E-CARES Research Project: Understanding Complex Legacy Telecommunication Systems

André Marburger

Lehrstuhl für Informatik III
RWTH Aachen
Ahornstraße 55
52074 Aachen
marand@i3.informatik.rwth-aachen.de

Dominikus Herzberg

Ericsson Eurolab Deutschland GmbH
Ericsson Allee 1
52134 Herzogenrath
Dominikus.Herzberg@eed.ericsson.se

April 19, 2000

Contents

1 Introduction

The importance of *embedded systems* for our daily life is rapidly increasing. More or less unnoticed they fulfil the task to control the "behavior" of technical systems ranging from drinks dispensers to big industrial complexes. Embedded *real-time systems* play a special role. Besides typical requirements of embedded systems concerning i.e. reliability or fault-tolerance embedded real-time systems have to fulfil requirements concerning availability and response time in addition. Another property often found in embedded real-time systems is concurrency. One important field of application for embedded real-time systems is in telecommunications industry. The complexity of these systems is rapidly increasing while at the same time the software part becomes more and more important.

2 The E-CARES Re-engineering Approach

In the E-CARES[1] research coopertation between Ericsson Eurolab Deutschland GmbH (EED) and the Department of Computer Science III, RWTH Aachen, the subject of study is Ericsson's Mobile-service Switching Center (MSC) called AXE10. The cooperation aims to develop methods, concepts, and tools to support the processes of understanding and restructuring complex legacy telecommunication systems. A brief outline of the re-engineering approach is sketched in figure 1.

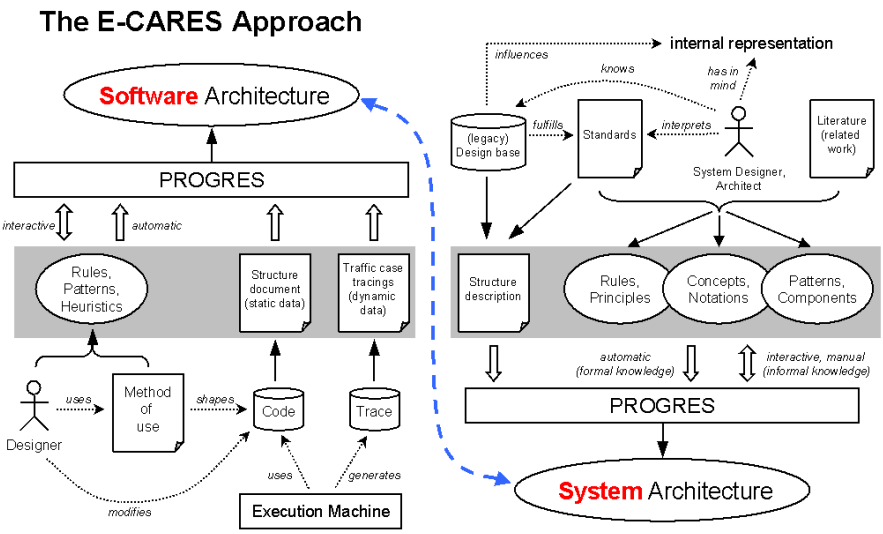

Figure 1: The top-down and bottom-up approach in E-CARES

The overall idea behind is to support the process of system understanding. That is, system designers should be able to understand the implemented system and the under-

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

lying system architecture without reading the source code and associated documentation. Suitable abstraction techniques comprising algorithms, heuristics, and notations are to be developed for that purpose. The difficulty is to distinguish vital from non-vital information. Our research project is determined by the combination of two apparently opposite approaches: a top-down approach from a systems perspective and a bottom-up approach from a "pure" software perspective. These will be briefly described in the following subsections.

2.1 The Top-Down Approach

The idea of the top-down approach is to identify a set of rules, principles, and concepts, which are typically used in the telecommunication domain for describing and modeling telecommunication systems along with a proper notation (ideally visual). Furthermore, patterns or components might be identified, which define a commonly used composition of conceptual entities. The problem is that these "intellectual tools" (concepts, patterns, etc.) are only partly explicitly defined; they have not reached the same level of "formal" maturity as known in the software engineering domain.

As shown in figure 1 standards (e.g. GSM, UMTS) are not the sole source of information. The system architect does not only read and interpret the standards, but has an intimate knowledge of the legacy. The problem of mediating between standards and actual implementations forces to reflect about deficiencies in the standards as well as in the implemented architecture. As a result, system designers/architects tend to develop their own mental representation of the problem and solution space, which is almost nowhere documented under public access or not documented at all.

Another source of information is literature, articles and related work. Related work includes languages as there are for example the Specification and Description Language (SDL) and the real-time profile of the Unified Modeling Language (UML). Typically, these languages condense basic concepts of their application domain.

Information about the system structure can be derived from the design base and standards. In Ericsson explicit structural information is stored in a data base, organized according to principles and rules defined by a framework called System 108. It reflects a functional grouping of products in a hierarchical manner and – on a higher level – a component-oriented system architecture. Implicit structural information is fixed in documents describing functional relations as there are for example protocol specifications, interworking descriptions and so on.[2] It is to be investigated to what extend implicit structural information can be extracted automatically from the documents and if it is of use for our considerations.

2.2 The Bottom-Up Approach

For a bottom-up approach, which we restrict to software only, code is definitively the authoritative source. Ericsson's in-house programming language called PLEX (Programming Language for EXchanges) structures code in *blocks*, self-contained units, which encapsulate data and code. The only way to communicate/interact between blocks is via *signals*. Whenever a signal is received by a block, this signal is the entry point to code execution in the block. So-called *job buffers* are some kind of signal stacks with different queuing priorities; this allows to prioritize and schedule different kind of "activities".

[2]The categorization *implicit* and *explicit* is based on the criteria if structural information is explicitly stored in a database or has to be implicitly derived from an information object.

The use of signals and their entry/exit points in the blocks determine code segments and relations between these segments; it is obvious that such segments and communication relations describe a structural model of the code and can easily be modeled as nodes and edges of a graph. Of course, further information like data structures should be respected as well. This type of structural description is attributed as “static data” in figure 1.

Further structural information can be derived from the fact that the designer, who writes and modifies the code, strictly follows conventions of coding, so-called design rules. This specific method of use does not only shape the code it implies also design patterns and heuristics. Heuristics include for example naming conventions; it is possible to get semantical information from names, which might improve the understanding of structural relations and dependencies.

While in the past the term “architecture” was mostly limited to the understanding of static structural aspects of a (software) system, a shift has happened: dynamic structural issues are generally recognized as being architecture-level issues. Independently of this insight, we found out that the understanding of so-called traffic cases is impossible without having a notion of how blocks are incarnated and linked together on execution time. Dynamic information can be retrieved from runtime tracings.

3 Conclusion

Although being in a prestudy phase of the research project it becomes already clear that we cannot see the two approaches as separated as indicated in figure 1. The view from the software side of the system will automatically be influenced by a systems view and vice versa. This is indicated by the dashed arrow between the two ellipsis.

The ability to handle and manipulate structural descriptions as graphs in the PROGRES-Environment allows us to correlate, combine and re-evaluate the different approaches described above – provided that suitable concepts, abstraction techniques, relations, heuristics, patterns and a notation have been identified.

The positive feedback we received on the “Design Base Navigator”, an early prototype of the idea presented, makes us quite confident that we are on the right track. It served as a proof of concept and showed the feasibility of the approach trying to visualize the block structure given by the signal interfaces and its grouping derived from the product structure. We could roughly verify that the block structure of two subsystems maps quite well to the GSM system architecture.