E-CARES Research Project: **Understanding Complex Legacy Telecommunication Systems**

Dominikus Herzberg

Ericsson Eurolab Deutschland GmbH
CNM – Node Product Unit MSC
Ericsson Allee 1
52134 Herzogenrath, Germany
Dominikus.Herzberg@eed.ericsson.se

André Marburger

Department of Computer Science III
University of Technology Aachen
Ahornstraße 55
52074 Aachen, Germany
marand@i3.informatik.rwth-aachen.de

Tony Jokikyyny

Ericsson Research / NomadicLab
Oy L M Ericsson Ab
02420 Jorvas, Finland
Tony.Jokikyyny@ericsson.com

WSR 2000

1 Introduction

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

2 The E-CARES Re-engineering Approach

In the E-CARES[1] research cooperation 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

[1]The acronym E-CARES stands for **E**ricsson **C**ommunication **AR**chitecture for **E**mbedded **S**ystems.

brief outline of the re-engineering approach is shown in figure 1. “E-CARES Metrics” (Ericsson Finland, NomadicLab) is a subproject of E-CARES and aims to support systems understanding and design decisions by measurements.

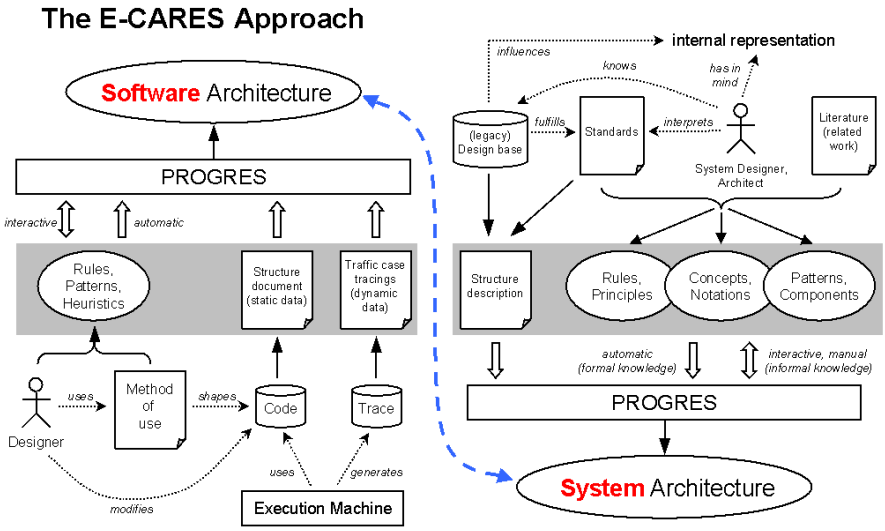

Figure 1: The top-down and bottom-up approach in E-CARES

The overall idea behind this, is to support the process of system understanding. That is, system designers should be able to understand the implemented system and the underlying system architecture without reading the source code and associated documentation. Suitable abstraction techniques comprising algorithms, heuristics, measurements and notations are to be developed for this 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 system’s perspective and a bottom-up approach from a “pure” software perspective. In addition, we will look for some practical measurements as a supporting approach. The following three subsections will briefly describe these approaches in more detail.

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; as yet they have not reached the same level of “formal” maturity as already obtained 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 also has an intimate knowledge of the legacy. The problem of mediating between standards and actual implementations forces one 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 documented almost nowhere with public access or is not documented at all.

Another source of information is literature, with its articles and related work. Related work includes languages, 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. At Ericsson's explicit structural information is stored in a database, 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, for example, protocol specifications, interworking descriptions and so on.[2] It remains to be investigated as to what extend implicit structural information can be extracted automatically from the documents and whether it is of use for our purposes.

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 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 the system to prioritize and schedule different kind of "activities".

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 considered 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 also implies design patterns and heuristics. Heuristics include naming conventions; it is possible to get semantical information from names, which could 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. Independent 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 upon execution time. Dynamic information can be retrieved from runtime tracings.

2.3 The Measurement Approach

To reliably measure something so abstract as a software system is far from trivial. Nevertheless, what else could be more convincing of system design quality than clear, quantitative measurement results? The major reason here for not understanding something properly is the inability to describe the systems with simple, *measurable* attributes. Additionally, having this kind of measurement in place would make it possible to define precise goals for these attributes, in order to quantitatively evaluate a system design.

Even though the attributes of complex systems could be effectively measured, there would be an additional problem left to solve; how to actually use the measurement data to gain better understanding of these systems? This requires a model of *system quality*, consisting of measurements of several attributes together with some equations between them. The definition of a valid model is clearly dependent on the specific system in question; its purpose, requirements, environment, etc. Usually, building such a model is a lot more difficult than creating the single measurements.

Furthermore, manual measurement data collection and presentation is often cumbersome and slow, resulting in a lack of measurement-based facts for decision making during early stages of system design. We have concluded that measurement should be an automated, integral part of the standard design process.

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

To support reuse and transfer of knowledge, it is also important to be able to collect and use the experience from previous designs. In general, it seems to be a good idea to have all measurement data available in one place, close to the developer where it can be taken into consideration each time the system design is modified. It remains to be investigated as to what extend measurement data can be extracted automatically from the code and other documents describing a system.

3 Conclusion

Although being in a prestudy phase of the research project, it has already became clear that we cannot see the top-down and bottom-up approaches as separate identities as indicated in figure 1. The software view will automatically be influenced by a system's view and vice versa. This is indicated by the dashed arrow between the two ellipses.

The ability to handle and manipulate structural descriptions as graphs in the PROGRES-Environment[3] allows us to correlate, combine and re-evaluate the different approaches described above – provided that suitable concepts, abstraction techniques, relations, measurements, 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.[4] It served as a proof of concept and showed the feasibility of the approach to try 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.

Measurement extraction during system design and *online* availability of the results are important steps towards creating a better understanding of complex systems. We intend to integrate this kind of technology into the PROGRES-Environment. Part of this work, a visual *measurement dashboard*, has already been investigated in a previous research project within Ericsson. We are confident that the facts provided by this technology will significantly support architectural decision-making. Of course, any such environment must also provide some means to define the used measurements and models.

[3] PROGRES (PROgramming with Graph Rewriting Systems) is an environment for manipulating graphs and has been developed at the Computer Science Department III, University of Technology Aachen (Prof. Dr. M. Nagl).

[4] The "Design Base Navigator" is based on work from Stefan Sandh (dpart.com) and Dominikus Herzberg (EED). Many thanks to Stephan Kruska (EED) for the JAVA implementation.