Maintainability of Functional Reactive Programs in a Telecom Server Software

Klervie Toczé
Ericsson AB
Linköping, Sweden
klervie.tocze@ericsson.com

Maria Vasilevskaya
Linköping University
Sweden
maria.vasilevskaya@liu.se

Patrik Sandahl
Ericsson AB
Linköping, Sweden
patrik.sandahl@ericsson.com

Simin Nadjm-Tehrani
Linköping University
Sweden
simin.nadjm-tehrani@liu.se

ABSTRACT

Functional Reactive Programming (FRP) is claimed to be a good choice for event handling applications. Current object-oriented telecom applications are known to suffer from additional complexity due to event handling code. In this paper we study the maintainability of FRP programs in the telecom domain compared to traditional object-oriented programming (OOP), with the motivation that higher maintainability increases the service quality and decreases the costs. Two implementations of the same procedure are created: one using Haskell and the reactive-banana FRP framework and one using C++ and the OOP paradigm. Four software experts each with over 20 years of experience and three development engineers working on a product subject to study were engaged in evaluations, based on a questionnaire involving five different aspects of maintainability. The evaluations indicate a higher maintainability profile for FRP compared with OOP. This is confirmed by a more detailed analysis of the code size.

CCS Concepts

•Applied computing → Telecommunications; •Software and its engineering → Maintaining software;

1. INTRODUCTION

As mobile networks grow and become prevalent in our lives, the users expect faster and more reliable connections despite the increased complexity of telecom server software. The core of such software is a large set of communication protocols that inherently requires to handle a lot of reactive behaviours. Implementation of these behaviours using the traditional programming paradigm used in the telecom context, i.e. OOP, creates so called accidental complexity into already complex software. This paper contributes with practical experience on the potential of FRP for reducing the complexity associated with handling reactive behaviours.

FRP is a fairly new (around 20 years) area of programming whose aim is to combine the strengths of the functional approach with the need to express reactive behaviour. FRP uses the *declarative* paradigm, meaning that a programmer focuses on *what* to program and the compiler decides *how* this will be done. Distinctly from an OOP program, when a value changes in a FRP program, all the dependent values are automatically updated. The FRP paradigm aims at abstracting away code needed to handle events and claims to produce small and clear programs, which are easy to reason about. In a nutshell, FRP appears to be very promising for reducing the complexity of telecom server applications that are inherently reactive. That is, in turn, expected to improve their maintainability. However, this claim of improved maintainability should be first extensively evaluated.

In this paper, we report our efforts towards providing such an evidence. We conduct a maintainability study of the FRP paradigm applied to a part of the Long-Term Evolution (LTE) base station software. Our goal is to evaluate to what extent the maintainability of this software improves when it is implemented as FRP in comparison with its OOP implementation. In particular, we carry out expert-based evaluations where the experts assess two prototypes of the same communication protocol implemented in those two paradigms. Additionally, we compute software metrics that reflect objective complexity of the implementations and compare these results with the outcome of our expert assessment. We also report preliminary results on performance aspects of evaluated implementations.

2. SYSTEM DESIGN

To conduct expert-based maintainability assessment of functional reactive and object-oriented paradigms we first designed a reference model which is a simplified version of the Radio Resource Control (RRC) connection procedure from the LTE standard. This reference model preserves the core messaging and reactive behaviours and focuses on two aspects: receiving/sending messages and updating stored information. From this reference model, we created two prototypes implemented in Haskell (FRP prototype) and C++ (OOP prototype). These two prototypes served as targets of the maintainability evaluation. The main goal of using

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC 2016, April 04-08, 2016, Pisa, Italy

© 2016 ACM. ISBN 978-1-4503-3739-7/16/04…$15.00

DOI: http://dx.doi.org/10.1145/2851613.2851954

a reference model instead of the fully-fledged RRC protocol is to achieve fairness in evaluations by abstracting away unnecessary details of a complicated protocol. The reference model and its implementation are briefly presented below.[1]

The reference model consists of 15 messages exchanged between the User Equipment (UE), the eNodeB and the Mobility Management Entity (MME), three high-level components of a LTE network. The content of the these exchanged messages is most of the time simplified. We keep only those fields of information that are needed for modifying the stored information or for sending the response.

The three components run on the same Linux machine (Ubuntu 14.04 LTS 64-bit equipped with a Intel Core i7-4500U Haswell CPU and 8Gb of DDR3 RAM) and they communicate with each other using the TCP protocol.[2] The components were compiled using version 4.8.1 of GCC (OOP prototype) and version 7.6.3 of GHC (FRP prototype).

3. MAINTAINABILITY EVALUATION

The maintainability evaluation of the two prototypes consisted of a combination of expert assessments and of structural measures. This combination is considered to be the most efficient way to assess software maintainability [**?**].

3.1 Design of the Expert Assessment

Seven software experts from Ericsson[3] participated in the maintainability assessment. Four experts have 20 years of experience in software development but not in the LTE base station. Prior to the study, two experts named the functional paradigm as their favourite and two named the object-oriented one. This means that the expert group was not biased towards one paradigm. Three other experts were used as application experts when constructing the questionnaire. Although seven experts may seem a small number, we have not seen a study with that many experts, who moreover have a long practical experience of software maintainability.

The assessment is performed using a questionnaire built of four parts: (1) expert information, (2) separate assessment, (3) comparative assessment, and (4) expert self-evaluation.

The first part contains four questions regarding the experts' experience. The second part investigates maintainability of each prototype considered in isolation. It contains 13 questions for each prototype assessing different maintainability characteristics (simplicity, modularity, expandability) and maintainability as a whole. To construct this second part, we adapt the Air Force Operational Test and Evaluation Center (AFOTEC) pamphlet [**?**], which provides guidelines and standardized questionnaires to evaluate software maintainability using experts. We selected a subset of six questions (out of 35) relevant to our study. Those questions (and the ones additionally included by the authors) were validated by the three experts working on the LTE base station who possess good understanding of the maintainability problems. The comparative assessment part contains 5 questions evaluating which prototype is the best with regards to simplicity, understandability, modularity, expandability and maintainability as a whole. Finally, the last part of the questionnaire contains 7 questions asking, for example, how much time was spent studying each program.

To conduct the evaluation, the four reviewing experts were provided with an online version of the designed questionnaire, the source code for both prototypes, and the sequence diagrams of the reference model.

3.2 Separate Assessment

The experts were asked to indicate to which degree they agree to 13 statements related to three maintainability characteristics (simplicity, modularity, and expandability) and to maintainability as a whole. Simplicity can be decomposed into program size, control structure complexity, data structure complexity, straightforward coding and understandability. A modular software is composed of largely independent parts. A program is expandable when it is easy to make changes and to add new parts to it. To indicate the degree of agreement, we adopted the AFOTEC scale [**?**] which is a scale from 1 (decidedly disagree) to 6 (decidedly agree).

Figure 1 plots the average score for each question for both prototypes. Moreover, we categorised these scores in three maintainability levels derived from AFOTEC maintainability thresholds.

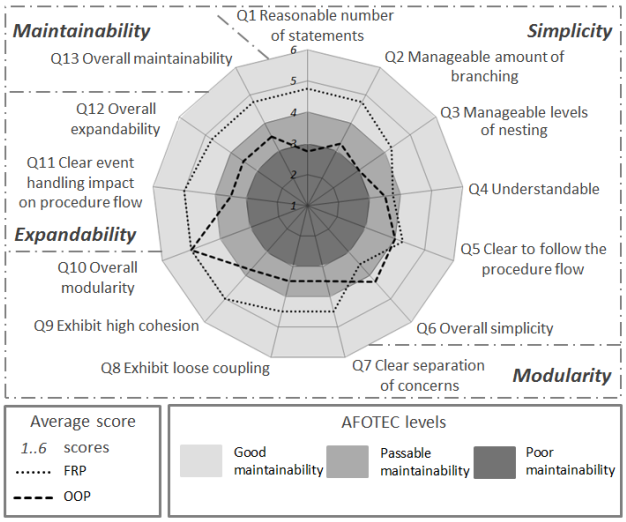

Figure 1: Average score on each question

Figure 1 clearly visualises that on the whole the FRP prototype has higher scores than the OOP prototype. There are only two questions where the OOP prototype performs equally or outperforms the FRP one. These two questions assess the overall simplicity (question 6) and modularity (question 10) of the prototypes.

Figure 1 also shows that 11 out of 13 attributes were evaluated as having a good maintainability level for the FRP prototype (the light-grey area). There are only two attributes (overall understandability and simplicity), that correspond to questions 4 and 6, that were assessed as having a passable maintainability level. On the contrary, the OOP prototype has three questions with a good maintainability level (i.e. questions 5, 6, and 10) and one with a poor maintainability level (i.e. question 1), the majority of the questions having a passable maintainability level.

3.3 Comparative Assessment

[1]Details can be found in the thesis by Toczé [**?**]
[2]In the context of the reference model, it is similar to the real-time signalling used in the real base station
[3]http://www.ericsson.com/

In order to validate that the separate assessment is reasonable, the comparative assessment asks the same experts to compare the two prototypes with each other with regards to the previous three maintainability characteristics (simplicity, modularity and expandability), and to maintainability as a whole. We also look at a specific part of simplicity: understandability. The 5 questions in this part asked the experts to choose which prototype was the best with regards to the five characteristics mentioned above. Figure 2 shows the frequencies of the obtained answers.

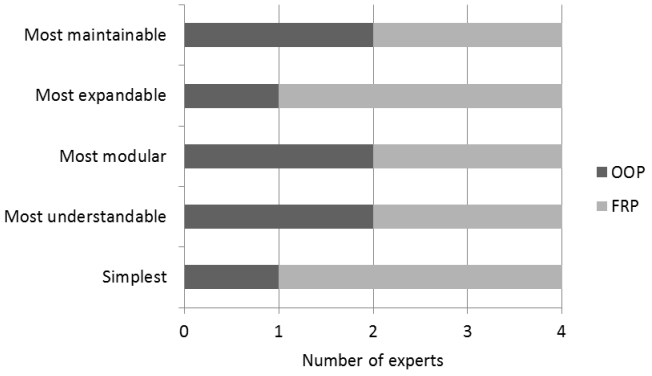

Figure 2: Results of the comparative assessment

According to Figure 2 the experts assess the FRP program as the simplest one. Interestingly this may seem in contradiction with the results from the separate assessment where the average score given to question 6 ("Overall, the program is simple") is higher for the OOP prototype. This indicates that simplicity is difficult to assess "as a whole" for the experts and should instead be only investigated through more precise statements. Therefore, we decided not to consider the results on simplicity as a whole for our conclusions.

The results for understandability indicate that the experts assess both programs as equally understandable. However, as it was mentioned by two experts in the comments, this may be biased towards the paradigm which the expert knows best. According to the experts, the programs are equally modular and the FRP one is the most expandable. However, one expert indicates that he/she found both prototypes equally expandable so choosing one over the other may not be relevant. Finally, Figure 2 shows that both programs are assessed as equally maintainable.

Hence, apart from the issue with the simplicity, the comparative assessment confirms the evaluation results obtained for the separate assessment.

3.4 Software Metrics

In this section, we compare the two prototypes with regards to software metrics related to the size of the code. Riaz et al. [**?**] conclude in their review that the code size metrics are the most successful software maintainability predictors that can be calculated easily.

The results show that when taking into account manually written files the OOP prototype represents 1469 lines of code. This is 1.5 times more lines than in the FRP prototype. It is a significant difference: more code to write means a higher probability to introduce mistakes which will later require maintenance.

When focusing on event-handling code, differences between the prototypes are also distinct. For example, receiving a UE message (decoding it and calling the function to handle it) is coded in 12 lines in the OOP prototype and in 2 lines in the FRP one. This is because the FRP prototype, in addition to having higher abstractions for event-handling with the FRP framework, benefits also from the high abstraction level of Haskell. Since Haskell and C++ are good representatives for each paradigm, our study contributes to confirming the claim of functional programming to produce more concise code.

3.5 Preliminary Performance Evaluation

Although performance aspects were not the main concern in our study, a preliminary performance evaluation was conducted using two scenarios: a peak load and a spread load. In the peak load study, an increasing number of UEs (from 100 to 1000) were powered on simultaneously, while the spread load study consisted in simulating a mean arrival rate of 100 UEs per second (up to 5000 attached UEs).

The OOP prototype is faster to attach UEs than the FRP prototype in both studies. It is up to 10 times faster in a peak load scenario with 1000 UEs and 8 times faster in a spread load scenario. Moreover, the CPU share is on average 51.06% (std. dev. is 5.45%) for the FRP prototype and 6.75% (std. dev. is 0.35%) for the OOP prototype.

4. CONCLUSIONS

This study was initiated by a group of software engineers considering alternatives to traditional paradigms with the long term goal of shifting part of the LTE software to another more maintainable paradigm. Such a shift would be critical and can not rely on "common beliefs" but requires empirical evidence on maintainability and performance.

Our analysis showed that when the two programs were assessed separately, the FRP prototype obtained better scores than the OOP one. Those results were confirmed by the comparative assessment. The calculated software metrics showed that FRP gives more compact code than OOP. This implies that the OOP prototype is more complex and, thus, less maintainable than the FRP one.

Thus, we can conclude that FRP has a large potential to improve maintainability in telecom server software. However, performance aspects need to be investigated further, in particular in real situations with a non-simplified and optimized software.

5. REFERENCES

[1] Air Force Operational Test and Evaluation Center. Software maintainability evaluation guide. 1996.

[2] B. Anda. Assessing software system maintainability using structural measures and expert assessments. In *Proceedings of the 2007 IEEE International Conference on Software Maintenance (ICSM 2007)*, pages 204–213. IEEE, 2007.

[3] M. Riaz, E. Mendes, and E. Tempero. A systematic review of software maintainability prediction and metrics. In *Third International Symposium on Empirical Software Engineering and Measurement*, pages 367 – 377. IEEE, 2009.

[4] K. Toczé. Functional reactive programming as programming model for telecom server software. Master's thesis, Linköpings universitet.