

Reengineering of State Machines in Telecommunication Systems

Christof Mosler

Department of Computer Science 3, RWTH Aachen University

Ahornstr. 55, D-52074 Aachen

christof.mosler@rwth-aachen.de

Abstract

In our reengineering project, we regard architecture modeling and reengineering techniques of telecommunication systems. Our work concerns analysis and restructuring of such systems, including the re-design and re-implementation. As telecommunication systems are often planned and modeled using state machines, it seems reasonable to analyze the systems also on the more abstract representation level of state machines. This paper presents a reengineering approach, in which we extract the state machines from PLEX source code, verify their consistency with the original specification documents, and use the information for further reengineering steps.

1 Introduction

In the E-CARES¹ research project, we study concepts, languages, methods, and tools for understanding and restructuring of complex legacy systems from the telecommunications domain. The project is a cooperation between Ericsson Eurolab Deutschland GmbH (EED) and Department of Computer Science 3, RWTH Aachen University. The current system under study is Ericsson's AXE10, a mobile-service switching center (MSC) comprising more than ten million lines of code written in PLEX (**P**rogramming **L**anguage for **EX**changes) [2]. This language was developed in about 1970 at Ericsson and is still in wide use within the company. PLEX systems are embedded real-time systems using the signaling paradigm, thus they pose additional requirements regarding fault tolerance, reliability, availability, and response time.

According to the model introduced by Byrne [1], a reengineering process comprises three phases: reverse engineering, modification, and forward engineering. All of them are supported by our reengineering tool environment (for details see [3]). In the reverse engineering phase, we use different sources of information. The most important and reliable one is the source code of the PLEX system. For representation of the legacy systems, we follow a graph-based approach, i.e. all underlying structures are graphs and editing operations are specified by graph transforma-

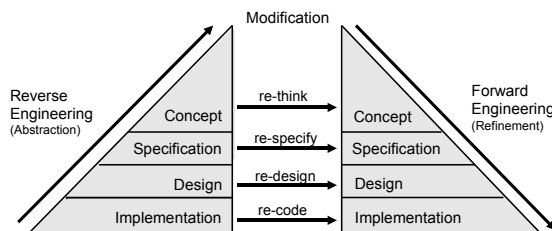


Figure 1: Reengineering Model by Byrne [1]

tion rules. We parse the source code and instantiate a graph describing the system structure, comprising its communication, control flow, and data flow. This *structure graph* consists of a main system node, sub-system nodes, block nodes, and the nodes for different program parts of each block (e.g. subroutines, signal entry points, data structures). The nodes are connected to each other by various kinds of edges (e.g. contains, goto, calls, from source, to target). On this graph, the user can perform various types of analyses by using different visualization and query techniques.

The tool supports also the modification phase by offering some complex algorithms for suggesting how to improve the legacy software. The user can interactively adapt the suggested transformations taking the semantics of the software into account or edit the graph manually. All modifications of the structure graph can later be propagated from the design level to the implementation level by using a TXL-based tool to generate the new source code.

2 Extraction and Modification of State Machines

When developing telecommunication systems, engineers often think in terms of state machines and protocols. Indeed, many parts of the AXE10 system were originally specified with state machines. We argue that reengineering can only be successful if this representation level is also considered. Up to now, all modification approaches in our project concerned the structure graph, corresponding to the design level of the model in figure 1. The extension presented in this paper focuses on the specification level, where parts

¹The acronym E-CARES stands for **E**ricsson **C**ommunication **A**rchitecture for **E**mbedded **S**ystems.

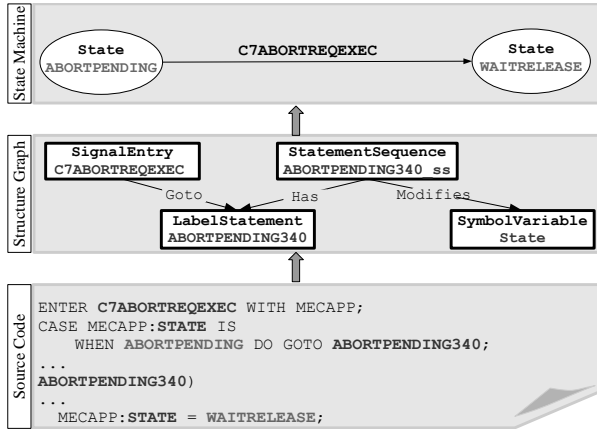


Figure 2: Extraction of State Machines

of the PLEX system are additionally represented as state machines. According to the model, we lift the abstraction level from design to specification level.

Generally, there is no explicit support for the implementation of state machines in PLEX. They are simulated by global enumeration variables whose possible values represent the possible states of the PLEX program during execution. Following the graph-based approach, our reengineering tool instantiates the state machines as independent graphs, allowing to apply different analyses and graph transformations. These *state machine graphs* are derived from the structure graph when certain patterns appear. Usually, these patterns comprise an assignment to the state variable, which indicates a transition in the state machine.

The example in figure 2 shows parts of the PLEX source code, the corresponding subgraph of the structure graph, and the extracted part of the state machine graph. The source code starts with an ENTER statement waiting for an incoming signal named C7ABORTREQEXEC. In the next line, a CASE statement on the STATE value of the received record MECAPP decides whether to jump to the statement sequence labeled ABORTPENDING340. Given the current block state ABORTPENDING, the new assignment of the STATE variable indicates the new state WAITRELEASE. The structure graph created for this code comprises all the information required for the state machine extraction. It is one of the major patterns used in our extraction algorithm. The extraction of the state machine starts at the signal entry points and looks for paths to statement sequences that change the state variable. For each assignment to the state variable there is a new transition created that is labeled with the signal name. The name of the source state is stored in the GOTO edge and the name of the target state is contained in the analyzed MODIFIES edge. The resulting state machine graph is shown in the upper part of the figure.

Before continuing developing concepts and tool extensions for state machine analysis, we wanted to

guarantee the utility of the extracted information. We extracted the state machines of several PLEX blocks and by using a simple parser compared them automatically with the specification files we got from Ericsson. Our final results showed that the extracted state machines contained all states and all relevant transitions described in the specification documents (e.g. for one of the major system blocks comprising 17 states and 53 transitions). All occasionally emerging problems during the analysis originated in outdated versions of our PLEX files.

After the extraction, the additional information visible in the state machine graphs can be taken into account when re-designing the structure graph. Currently, our most important area of application for state machine analysis is merging and splitting of PLEX blocks. Merging of small blocks into one bigger block can reduce the number of exchanged signals and thus improve the system performance. After the merging, the new block usually implements several state machines which could also be merged in order to reduce the state space. Splitting a block into several independent blocks is usually done when the block reaches a size which makes maintenance too painful. If a block implements several state machines, the splitting of the structure graph should result in two blocks each containing one of the original state machines. If the block contains only one state machine, a preceding splitting of this state machine and the adaption of the structure graph (e.g. by providing a new state variable and dividing the state space) are required before splitting the structure graph.

3 Outlook

The introduced extraction algorithm has been successfully implemented. Also some basic re-design algorithms for the structure graph considering the state machines have been developed. Future work concerns the study of simple operations directly for the state machine graphs and the corresponding changes to be propagated to the structure graph. The goal is to prove generally that automatic source code generation after restructuring of state machine graphs is possible.

References

- [1] BYRNE, ERIC J.: *A Conceptual Foundation of Software Re-engineering*. In *ICSM '92*, pages 226–235. IEEE Computer Society Press, 1992.
- [2] MARBURGER, ANDRÉ: *Reverse Engineering of Complex Legacy Telecommunication Systems*. Shaker Verlag, Aachen, Germany, 2004. ISBN 3-8322-4154-X.
- [3] MOSLER, CHRISTOF: *E-CARES Project: Reengineering of Telecommunication Systems*. In LÄMMEL, R., J. SARAIVA and J. VISSER (editors): *GTTSE'05*, number 4143 in *LNCS*, pages 437–448, Braga, Portugal, 2006. Springer.