lot of persons to work on it at the same time.

The experts have confirmed that there is room for maintainability improvements in the LTE base station code base. Those improvements can either concern the implementation itself (e.g. reducing the size of the code or achieving a satisfactory number of modules) or the way of working with it (e.g. improving the use of refactoring). Moreover, when discussing the maintainability characteristics presented (understandability, expandability, testability, readability and simplicity) with the experts during the interviews, it appeared that those characteristics are important for the maintainability of the base station implementation. Finally, according to the experts, it is realistic to change the paradigm used in one part of the system. However, they also expressed some reservations: changing the paradigm may solve some problems but it will also probably create new ones. Moreover, the skills and wills of the persons working with the base station code must be considered.
Therefore, in order to be a good candidate for a change, the paradigm must cohabit well with the rest of the system and must provide strong benefits. According to the experts, such benefits could be:

- The paradigm makes it easy to separate different parts of the system.
- The paradigm provides high abstraction levels.
- The paradigm provides a good, quick and easy to use support for documentation (like documentation written in the code).
- The paradigm provides a good fault-handling mechanism.
- It is possible to see different abstraction levels of the program.
- It is easy to follow and understand the flow of the program.
- It is easy to extract meta information from the code.
- It is easy for several persons to work on the program at the same time.
- The resulting code is highly readable.
- The resulting program is portable.
- There is a good tool support for making changes and testing.
- The test environment is close to the code.

Those initial thoughts were used in the other parts of the maintainability evaluation which concerned the two prototypes, either for the analysis of the evaluation questionnaire or for investigating characteristics of the prototypes.

### 7.2 Expert evaluation - Assessment

The second part of the maintainability evaluation was also carried out with the help of experts from Ericsson. A group of software experts were asked to assess the maintainability of the two prototypes that had been developed. This assessment was performed using a questionnaire. The prototypes used for this assessment were the one used in Scenario 1 of the performance evaluation (with the FRP[UE program). It was chosen to include expert assessments in the maintainability evaluation because it is the most frequently employed strategy for assessing the maintainability of a software system [13].

#### 7.2.1 Methodology

The approach which was chosen to conduct the expert assessment of the two prototypes is composed of four steps:

1. Literature review to find guidelines for the questionnaire design.
2. Elaboration of the evaluation questionnaire and preparation of the evaluation material.
3. Selection of experts who have worked with software for at least twenty years.

4. Filling of the questionnaire by the experts.

5. Analysis of the questionnaire results.

In the following subsections, we will outline the structure of the questionnaire elaborated according to the guidelines from the literature and give some background about our experts, as well as brief information about the assessment setup.

Evaluation questionnaire

In the literature which was reviewed, two different ways of performing expert assessments were presented: guided or unguided \[13\]. According to the time and resources available for the thesis work, it was chosen to use a guided assessment strategy adapted from the one presented in the Air Force Operational Test and Evaluation Center (AFOTEC) pamphlet \[55\]. This pamphlet describes guidelines for evaluating software maintainability and provides standardized questionnaires to obtain the experts’ assessment of the software.

Thus, the evaluation questionnaire which was designed during the thesis is derived from the module source listing questions presented in the pamphlet. A subset of the questions was chosen, according to their relevance for the maintainability evaluation of the prototypes. Indeed, part of the questions presented in the pamphlet evaluate aspect independent from the programming paradigm (like the quality of the documentation), which are not interesting for our study. Additional questions about the experts and for directly comparing the two prototypes were also added.

The initial thoughts obtained in Section 7.1 were reused here for validating the choice of the assessed maintainability aspects and for the analysis of the results of the expert evaluation. A delay in the scheduling of the interviews did not enable a wider use of the initial thoughts.

The evaluation questionnaire is structured in four parts:

- Four questions about the expert.

- Twenty-six questions where the expert is asked to indicate to what extent he agrees with the given statements when assessing each of both prototypes. There are thirteen statements (the same for both prototypes) which are related to different maintainability characteristics (simplicity, modularity, expandability) or to maintainability as a whole. We refer to the investigation conducted by this part as the first evaluation task. The aim of this task is to have an assessment of the maintainability of each prototype, when they are taken separately.

- Five questions where the expert is asked to compare both prototypes and to determine which one is the best with regards to several maintainability characteristics (simplicity, understandability, modularity and expandability) and to maintainability as a whole. We refer to the investigation conducted by this part as the second evaluation task. The aim of this task is to have an assessment of the maintainability of each prototype, when they are compared to each other.

- Seven questions about the two prototypes in general, including a self-evaluation of the expert’s understanding of them.
CHAPTER 7. MAINTAINABILITY EVALUATION

One maintainability characteristic which was mentioned in Section 7.1 as an important one is however not present in this evaluation questionnaire: the testability. This is due to the fact that the experts being provided only the written source code of the prototypes, they would have only been able to evaluate partially the testability of the prototypes, since the testing environment was not provided.

During the first assessment task, the experts were asked to say to which extent they agree to a statement. In order to facilitate the analysis of the answers and following the AFOTEC guide, this agreement is given using a Likert scale which is presented in Table 7.2.

Table 7.2: Table explaining the meaning of the different scale items.

<table>
<thead>
<tr>
<th>RESPONSE</th>
<th>MEANING</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decidedly Disagree</td>
<td>This response indicates the program being evaluated is terrible with respect to the attribute addressed.</td>
</tr>
<tr>
<td>2</td>
<td>Moderately Disagree</td>
<td>This response indicates the program being evaluated is very poor with respect to the attribute addressed.</td>
</tr>
<tr>
<td>3</td>
<td>Somewhat Disagree</td>
<td>This response indicates the program being evaluated is unacceptable with respect to the attribute addressed.</td>
</tr>
<tr>
<td>4</td>
<td>Somewhat Agree</td>
<td>This response indicates the program being evaluated is barely acceptable with respect to the attribute addressed.</td>
</tr>
<tr>
<td>5</td>
<td>Moderately Agree</td>
<td>This response indicates the program being evaluated is very good with respect to the attribute addressed.</td>
</tr>
<tr>
<td>6</td>
<td>Decidedly Agree</td>
<td>This response indicates the program being evaluated is excellent with respect to the attribute addressed.</td>
</tr>
</tbody>
</table>

Since the experts completed the questionnaire on their own, definitions of some terms or explanations are provided after each statement, in order to limit the misunderstandings. Last but not least, the experts are asked to give as many comments as possible about all parts of their assessment. Those comments can provide useful information, such as helping to identify which characteristics of the paradigms are particularly relevant with regards to maintainability. The full evaluation questionnaire can be found in Appendix E.

Expert panel

The four experts who performed the maintainability assessments were not the same as the first group of experts who performed the evaluation of the current implementation (see Section 7.1.1). The experts for this part of the maintainability evaluation were software experts working at Ericsson in different areas not related to the base station implementation. All of them have been working with software for at least twenty years.
Assessment

The assessment was performed by the experts on their own. They were provided the written source code implementing the peak load scenario for both prototypes and the sequence diagrams representing the messaging sequences. The material was accessed through the GitHub repository. They were asked to study the two prototypes and to answer using an online version of the evaluation questionnaire. They were also given a contact address, if some problems should arise when filling in the questionnaire.

7.2.2 Outcome

Four software experts completed the maintainability evaluation questionnaire. Among them, two named the functional paradigm as their favourite programming paradigm and two named the object-oriented one. The expert group as a whole was thus not biased towards one programming paradigm.

Outcome of the first task

First, the agreement of the evaluators on the questions of the first task was calculated, using a formula presented in [47]. This article describes statistical analysis techniques which were used to validate the AFOTEC methodology.

The formula is:

\[ AG = \frac{1}{NE} \sum_{i=0}^{NS} \frac{NR_i}{2^i} \]

where \( AG \) is the agreement factor, \( NS \) the number of unit steps in the scoring scale (in our case 5), \( NR_i \) the number of responses which are \( i \) steps from the mode \(^1\) and \( NE \) the number of evaluators (in our case 4). This formula assigns a weight to each possible response value depending on how many evaluators chose it and how far it is from the mode. The closer a response value is to the mode, the higher weight it gets. The sum of the weights of all response values is then divided by the number of evaluators to obtain an agreement factor between 0 and 1. The calculation of the agreement factors shows that the average agreement factor is 0.62 (standard deviation 0.06) for the OOP prototype and of 0.68 (standard deviation 0.10) for the FRP prototype. This indicates that the experts have different opinions on some questions since it is below 0.8 (defined in [47] as the threshold for expert agreement). These scores can be explained by the fact that there were only four experts who answered the questionnaire (which means that the weight of outliers will be important) and also by the fact that the range of the given responses was indeed very large on some questions (up to four on a scale of five steps). The agreement measures for each questions can be found in Appendix F.2.

The mean and standard deviation were then calculated for each question of the first task. The results are presented in Table 7.4 for the OOP prototype and in Table 7.5 for the FRP prototype. The mean values are divided in three maintainability levels, each of them corresponding to a background colour. Those three levels are presented in Table 7.3 and are derived from the AFOTEC maintainability thresholds.

---

\(^1\) In a data set, the mode is the value which appears the highest number of times. For example, in the data set [1,2,2,2,5], the mode is 2.
### Table 7.3: Maintainability levels according to the score on a question

<table>
<thead>
<tr>
<th>Score range</th>
<th>Maintainability level</th>
<th>Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00 - 6.00</td>
<td>Good</td>
<td>Green</td>
</tr>
<tr>
<td>3.00 - 3.99</td>
<td>Passable</td>
<td>Orange</td>
</tr>
<tr>
<td>1.00 - 2.99</td>
<td>Poor</td>
<td>Red</td>
</tr>
</tbody>
</table>

### Table 7.4: Mean and standard deviation for the OOP prototype

<table>
<thead>
<tr>
<th>Question</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The program contains a reasonable number of executable statements with regards to its purpose</td>
<td>2.75</td>
<td>1.50</td>
</tr>
<tr>
<td>The program contains a manageable amount of branching</td>
<td>3.25</td>
<td>1.71</td>
</tr>
<tr>
<td>The program contains manageable levels of nesting</td>
<td>3.00</td>
<td>1.41</td>
</tr>
<tr>
<td>The program is understandable</td>
<td>3.50</td>
<td>1.29</td>
</tr>
<tr>
<td>The program clearly allow you to follow the flow of the implemented [RRC] Connection Setup procedure</td>
<td>4.00</td>
<td>1.41</td>
</tr>
<tr>
<td>Overall, the program is simple</td>
<td>4.25</td>
<td>1.26</td>
</tr>
<tr>
<td>The program clearly separates different concerns from each other in the code</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>The program exhibits loose coupling between its different parts</td>
<td>3.50</td>
<td>1.92</td>
</tr>
<tr>
<td>The program exhibits high cohesion within its different parts</td>
<td>3.75</td>
<td>1.50</td>
</tr>
<tr>
<td>Overall, the program is modular</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>The program clearly describes how the flow of the implemented [RRC] Connection Setup procedure is impacted by the handling of an event</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>Overall, the program is expandable</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>Overall, the program is maintainable</td>
<td>3.50</td>
<td>1.73</td>
</tr>
</tbody>
</table>
### Table 7.5: Mean and standard deviation for the FRP prototype

<table>
<thead>
<tr>
<th>Question</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The program contains a reasonable number of executable statements with regards to its purpose</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>The program contains a manageable amount of branching</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>The program contains manageable levels of nesting</td>
<td>4.25</td>
<td>0.96</td>
</tr>
<tr>
<td>The program is understandable</td>
<td>3.75</td>
<td>1.71</td>
</tr>
<tr>
<td>The program clearly allow you to follow the flow of the implemented RRC Connection Setup procedure</td>
<td>4.25</td>
<td>1.71</td>
</tr>
<tr>
<td>Overall, the program is simple</td>
<td>3.50</td>
<td>1.29</td>
</tr>
<tr>
<td>The program clearly separates different concerns from each other in the code</td>
<td>4.50</td>
<td>1.00</td>
</tr>
<tr>
<td>The program exhibits loose coupling between its different parts</td>
<td>4.50</td>
<td>0.58</td>
</tr>
<tr>
<td>The program exhibits high cohesion within its different parts</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>Overall, the program is modular</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>The program clearly describes how the flow of the implemented RRC Connection Setup procedure is impacted by the handling of an event</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>Overall, the program is expandable</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>Overall, the program is maintainable</td>
<td>4.75</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Figure 7.1 presents the average score on each question for both prototypes. We can see than on a whole, the FRP prototype has higher scores than the OOP prototype, with all the questions except two having a score corresponding to a good maintainability level (see Table 7.3). On the contrary, the OOP prototype has three questions with a good maintainability level and one with a poor maintainability level, the majority of the questions having a passable maintainability level.

The standard deviations presented in Tables 7.4 and 7.5, which are higher on average in the OOP prototype than in the FRP prototype confirm what the agreement measure indicated: the experts tend to give similar scores to the FRP prototype whereas they tend to disagree on the scores for the OOP prototype. Therefore, the OOP prototype scores, which are around the average, can be interpreted as the result of some experts giving it very high scores whereas others are giving it very low scores. This is confirmed by a study of the answers’ frequencies (which can be found in Appendix F.1). This higher diversity of results for the OOP prototype may be the results of all experts having very good experience of the OOP paradigm and being very picky and precise in their answers, which may not have been the case for the FRP prototype since some of the experts had little experience with the functional programming paradigm.

The scores for the question "Overall, the program is simple" are particularly interesting. In the OOP prototype, the question has a good score (4.25) and is part of the minority of questions having a good maintainability level. On the contrary, in the FRP prototype, it has a rather bad score (3.50) and is part of the minority of questions having a passable maintainability level. The standard deviation (1.291 in both cases) is among the highest in the FRP prototype and the lowest in the OOP prototype. According to the comments of the experts, it might be that the scores on this question are actually showing the ease of the experts with each prototype. This could explain contradictory results such as the OOP prototype having a high score on "Overall the program is simple" and a rather low one on "The program is understandable".
Outcome of the second task

The second task asked the experts to compare the two programs on the three maintainability characteristics which were investigated in the first task (simplicity, modularity and expandability), plus on understandability (which was part of the simplicity section in the first task) and on maintainability in a whole. Table 7.6 presents the questions asked during the second task.

<table>
<thead>
<tr>
<th>Number</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Indicate the programming paradigm that exhibits the simplest program</td>
</tr>
<tr>
<td>33</td>
<td>Indicate the programming paradigm that exhibits the most understandable program</td>
</tr>
<tr>
<td>34</td>
<td>Indicate the programming paradigm that exhibits the most modular program</td>
</tr>
<tr>
<td>35</td>
<td>Indicate the programming paradigm that exhibits the most expandable program</td>
</tr>
<tr>
<td>36</td>
<td>Indicate the programming paradigm that exhibits the most maintainable program</td>
</tr>
</tbody>
</table>

The first investigated maintainability characteristic was simplicity. In the evaluation questionnaire, it is indicated that simplicity can be decomposed into program size, control structure complexity, data structure complexity and straightforward coding. It is related to the understandability of the program [55]. According to Figure 7.2, which presents the frequencies of the answers, the experts assess the FRP program as the simplest one, which actually gives an contradictory result to the one obtained when comparing the answers given to question "Overall, the program is simple" during the first task. This makes the study performed on simplicity unconclusive.

![Figure 7.2: Frequencies of the answers to question 32](image)

The second investigated maintainability characteristic was understandability. As defined in the questionnaire, a part of the program is understandable if the expert is able to understand what it does [55]. The results shown in Figure 7.3 indicate that the experts assess both programs...
as equally understandable. However, as it was mentioned by two experts in the comments, this question may be biased towards the paradigm which the expert knows best.

The third investigated maintainability characteristic was modularity. As explained in the questionnaire, modular software is composed of largely independent parts \[55\]. The frequencies of the answers shown in Figure 7.3 indicate that according to the experts, the programs are equally modular.

![Figure 7.3: Frequencies of the answers to question 33](image)

The fourth investigated maintainability characteristic was expandability. A program is expandable, according to the definition provided in the questionnaire, when it is easy to make changes and to add new parts to it \[55\]. Figure 7.4 shows that the experts assessed the FRP

![Figure 7.4: Frequencies of the answers to question 34](image)
program as being the most expandable. However, one expert indicated that he found both prototypes equally expandable so choosing one over the other may not be relevant.

Finally, the expert were asked to choose the most maintainable paradigm. Figure 7.6 shows that both programs are assessed as equally maintainable. After an analysis of the comments, it appears that the understandability characteristic played a great role in the experts’ assessments on this question. Indeed it is also interesting to notice that three out of the four experts declared having a quite good or good understanding of the OOP program, whereas three out of four declared having a bad or quite bad understanding of the FRP prototype.
Threats to validity and improvements

A potential bias in this study is that some experts having a lot more experience with the object-oriented programming than with functional programming may tend to favour the first one because they have troubles understanding the FRP program. Such troubles were indeed mentioned by one expert. Two different improvements to the study could have reduced this source of bias. The first one would have been to provide the experts with proper education in the FRP paradigm to help them understand it. The second would have been to ask people without any specific knowledge of programming after educating them in both paradigms, as suggested by one expert. Both improvements were however not feasible within the thesis timeframe. Nonetheless, this situation actually reflects the real situation as described by the first group of experts in their initial thoughts: most people have experience and skills in object-oriented programming and only few have experience and skills in functional programming. The same understandability problems would probably arise in a real situation as well.

Moreover, this study would greatly benefit from having more experts assessing the two prototypes. This would increase the reliability of the results by reducing the weight of experts that are strongly biased towards one paradigm for any reason. This may also emphasize better the differences between the two paradigms.

7.3 Software Metrics

The third part of the maintainability evaluation consisted in using software metrics in order to compare the maintainability of the OOP and the FRP programs.

7.3.1 Methodology

The chosen approach to conduct this last part of the maintainability evaluation is composed of three steps:

1. Review of the literature to find potential metrics.
2. Choice of the metrics to be used.
3. Calculation of the chosen metrics.

Literature review

The aim of the literature review was to find potential metrics to use for the comparison of the two prototypes. The review was focused on software metrics which measure maintainability or characteristics related to it, such as complexity or readability for example.

Choice of the metrics

During the literature review, some interesting metrics were found such as McCabe’s cyclomatic complexity [43], Halstead’s measures [31] or Shao and Wang’s cognitive functional size [52]. However, calculating those measures is either time-consuming or requires specific software. Using those measures was thus not possible with the thesis time frame and resources.
Therefore, it was chosen to measure metrics related to the size of the code. Indeed, those metrics were mentioned in most of the studies, were found to be successful software maintainability predictors [48] and can be calculated easily for both prototypes.

Among the existing code size metrics, the following were measured:

- Number of lines of code (LOC)
- Number of lines of comments (CM)
- Total number of lines in the files
- Average line length
- Number of files

### 7.3.2 Outcome

The software metrics were calculated on the version of the prototypes which corresponds to the one evaluated by the experts. The programs considered in this part are thus the code used in Scenario 1 of the performance evaluation (with the FRP spirit).

The number of lines of code was measured using the tool Code Analyzer. The measures obtained are presented in Table 7.7 for the FRP prototype and Table 7.8 for the OOP prototype. In the latter table, a subtotal has been calculated to separate the files which have been manually coded and the files which have been generated by the Google protocol buffers API. The two .proto files used for this generation are presented separately at the end of the table.

<table>
<thead>
<tr>
<th>File name</th>
<th>Number of files</th>
<th>Total lines</th>
<th>Average line length</th>
<th>Total code lines</th>
<th>Total comment lines</th>
<th>Total white space lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>eNodeB.hs</td>
<td>297</td>
<td>42</td>
<td>251</td>
<td>7</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>ue.hs</td>
<td>191</td>
<td>36</td>
<td>161</td>
<td>4</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>mme.hs</td>
<td>104</td>
<td>36</td>
<td>81</td>
<td>4</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>EventSources.hs</td>
<td>14</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>UeContextStates.hs</td>
<td>194</td>
<td>28</td>
<td>160</td>
<td>10</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Identifiers.hs</td>
<td>46</td>
<td>20</td>
<td>35</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>RrcMessages.hs</td>
<td>146</td>
<td>30</td>
<td>132</td>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>S1Messages.hs</td>
<td>79</td>
<td>27</td>
<td>65</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LogTools.hs</td>
<td>59</td>
<td>31</td>
<td>51</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>9</strong></td>
<td><strong>1130</strong></td>
<td><strong>957</strong></td>
<td><strong>24</strong></td>
<td><strong>150</strong></td>
<td></td>
</tr>
</tbody>
</table>

These tables show that the total number of lines of code needed to compile the OOP prototype (10990) is a lot higher (more than 11 times more) than the one needed for the FRP prototype.

---

2 [http://www.codeanalyzer.teel.ws/](http://www.codeanalyzer.teel.ws/)
### Table 7.8: Code size metrics in the OOP implementation

<table>
<thead>
<tr>
<th>File name</th>
<th>Number of files</th>
<th>Total lines</th>
<th>Average line length</th>
<th>Total code lines</th>
<th>Total comment lines</th>
<th>Total white space lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>eNodeB.cc</td>
<td>11</td>
<td>9</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>EnbEventHandler.cc</td>
<td>283</td>
<td>31</td>
<td>248</td>
<td>2</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>EnbEventHandler.hh</td>
<td>27</td>
<td>21</td>
<td>22</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>UeContextEnb.cc</td>
<td>262</td>
<td>39</td>
<td>212</td>
<td>0</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>UeContextEnb.hh</td>
<td>47</td>
<td>26</td>
<td>39</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>ue.cc</td>
<td>21</td>
<td>16</td>
<td>13</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>UeEventHandler.cc</td>
<td>190</td>
<td>31</td>
<td>170</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>UeEventHandler.hh</td>
<td>28</td>
<td>19</td>
<td>23</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>UeContextUe.cc</td>
<td>217</td>
<td>40</td>
<td>175</td>
<td>1</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>UeContextUe.hh</td>
<td>44</td>
<td>25</td>
<td>37</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>mme.cc</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MmeEventHandler.cc</td>
<td>193</td>
<td>28</td>
<td>170</td>
<td>2</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>MmeEventHandler.hh</td>
<td>24</td>
<td>17</td>
<td>18</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>UeContextMme.cc</td>
<td>62</td>
<td>39</td>
<td>51</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>UeContextMme.hh</td>
<td>25</td>
<td>19</td>
<td>20</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>EventHandler.cc</td>
<td>26</td>
<td>15</td>
<td>21</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>EventHandler.hh</td>
<td>12</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>UeContext.cc</td>
<td>41</td>
<td>23</td>
<td>32</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>UeContext.hh</td>
<td>18</td>
<td>20</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Log.cc</td>
<td>37</td>
<td>23</td>
<td>30</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Log.hh</td>
<td>18</td>
<td>14</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>SUBTOTAL</strong></td>
<td><strong>21</strong></td>
<td><strong>1594</strong></td>
<td><strong>1332</strong></td>
<td><strong>5</strong></td>
<td><strong>257</strong></td>
<td></td>
</tr>
<tr>
<td>RrcMessages.pb.cc</td>
<td>5513</td>
<td>33</td>
<td>4662</td>
<td>210</td>
<td>641</td>
<td></td>
</tr>
<tr>
<td>RrcMessages.pb.h</td>
<td>3550</td>
<td>33</td>
<td>2887</td>
<td>213</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>S1Messages.pb.cc</td>
<td>1511</td>
<td>34</td>
<td>1269</td>
<td>62</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>S1Messages.pb.h</td>
<td>1030</td>
<td>34</td>
<td>840</td>
<td>61</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>25</strong></td>
<td><strong>13198</strong></td>
<td><strong>10990</strong></td>
<td><strong>551</strong></td>
<td><strong>1657</strong></td>
<td></td>
</tr>
<tr>
<td>RrcMessages.proto</td>
<td>125</td>
<td>22</td>
<td>106</td>
<td>0</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>S1Messages.proto</td>
<td>37</td>
<td>21</td>
<td>31</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>2</strong></td>
<td><strong>162</strong></td>
<td><strong>137</strong></td>
<td><strong>0</strong></td>
<td><strong>25</strong></td>
<td></td>
</tr>
</tbody>
</table>

Prototype (957). However, most of the additional lines (96%) in the OOP prototype are due to the files which were automatically generated by the Google protocol buffers API.

If we take into account only the files which were manually written (including the .proto files), the OOP prototype represents 1494 lines of code. It means that the OOP prototype is 1.6 times larger than the FRP prototype in terms of code which needs to be written. Considering that the prototypes are rather simple programs, it is a significant difference. Indeed, the more code
7.4. CONCLUSION OF THE MAINTAINABILITY EVALUATION

is to be written, the more probable it is for the programmer to make a mistake which will later need a maintenance operation: simplicity is an important maintainability characteristic.

However, in addition to the fact that changing or adding a message means having to recompile the .proto files (which might be time-consuming for large files: for the OOP prototype, it took 0.140s), the generated C++ files need to be understood in order to use the messages in the OOP prototype (for example to know how to fill a field of a message). In this case, a programmer who is not familiar with the implementation needs to browse through 11.5 times more lines of code to be able to work with it. This can lead to a huge difference in the time needed to perform a maintenance task since no programmer knows the complete implementation of the base station.

The average line length is similar between the two prototypes, with the OOP prototype having a slightly lower value. Small average line length is indeed better for the readability of the programs [21].

Moreover, there is also a big difference in the number of files in each prototype. With 23 manually written files in the OOP prototype and 9 in the FRP prototype, the OOP prototype contains 2.6 times more files than the FRP, which also contributes to make the source code more complex.

Last but not least, when focusing on the parts of the code dealing with the events, differences between the prototypes are also present. For example, the code parts which handle the RRC connection request consist of 30 lines in the OOP prototype (see Snippet 5.8) and 16 lines in the FRP one (see Snippet 5.4). Moreover, in the FRP prototype, 5 of the 16 lines are actually used by several messages, thus showing that the FRP paradigm more easily enables to reuse code, which is a good feature for maintainability.

Another example is receiving a UE message, decoding it and calling the function which handles it. It corresponds to 12 lines in the OOP prototype (see Snippet 5.6) and to 2 lines in the FRP one (see Snippet 5.3). This is because the FRP prototype, in addition to having higher abstractions for event-handling with the FRP framework, benefits from the high abstraction level of Haskell as a whole, here with the abstractions dealing with the reception and decoding of a message arriving in the TCP network. Being able to have higher abstractions was part of the experts’ wishes (see Section 7.1).

7.4 Conclusion of the maintainability evaluation

The maintainability evaluation of the two prototypes consisted in a combination of experts’ assessments and of structural measures. During the literature review, it was found that such a combination has been proven to be the most efficient way to assess the maintainability of software [13].

The first part of the maintainability evaluation was an investigation of how three experts working with the real implementation of the base station understood its maintainability. The obtained initial thoughts show, for example, that high level abstractions are desirable. Moreover, the initial thoughts indicated that the maintainability of a FRP base station would be affected by the persons working with it, since FRP is not currently very well-known.

3This was one of the initial thoughts of Section 7.1.
The important aspects discovered during this first part were then studied through an expert-based assessment. During this assessment, the experts evaluated the FRP prototype better than the OOP prototype in 11 of the 13 questions composing the first evaluation task. Moreover, when asked to compare the two prototypes, the FRP was assessed better or equal to the OOP prototype on all criteria. According to the participating experts, the FRP prototype is thus the most maintainable. The evaluation of the two prototypes also showed that the FRP prototype provides higher level abstractions compared to the OOP one, which was one of the desirable characteristics mentioned in the initial thoughts. The thought that the maintainability of a FRP base station would be affected by the persons working with it was also brought up by the experts performing the assessment of the two prototypes.

Finally, structural measures were investigated. The measures used were related to code size. They show that the FRP implementation is smaller than the OOP one with regards to number of code lines and number of files. Moreover, the number of code lines needed to implement event-handling is at least reduced by two when using the FRP framework. Finally, the two prototypes perform comparably on readability (this is confirmed by the experts’ assessment of question "The program clearly allow you to follow the flow of the implemented RRC Connection Setup procedure"). According to the chosen software metrics, the FRP prototype is thus the most maintainable.

Therefore, the maintainability evaluation of the two prototypes shows that the FRP prototype is more maintainable than the OOP one.

However, the initial thoughts from the real implementation also mentioned areas that were not investigated in the evaluation of the prototypes due to practical and time reasons. The first one is the testability of the programs and the second one is the scale of the programs, the real base station being a huge program whereas the prototypes are rather small ones. However, it was out of the scope of the thesis to investigate if all of the desirable characteristics could be provided by the FRP paradigm.

\footnote{The study on simplicity being un conclusive because of contradictory results between the two assessments.}
Chapter 8

Conclusions and future work

This last chapter presents the conclusions of the thesis work, further investigations which could be conducted as well as personal reflections on the thesis work.

8.1 Conclusions

We can now answer the thesis hypothesis which was formulated in Section 1.2 using FRP as a programming model leads to a more maintainable but not suitably performing program compared to the current OOP paradigm.

The goals defined for the thesis work are fulfilled. First, a theoretical investigation of FRP including a survey of the existing frameworks, was performed. Then, two prototypes implementing the same reference model were developed, one using the FRP paradigm and one using the OOP paradigm. Finally, those two prototypes were compared regarding performance and maintainability. The performance evaluation consisted of two scenarios: one representing a peak load situation and one representing a spread load situation. The maintainability evaluation used experts’ initial thoughts, experts’ assessment and software metrics.

At the end of the performance and maintainability evaluations presented in the two preceding chapters, the FRP prototype is assessed as more maintainable than the OOP one but the OOP prototype performs better than the FRP prototype in terms of average time needed for a UE to attach, CPU usage and memory usage. It was also shown that both prototypes perform better in a spread load situation than in a peak load situation, both in terms of time needed to attach a UE and maximum number of UEs which can be handled by the prototype. However, even if the performance of the prototypes cannot be directly compared to the performance of the real base station, some input about the performance of the real base station will be used to determine whether the prototypes perform suitably. Indeed, since the prototypes implement a simplified version of the connection setup procedure in a simplified base station, they are expected to perform better than the real implementation on certain aspects like execution time and number of UEs handled.

The performances expected in the real base station are the following: a UE should attach in less than 100ms, with slightly less than 10% of this time (meaning less than 10 ms) dedicated to the control plane procedures such as the RRC connection setup procedure (the rest of the time being used for radio and baseband signalling). The FRP prototype performances, as evaluated during the two scenarios presented in Chapter 6, are thus not suitable for an implementation
in the real base station, as the FRP prototype, which only implements the RRC connection setup procedure, needs more than 10 ms to perform a simplified attach of a UE.

Nevertheless, according to the results of the performance evaluation, the FRP prototype using reactive-banana did not have (for the investigated number of UEs) the dreaded space and time leaks (mentioned in Section 3.3.1) which were a big problem in preceding FRP implementations. Yet, having a lot of FRP networks running in the same operating system thread (as in the UE program) decreases the performance of the prototype. Nonetheless, the FRP prototype using the classic functional UE program, although faster than the one using only reactive-banana, sometimes has a time leak when more than 500 UEs were involved.

However, some aspects of this thesis work may weaken those conclusions. First, the maintainability evaluation was conducted with a limited number of experts, meaning that the results are influenced by the opinion of those specific experts. Moreover, the experts were not given training, particularly regarding FRP. This may have affected their assessment. Finally, the prototypes were developed with basic knowledge and experience of both OOP and FRP. Therefore they may not be representative of what experts in those paradigms can achieve and this may affect both the performance and the maintainability of the prototypes.

Finally we conclude with a reflection about the problem. In this thesis, we have investigated how a new programming model, FRP, could lead to a less complex code and thus an increased maintainability for the telecom servers. However, the thesis work showed that maintainability of the telecom servers encompasses other areas than the ones strictly related to the code. For example the workflow used with the programming model is also very important. However, the study of these other areas was outside the scope of this thesis.

8.2 Future investigations

This section presents ideas for further investigations of the performance or maintainability evaluation of the FRP paradigm, but which did not fit either in the time frame or in the scope of this thesis.

8.2.1 About performance

As it has been already mentioned, the prototypes are different from the real implementation on several aspects. The prototypes are indeed a simplified version of the real implementation and interact with simulated UEs and MME. In order to know the performance of the reactive-banana framework in real conditions and to test how easy or hard it is to use with the existing software, it would be interesting to implement the RRC connection setup procedure completely and to integrate it with the real base station software. Moreover, it would then be possible to check if time leaks sometimes occur in real conditions. Finally, performing a formal analysis could find the potential deadlocks (or prove their absence).

Then, Heinrich Apfelmus, the developer and maintainer of reactive-banana, announced on his blog [14] that he will focus on improving the efficiency of the framework for the next releases of reactive-banana. The performance discrepancies between the FRP prototype and the OOP prototype may thus diminish or even disappear with a future release of the FRP framework, so it may be interesting to perform the tests again with an improved version of the FRP framework.
8.3 PERSONAL REFLECTION ON THE THESIS WORK

8.2.2 About maintainability

The maintainability evaluation could be enlarged with the investigation of the testability characteristic, which was mentioned as important in the experts’ initial thoughts. Moreover, since the FRP prototype may need to use different tools than the one used currently with the implementation of the base station, it could be interesting to investigate the tool support that the FRP paradigm offers. In addition, the FRP paradigm may provide support in aspects that were mentioned as desirable by the experts, such as offering a way to extract high-level views of the software, so it could be interesting to investigate the other desirable aspects mentioned by the experts such as testability.

The experts also emphasized that changing the paradigm which is used for programming implies a lot of changes, ranging a new language to change of tools. A further study of all the impacts of changing the paradigm would thus have to be carried carefully before actually changing the paradigm.

8.2.3 About the FRP paradigm in general

Finally, it would be interesting to investigate further other FRP paradigms. Indeed, reactive-banana was chosen because it fitted the requirements for this thesis when the survey of the frameworks was made. However, a lot of research and development is currently going on around the FRP paradigm and there may be an alternative framework in the future which would be more suitable for the base station implementation.

8.2.4 About the prototypes

As mentioned before, the prototypes developed during this thesis are lacking comprehensive error-handling. Adding this would enable the prototypes to be closer to what is expected from a real implementation. It would also enable to compare the amount of code needed for this purpose in both paradigms.

Moreover, it could be interesting to ask an expert in each of the paradigms to implement the reference model and compare the performance and maintainability of the resulting programs with the ones developed in this thesis to know if the experience of the programmer has an impact and if yes, the extent of it.

8.3 Personal reflection on the thesis work

First I would like to examine whether my own background as a Computer Science student could have influenced this thesis work. I have encountered both the object-oriented and the functional programming paradigms during my studies, so I was already familiar with both those approaches before the thesis work. I am also comfortable with both paradigms. Therefore, I think that the thesis work was not influenced by a personal preference for one paradigm over the other. Regarding the languages, I had some university experience of programming with C++ whereas Haskell and the reactive-banana framework were completely new to me. However, I had the prototypes reviewed by someone experienced in both languages to avoid any beginner mistake in the prototypes which could compromise the study. Therefore, I think that my own background had a limited influence on the conclusions of this thesis.
Then, I would like to reflect on the thesis process. There were a lot of interesting investigations possible during this thesis but it was not possible to carry them out within the thesis timeframe. Therefore having a clear scope for the work from the beginning was very important in order to prioritize the work to do. Moreover, I have improved my prioritizing skills along the thesis and I think that if I had to do a similar work again, I would allocate less time for the literature review and for the framework selection, knowing what time is actually needed for implementing and evaluating the prototypes (which I had a bit underestimated). I would also define a clear scope for each phase of the work at the beginning of them in order to be able to refer to it when having to decide whether to investigate a new idea or not, and thus avoid investigating too many "nice but a bit unrelated" areas and have more time to dig deeper in the areas which are the most relevant for the thesis.
References


REFERENCES


[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns - Elements of reusable Object-Oriented Software. Addison-Wesley, 1995.
REFERENCES


REFERENCES


Appendix A

Survey of the frameworks

Table A.1: Framework list summarizing the preliminary selection outcome

<table>
<thead>
<tr>
<th>Name of the framework</th>
<th>Host language</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataDriven</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Dire</td>
<td>Scala</td>
<td>Rejected</td>
</tr>
<tr>
<td>Elerea</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Elm</td>
<td>Elm</td>
<td>Rejected</td>
</tr>
<tr>
<td>Euphoria</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Flapjax</td>
<td>JavaScript</td>
<td>Rejected</td>
</tr>
<tr>
<td>Fran</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Frappé</td>
<td>Java</td>
<td>Rejected</td>
</tr>
<tr>
<td>Frenetic</td>
<td>OCaml</td>
<td>Rejected</td>
</tr>
<tr>
<td>F roc</td>
<td>OCaml</td>
<td>Rejected</td>
</tr>
<tr>
<td>FrTime</td>
<td>Racket</td>
<td>Rejected</td>
</tr>
<tr>
<td>Grapefruit</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Netwire</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Ordrea</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Pyretic</td>
<td>Python</td>
<td>Rejected</td>
</tr>
<tr>
<td>Reactive</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>Reactive-banana</td>
<td>Haskell</td>
<td>Selected</td>
</tr>
<tr>
<td>Scala.React</td>
<td>Scala</td>
<td>Selected</td>
</tr>
<tr>
<td>Sodium</td>
<td>Haskell</td>
<td>Rejected</td>
</tr>
<tr>
<td>TimeFlies</td>
<td>Haskell</td>
<td>Selected</td>
</tr>
<tr>
<td>Yampa</td>
<td>Haskell</td>
<td>Selected</td>
</tr>
</tbody>
</table>
Table A.2: Characteristics needed for the framework to be chosen

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle several UEs simultaneously</td>
<td>The framework must be able to deal with several UEs sending the same type messages and not mixing them.</td>
</tr>
<tr>
<td>Respect the event order</td>
<td>The connection setup procedure should be executed according to a well-defined sequence of messages. The events representing the messages arrivals must be handled in the same order as their arrival order.</td>
</tr>
<tr>
<td>Use discrete events</td>
<td>The events are going to represent the messages arrivals and as such, it must be possible to know the arrival time.</td>
</tr>
<tr>
<td>Support</td>
<td>In order to efficiently learn it, the framework should have sufficient documentation (including examples) and if possible a responsive community.</td>
</tr>
<tr>
<td>Performance</td>
<td>The performance of the framework must not be hindered by known issues, such as space or time leaks.</td>
</tr>
<tr>
<td>Status</td>
<td>The framework must be in a stable state and it is preferable if it is actively maintained.</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>Since there is not an unlimited amount of time for understanding the framework, it should be rather easy to understand.</td>
</tr>
<tr>
<td>Impact of change is low</td>
<td>If possible, the framework should enable the programmer to make changes easily (for example to add a new message without having to refactor everything.)</td>
</tr>
<tr>
<td>High cohesion</td>
<td>If possible, the framework should have a high cohesion.</td>
</tr>
</tbody>
</table>
Figure B.1: Messaging sequence during a failed connection setup procedure: rejection of the RRC connection request
Figure B.2: Messaging sequence during a failed connection setup procedure: failure of the security mode

Figure B.3: Messaging sequence during a failed connection setup procedure: failure of the RRC connection reconfiguration
Appendix C

Content of the UE context

Table C.1: Data stored for each UE in the MME

<table>
<thead>
<tr>
<th>UE context</th>
<th>MME</th>
<th>UE</th>
<th>S1 AP id</th>
<th>Security key</th>
</tr>
</thead>
</table>

Table C.2: Data stored for each UE in the eNodeB

<table>
<thead>
<tr>
<th>UE context</th>
<th>RRC state (idle or connected)</th>
<th>C-RNTI</th>
<th>IMSI</th>
<th>SRB identity</th>
<th>eNodeB</th>
<th>UE</th>
<th>S1 AP id</th>
<th>Radio Access Technologies capabilities</th>
<th>Security key (for the encryption)</th>
<th>Evolved Packet System bearer id</th>
</tr>
</thead>
</table>

80
Appendix D

Interview questionnaires

D.1 Preliminary questionnaire

Preliminary questionnaire

The aim of this questionnaire is to give you ideas about the topics that I would like to talk about during the interview.

About you

1. How long have you been working with the LTE base station product development?
2. Please explain briefly what are the tasks you perform here at Ericsson (related to the base station or not).

About maintainability

3. What is maintainability for you?
4. How would you qualify the maintainability of the LTE base station product development? Why?
5. What aspects of the implementation are improving/reducing its maintainability?
6. What would you like to change in the implementation to make it less complex?
Questionnaire for the expert evaluation of the current implementation

The aim of this questionnaire is to give directions to the experts in order to get their insights about the maintainability of the current implementation of a LTE base station and more particularly of the RRC Protocol.

About the expert

1. Name of the expert: __________________________________________

2. How long have you been working with the LTE base station implementation?

3. Please explain briefly what are the tasks you perform here at Ericsson (related to the base station or not).

About maintainability in general

4. What is maintainability for you?

About the maintenance of the base station implementation

5. Could you describe what the maintenance environment looks like? Some sketches of the organization of the software used are planed to be made on separate sheets of paper.

6a. Could you describe usual maintenance tasks? (What needs to be done/How it is done/Order them from the one that is the most often performed to the least often performed)

6b. How often do you perform them?

7a. When performing maintenance tasks, do you sometimes spend time on part of the code non-directly related to the original task? □ Yes □ No

7b. If yes, on which parts of the code?

7c. If yes, is this more or less time-consuming than the original task alone?

8a. Would you say that a clear separation of concerns is achieved in the implementation? (protocol logic, state handling, event handling, concurrent execution)
□ No
□ Rather no
□ Rather yes
□ Yes

8b. Why?

9. Please give a grade (between 1 and 10) to the maintainability of the current implementation. (1 means no maintainability at all and 10 means perfect maintainability)

10. What makes maintaining the code difficult for you?

11. What makes maintaining the code easy for you?

12. What would you like to change in the current implementation to make it more maintainable (even if not technically possible)?

More specific aspects of maintainability

Understandability

13. How much time did it take you to understand the implementation? □ I don’t remember
□ Time:(how long?) ___________________

□ I don’t remember
□ Time:(how long?) ___________________
Expert evaluation of the current implementation

14a. Would you say that understanding the base station implementation was?
- Difficult
- Rather difficult
- Rather easy
- Easy
- Doesn’t remember

14b. Why?

Simplicity

15a. Do you think that it is possible to make the implementation simpler?  □ No  □ Yes

15b. Why?

15c. If yes, how?

Expandability

16a. Please evaluate the following task using a number between 1 and 10 (1 means it’s extremely
difficult and 10 that it’s extremely easy):

You have to add a new message in the middle of the current message sequence.

16b. Check all the following words/sentences that would apply to the task:
- time-consuming
- quick
- will probably introduce errors (error-prone)
- won’t introduce errors
- hard to locate where to modify the code
- easy to locate where to modify the code
- hard to test
- easy to test

16c. Could you explain why?

Making changes

17a. Please evaluate the following task using a number between 1 and 10 (1 means it’s extremely
difficult and 10 that it’s extremely easy):

You have to change the behavior of the program when a specific message is received.

17b. Check all the following words/sentences that would apply to the task:
- time-consuming
- quick
- will probably introduce errors (error-prone)
- won’t introduce errors
- hard to locate where to modify the code
- easy to locate where to modify the code
- hard to test
- easy to test

17c. Could you explain why?
Readability

18a. Please evaluate the following task using a number between 1 and 10 (1 means it’s extremely difficult and 10 that it’s extremely easy):

You need to follow the flow of a protocol procedure from its beginning to its end.

18b. Check all the following words/sentences that would apply to the task:

- time-consuming
- quick
- hard to follow the different steps of the flow
- easy to follow the different steps of the flow
- hard to locate where to look for the code
- easy to locate where to look for the code
- Other: ________________

18c. Could you explain why?

Testability

19a. Please evaluate the following task using a number between 1 and 10 (1 means it’s extremely difficult and 10 that it’s extremely easy):

You need to test a specific part of the procedure.

19b. Check all the following words/sentences that would apply to the task:

- time-consuming
- quick
- hard to determine the test cases
- easy to determine the test cases
- hard to isolate the code to test
- easy to isolate the code to test
- hard to run the test
- easy to run the test
- Other: ________________

19c. Could you explain why?

Others

20. Is changing the paradigm a realistic idea? What could prevent this change?
Appendix E

Evaluation questionnaire
Evaluation Questionnaire

You are presented two small computer software programs. Each program is implementing the same task, and implementing the same functional and non-functional requirements. The task implemented is typical for telecommunications control systems and is modelled from the RRC connection setup procedure in LTE. Several sequence diagrams modelling the different flows of the procedure implemented are available in the evaluation material.

The programs are implemented using two different programming paradigms: one is using the object-oriented programming paradigm and the other is using the functional reactive programming paradigm. Both programs are implemented in what is regarded as idiomatic style for their respective paradigms.

The first task is to assess the two programs with regards to maintainability. Hence you are asked to answer two identical sets of questions, one for each paradigm. Please give as detailed answers as possible and do not hesitate to write comments.

The second task is to compare the two programs with regards to several maintainability characteristics.

*How to answer?* Fill in the online questionnaire.

1. Your name: ____________________________

2. *How long have you been working with software?*
   *Note:* Including both experience within the software industry and other software development experience (e.g. open source projects)

3. Please explain briefly what are the tasks you perform here at Ericsson

4. *What programming paradigm(s) are you experienced in?*
   *Note:* Examples of programming paradigms: functional, object-oriented, declarative, imperative, reactive, etc...

5. Do you have a favourite programming paradigm? If yes, please write which one

2 First task: Maintainability assessment

This part of the questionnaire is separated into two sets of questions, one for the object-oriented programming paradigm and one for the functional reactive programming paradigm. The two sets are identical. You can choose which programming paradigm you want to assess first.

The statements are gathered into categories corresponding to the program characteristic that is evaluated. You are asked to indicate to which extent you agree or disagree with the given statements using a scale from 1 to 6. The meaning of each response is provided in the following table. Please read it carefully.
### Evaluation Questionnaire

<table>
<thead>
<tr>
<th>RESPONSE</th>
<th>MEANING</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Decidedly Disagree</td>
<td>This response indicates the program being evaluated is terrible with respect to the attribute addressed</td>
</tr>
<tr>
<td>2</td>
<td>Moderately Disagree</td>
<td>This response indicates the program being evaluated is very poor with respect to the attribute addressed</td>
</tr>
<tr>
<td>3</td>
<td>Somewhat Disagree</td>
<td>This response indicates the program being evaluated is unacceptable with respect to the attribute addressed</td>
</tr>
<tr>
<td>4</td>
<td>Somewhat Agree</td>
<td>This response indicates the program being evaluated is barely acceptable with respect to the attribute addressed</td>
</tr>
<tr>
<td>5</td>
<td>Moderately Agree</td>
<td>This response indicates the program being evaluated is very good with respect to the attribute addressed</td>
</tr>
<tr>
<td>6</td>
<td>Decidedly Agree</td>
<td>This response indicates the program being evaluated is excellent with respect to the attribute addressed</td>
</tr>
</tbody>
</table>

Some glossary notes are given to make sure that the statement being assessed has the same meaning for everyone. However, if something should not been clear, please contact me for clarifications.

### 2.1 Object-oriented programming paradigm

#### Simplicity

Simple programs are easier for programmers to understand. Simplicity can be decomposed into program size, control structure complexity, data structure complexity and straightforward coding. Maintainability will be improved when the maintainer has fewer things to understand and if the techniques and structures used are basic ones.

6. The program contains a reasonable number of executable statements with regards to its purpose.

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree  

**Glossary:** An executable statement is a language statement, excluding comments, data declarations and variable/constant declarations.

Please write any relevant information that made you choose this alternative

7. The program contains a manageable amount of branching

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree  

**Glossary:** A branching is a non-sequential execution caused by the use of control statements (If-then, case, ..) or of explicit transfers of control instructions (Goto).

Please write any relevant information that made you choose this alternative

8. The program contains manageable levels of nesting

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree  

**Glossary:** Nesting refers to the use of control statement(s) inside another control statement.

Please write any relevant information that made you choose this alternative

9. The program is understandable

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree  

**Glossary:** A part of the program is understandable if you are able to understand what it does.

Please write any relevant information that made you choose this alternative
10. The program clearly allow you to follow the flow of the implemented RRC Connection Setup procedure

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Reminder: Several sequence diagrams representing the different flows of the implemented RRC connection setup procedure are part of the evaluation material.

Please write any relevant information that made you choose this alternative

11. Overall, the program is simple

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Glossary: Simplicity can be decomposed into program size, control structure complexity, data structure complexity and straightforward coding. The degree of simplicity of a source code is related to the understandability of the program.

Please write any relevant information that made you choose this alternative

Modularity

Modular software is composed of largely independent parts. Modularity is important because it is easier to understand and manage small pieces of code than large and complex ones.

12. The program clearly separates different concerns from each other in the code

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Glossary: Different concerns can be for example: protocol logic, state handling, event handling, concurrent execution.

Please write any relevant information that made you choose this alternative

13. The program exhibits loose coupling between its different parts

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Glossary: Coupling describes the degree of interdependence between program parts.

Please write any relevant information that made you choose this alternative

14. The program exhibits high cohesion within its different parts

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Glossary: Cohesion describes the strength of association of the elements within the same program part.

Please write any relevant information that made you choose this alternative

15. Overall, the program is modular

- □ 1 Decidedly Disagree  □ 2 Moderately Disagree  □ 3 Somewhat Disagree  □ 4 Somewhat Agree
- □ 5 Moderately Agree  □ 6 Decidedly Agree

Glossary: Modular software is composed of largely independent parts.

Please write any relevant information that made you choose this alternative
Expandability

A program is expandable when it is easy to make changes to code and to add new parts to it.

16. The program clearly describes how the flow of the implemented RRC Connection Setup procedure is impacted by the handling of an event

☐ 1 Decidedly Disagree ☐ 2 Moderately Disagree ☐ 3 Somewhat Disagree ☐ 4 Somewhat Agree
☐ 5 Moderately Agree ☐ 6 Decidedly Agree

Glossary: It is easy to determine what happens when an event occur and how it affects the flow of the implemented RRC Connection Setup procedure.

Please write any relevant information that made you choose this alternative

17. Overall, the program is expandable

☐ 1 Decidedly Disagree ☐ 2 Moderately Disagree ☐ 3 Somewhat Disagree ☐ 4 Somewhat Agree
☐ 5 Moderately Agree ☐ 6 Decidedly Agree

Glossary: In this program, expanding it could mean: modifying what happens when a certain message is received, adding a message in the middle of the message sequence, add additional information to the state that is stored in the eNodeB, etc...

Please write any relevant information that made you choose this alternative

Maintainability

18. Overall, the program is maintainable

☐ 1 Decidedly Disagree ☐ 2 Moderately Disagree ☐ 3 Somewhat Disagree ☐ 4 Somewhat Agree
☐ 5 Moderately Agree ☐ 6 Decidedly Agree

Glossary: Maintainability is evaluated regarding the simplicity, understandability, modularity and expandability of the program.

Please write any relevant information that made you choose this alternative

2.2 Functional reactive programming paradigm

Simplicity

Simple programs are easier for programmers to understand. Simplicity can be decomposed into program size, control structure complexity, data structure complexity and straightforward coding. Maintainability will be improved when the maintainer has fewer things to understand and if the techniques and structures used are basic ones.

19. The program contains a reasonable number of executable statements with regards to its purpose.

☐ 1 Decidedly Disagree ☐ 2 Moderately Disagree ☐ 3 Somewhat Disagree ☐ 4 Somewhat Agree
☐ 5 Moderately Agree ☐ 6 Decidedly Agree

Glossary: An executable statement is a language statement, excluding comments, data declarations and variable/constant declarations.

Please write any relevant information that made you choose this alternative

20. The program contains a manageable amount of branching

☐ 1 Decidedly Disagree ☐ 2 Moderately Disagree ☐ 3 Somewhat Disagree ☐ 4 Somewhat Agree
☐ 5 Moderately Agree ☐ 6 Decidedly Agree

Glossary: A branching is a non-sequential execution caused by the use of control statements (If-then, case, ..) or of explicit transfers of control instructions (Goto).

Please write any relevant information that made you choose this alternative
21. The program contains manageable levels of nesting
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: Nesting refers to the use of control statement(s) inside another control statement.

Please write any relevant information that made you choose this alternative

22. The program is understandable
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: A part of the program is understandable if you are able to understand what it does.

Please write any relevant information that made you choose this alternative

23. The program clearly allow you to follow the flow of the implemented RRC Connection Setup procedure
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Reminder: Several sequence diagrams representing the different flows of the implemented RRC connection setup procedure are part of the evaluation material.

Please write any relevant information that made you choose this alternative

24. Overall, the program is simple
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: Simplicity can be decomposed into program size, control structure complexity, data structure complexity and straightforward coding. The degree of simplicity of a source code is related to the understandability of the program.

Please write any relevant information that made you choose this alternative

Modularity
Modular software is composed of largely independent parts. Modularity is important because it is easier to understand and manage small pieces of code than large and complex ones.

25. The program clearly separates different concerns from each other in the code
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: Different concerns can be for example: protocol logic, state handling, event handling, concurrent execution.

Please write any relevant information that made you choose this alternative

26. The program exhibits loose coupling between its different parts
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: Coupling describes the degree of interdependence between program parts.

Please write any relevant information that made you choose this alternative

27. The program exhibits high cohesion within its different parts
   □ 1 Decidedly Disagree □ 2 Moderately Disagree □ 3 Somewhat Disagree □ 4 Somewhat Agree
   □ 5 Moderately Agree □ 6 Decidedly Agree
Glossary: Cohesion describes the strength of association of the elements within the same program part.

Please write any relevant information that made you choose this alternative
28. Overall, the program is modular
☐ 1 Decidedly Disagree  ☐ 2 Moderately Disagree  ☐ 3 Somewhat Disagree  ☐ 4 Somewhat Agree  
☐ 5 Moderately Agree  ☐ 6 Decidedly Agree  
*Glossary:* Modular software is composed of largely independent parts.  
*Please write any relevant information that made you choose this alternative.*

**Expandability**
A program is expandable when it is easy to make changes to code and to add new parts to it.

29. The program clearly describes how the flow of the implemented RRC Connection Setup procedure is impacted by the handling of an event
☐ 1 Decidedly Disagree  ☐ 2 Moderately Disagree  ☐ 3 Somewhat Disagree  ☐ 4 Somewhat Agree  
☐ 5 Moderately Agree  ☐ 6 Decidedly Agree  
*Glossary:* It is easy to determine what happens when an event occur and how it affects the flow of the implemented RRC Connection Setup procedure.  
*Please write any relevant information that made you choose this alternative.*

30. Overall, the program is expandable
☐ 1 Decidedly Disagree  ☐ 2 Moderately Disagree  ☐ 3 Somewhat Disagree  ☐ 4 Somewhat Agree  
☐ 5 Moderately Agree  ☐ 6 Decidedly Agree  
*Glossary:* In this program, expanding it could mean: modifying what happens when a certain message is received, adding a message in the middle of the message sequence, add additional information to the state that is stored in the eNodeB, etc...  
*Please write any relevant information that made you choose this alternative.*

**Maintainability**

31. Overall, the program is maintainable
☐ 1 Decidedly Disagree  ☐ 2 Moderately Disagree  ☐ 3 Somewhat Disagree  ☐ 4 Somewhat Agree  
☐ 5 Moderately Agree  ☐ 6 Decidedly Agree  
*Glossary:* Maintainability is evaluated regarding the simplicity, understandability, modularity and expandability of the program.  
*Please write any relevant information that made you choose this alternative.*

### 3 Second task: Comparison of the two programming paradigms

In this second part of the questionnaire, you are asked to compare the two programs that you have been given with regards to several characteristics related to maintainability.

32. Indicate the programming paradigm that exhibits the simplest program:
☐ 1- Object-oriented programming paradigm  ☐ 2- Functional reactive programming paradigm  
*Please explain your decision*  

33. Indicate the programming paradigm that exhibits the most understandable program:
☐ 1- Object-oriented programming paradigm  ☐ 2- Functional reactive programming paradigm  
*Please explain your decision*  

34. Indicate the programming paradigm that exhibits the most modular program:
☐ 1- Object-oriented programming paradigm  ☐ 2- Functional reactive programming paradigm  
*Please explain your decision*  

35. Indicate the programming paradigm that exhibits the most expandable program:
☐ 1- Object-oriented programming paradigm  ☐ 2- Functional reactive programming paradigm  
*Please explain your decision*  

36. Indicate the programming paradigm that exhibits the most maintainable program:
☐ 1- Object-oriented programming paradigm  ☐ 2- Functional reactive programming paradigm  
*Please explain your decision*
4 General questions

These last questions concerns the two programs in general. You also have the possibility to add some comments.

Object-oriented programming paradigm program

37. How much time did you spend studying the object-oriented programming paradigm program?
   □ < 30 min
   □ 30 min - 1h
   □ 1h - 1h30
   □ 1h30 - 2h
   □ > 2h

38. How would you describe your understanding of the object-oriented programming paradigm program?
   □ I have a poor understanding of it
   □ I have a bad understanding of it
   □ I have a quite bad understanding of it
   □ I have a quite good understanding of it
   □ I have a good understanding of it
   □ I have a perfect understanding of it
   □ Other: __________________

39. Free expression on the object-oriented programming paradigm program

Functional reactive programming paradigm program

40. How much time did you spend studying the functional reactive programming paradigm program?
   □ < 30 min
   □ 30 min - 1h
   □ 1h - 1h30
   □ 1h30 - 2h
   □ > 2h

41. How would you describe your understanding of the functional reactive programming paradigm program?
   □ I have a poor understanding of it
   □ I have a bad understanding of it
   □ I have a quite bad understanding of it
   □ I have a quite good understanding of it
   □ I have a good understanding of it
   □ I have a perfect understanding of it
   □ Other: __________________

42. Free expression on the functional reactive programming paradigm program

General comments

43. Free expression

Thank you for participating!
Appendix F

Analysis of the maintainability assessment

F.1 Answers frequencies

Table F.1: Frequencies of the answers for the OOP prototype

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table F.2: Frequencies of the answers for the FRP prototype

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F.2 Descriptive statistics

Table F.3: Descriptive statistics for the OOP prototype

<table>
<thead>
<tr>
<th>Question number</th>
<th>Agreement factor</th>
<th>Range</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.60</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2.75</td>
<td>1.50</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3.25</td>
<td>1.71</td>
</tr>
<tr>
<td>8</td>
<td>0.66</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.00</td>
<td>1.41</td>
</tr>
<tr>
<td>9</td>
<td>0.56</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.50</td>
<td>1.29</td>
</tr>
<tr>
<td>10</td>
<td>0.66</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4.00</td>
<td>1.41</td>
</tr>
<tr>
<td>11</td>
<td>0.69</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>4.25</td>
<td>1.26</td>
</tr>
<tr>
<td>12</td>
<td>0.66</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>13</td>
<td>0.58</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3.50</td>
<td>1.92</td>
</tr>
<tr>
<td>14</td>
<td>0.59</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.75</td>
<td>1.50</td>
</tr>
<tr>
<td>15</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>16</td>
<td>0.56</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>17</td>
<td>0.56</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.50</td>
<td>1.73</td>
</tr>
<tr>
<td>18</td>
<td>0.66</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3.50</td>
<td>1.73</td>
</tr>
</tbody>
</table>
### Table F.4: Descriptive statistics for the FRP prototype

<table>
<thead>
<tr>
<th>Question number</th>
<th>Agreement factor</th>
<th>Range</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>0.69</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>20</td>
<td>0.69</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>21</td>
<td>0.69</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4.25</td>
<td>0.96</td>
</tr>
<tr>
<td>22</td>
<td>0.5</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>3.75</td>
<td>1.71</td>
</tr>
<tr>
<td>23</td>
<td>0.5</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>4.25</td>
<td>1.71</td>
</tr>
<tr>
<td>24</td>
<td>0.56</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3.50</td>
<td>1.29</td>
</tr>
<tr>
<td>25</td>
<td>0.88</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4.50</td>
<td>1.00</td>
</tr>
<tr>
<td>26</td>
<td>0.75</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4.50</td>
<td>0.58</td>
</tr>
<tr>
<td>27</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>28</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>29</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5.00</td>
<td>0.82</td>
</tr>
<tr>
<td>30</td>
<td>0.69</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4.75</td>
<td>0.96</td>
</tr>
<tr>
<td>31</td>
<td>0.69</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4.75</td>
<td>0.96</td>
</tr>
</tbody>
</table>
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Klervie Toczé