Influences between Software Architecture and its Environment in Industrial Systems – a Case Study

Goran Mustapic, Christer Norström, Anders Wall, Ivica Crnkovic,
Kristian Sandström, Johan Andersson, Joakim Fröberg

Mälardalen University
Department of Computer Engineering
Box 883
721 23 Västerås
+46 (0)21 10 70 35

{goran.mustapic, christer.norstrom, anders.wall, ivica.crnkovic,
kristian.sandstorm, johan.andersson, joakim.froberg}
@mdh.se

Abstract:

Software Architecture has received much attention in Software Engineering Research Community in the last decade. In this report, we have collected data from a number of real systems, which are successful and complex industrial systems. We tried to identify factors that have significant influence on their Software System Architecture lifecycle. The main goal of the investigation is to identify some key common factors that make architectures of these systems, successful or fail. The investigation was performed through a series of interviews with a number of key persons of research and development groups of successful international companies located in Sweden. The interviews were conducted in the late fall of 2003. This document contains the description of the case study setup and the material collected during interviews, without any analysis being done yet.

1. Introduction

To perform investigation of Software Architecture a case study was performed. According to Pflegeer [2], the following steps are involved in a case study: conception, hypothesis setting (particularly important as it describes what we measure and how we analyze the results), design, preparation, execution, analysis, dissemination and decision-making. Also, because case study usually compares one situation to another, some measures need to be taken to avoid bias and make sure that hypothesized relationships are tested. Possible techniques are: sister project, comparison to a general baseline and random selection.

In the preparation phase of the case study we performed several brainstorming meetings. The outcome of the meetings was a plan with the following main action items:

- Perform the case study based on a random selection of large companies. Only large companies were chosen because they own the most complex and long-lived systems and are likely to have data about several generations of systems.
- Create a list of question to guide the interview and create a material that is more suitable for the analysis. We decided to give a structure to the list based on architecture lifecycle.
- Setup interviews with people that have architect, chief designer or similar important role and know the history of the system. Interview was estimated to about two hours, and was conducted by one or preferably several researchers (authors of this report).
- After the interview, write a summary and send if to the interviewed architects for the review.
- Collect the results of the interviews in a technical report.

- Make a preliminary analysis of the interview and find out the similarities and differences between the cases.
- Setup a workshop with the interviewed and perform the analysis of the results.
- Publish the results of the case study as a scientific article.

The remainder of this document contains a list of interview questions and summarized and reviewed material from the interviews.

1.1 Interview questions

The first column of the table contains **factors** that influence architecture, and it should be interpreted as suggestions or examples. The purpose of having some common suggested factors is to more easily compare answers from different companies and find common important factors. The remaining three columns contain answers for the factors in the column one, related to the three different phases in architecture lifecycle. Answers will preferably contain a level of relevance for the particular factor, but also a short motivation why.

Levels can be the following: Very High (V), High (H), Medium (M), Low (L), None (N), Not Applicable (A).

An influence of the factor can be Implicit (I) or Explicit (E), depending on if the influence was planed, or not.

Factors That influence architecture	**Different phases in the architecture lifecycle**					
	Creation of architecture		**Architecture evolution**		**Architecture end**	
1. Software and system How would you order influences between: • software • system • hardware architectures (e.g. which one is determined first, second and third)? How are influences between SW and system architecture communicated (e.g. through requirements, what notation is used etc)?						
2. Inheritance/legacy Code Subsystems Experiences						
3. Business / application domain • Standards • Requirements • Type of customer • Volumes • Length of life • Non-functional requirements (NFR)						
4. Choice of technologies Who influences who and how (architecture and technologies)? In what extent? How to handle a mix of technologies?						

How do possible future technology changes influence architecture?						
5. Organization Who influences who and how? Work distribution and allocation. Distributed development. Are there third party contractors? If so, how does it influence the architecture? How to balance project and competence organization? Size (5 or 5000 developers needed to implement the system)? Dynamic (how often does the organization change). Maturity of the development organization and people competence						
6. Process What type of development process do you use and why? Does process influence architecture and how?						
7. Resources in architectural activities Calendar time for architectural activity People involved in architecture related activity						
8. Merger of companies						
9. Other						

Table 1. Questionnaire used to conduct architecture interviews

2. Interview summaries

In this section, we present the summaries of the individual interviews. In each section, a very brief description of the system is given, followed by the summarized data from the interview. The data was summarized in the following way:

- One of the interviewers was chosen as responsible for the summary;
- Each of the interviewers wrote down his own comments, and they were merged by the responsible person;
- After the merge, the summary was sent to the other interviewers for approval;
- Finally, the summary was sent to the interviewed for approval, and eventual feedback was then incorporated by the responsible person.

Because the goal of the interview was not to get answers of the questions in the questionnaire but to discuss the most important factors related to the system discussed, the summaries we present in the section below and in a slightly different format than the table above. To make the summaries in a more compact form, they will be presented as a list of items, rather than a table.

1. A short description of the system.
2. Relationship of software and system architecture

3. Inheritance/legacy issues
4. Business/application domain
5. Choice of technologies
6. Organization
7. Process issues
8. Resources in architectural activities
9. Merger of companies
10. Other issues

2.1 ABB Robotics

At ABB Robotics we interviewed a chief system architect who has one of the leading roles in architectural design of the two most recent system generations.

A short description of the system.

ABB Robotic is a manufacturer of Industrial robotic systems. ABB Robotics system has gone through 4 generations, and the 5th generation is on its way. Industrial robots are systems, which consist of one or more mechanical units (robot arms that can carry different tools), electrical motors, Robot Controller (computer hardware and software) and clients. Clients are used for on-line and off-line programming of the Robot Controller. ABB Robot Controller was initially designed in the beginning of the 1990-ties. The requirement was that the same controller should be used for all different types of robots, and thus the architecture is product line architecture. In essence, the controller has an object-oriented architecture and the implementation consists of approximately 2500 KLOC of C language source code divided in 400-500 classes and organized in 15 subsystems. The system consists of three computers that are tightly connected: a main computer that basically generates the path to follow, the axis computer, which controls each axis of the Manipulator, and finally the I/O computer, which interacts with external sensors and actuators.

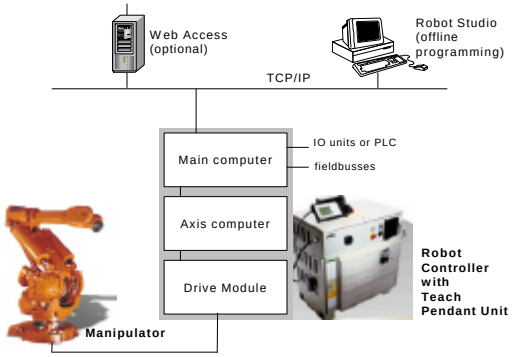

Figure. 1. System overview of an industrial robot system from ABB Robotics

The software platform of the robot controller defines infrastructure that provides basic services like: message-based inter-task communication, configuration support, persistent storage handling, system startup and shutdown, etc. These base services shape the implementation of the software system, as it is defined by the architecture.

Relationship of software and system architecture

When determining the *system architecture*, it is "system" that comes in the first place in the design. From system design, software and hardware design is determined. Software is becoming more important in the design and is closer connected to the system design, while the computer hardware can undergo more

changes while the software architecture has not changed. There is increased focus on software, to support: engineering, programming, commissioning and operation.
Requirements are written in a natural language. There are several levels of requirements and standard templates are defined for all of them.

Inheritance/legacy issues
When designing a new generation of the system, experiences with previous system generations were very important. However, other constraints related to the development organization environment have had very much impact on the architectural design, in a way that several of the top-level system subsystems were reused in the next system generation. To start design with a blank paper was not an acceptable option, because of unacceptable risk and associated cost.

Business/application domain
Standards have influence on the system architecture mostly on subsystem level. An example is safety, which is handled by a subsystem.
Type of customer – most of the systems are sold to automotive market. Because there are four major robotics manufacturers on the market, cost, product quality and new features that increase customer's productivity are extremely important.
Volumes and lifetime influence the "pre-study phase" when architecture is determined. Because the payoff expected for the effort invested is related to the volumes, more or less time can be spent on optimizations in the design phase.
NFR that were explicitly addressed in the architectural design were:

- Safety (human safety and safety of other robot environment)
- Easy to understand (architecture, development process)
- Reuse (design and code, process & tools)
- Adaptable/open
- Quality (reuse - subsystems change inside)

Regarding system *evolution*, new functionality has had more impact on the architecture than the non-functional requirements (NFR). Requirements for new functions are very related to decision to discontinue a system architecture. Together with the technology they have the most influence. It is not an individual requirement that leads to an architecture end, but a "stream" of new requirements that are costly to be met on the current architecture.

Choice of technologies
The preferred architecture is definitely influenced by the current *technology*, but it is not dictated by the technology. Technology gives a lot of inspiration for determining the architecture, as the architecture is likely to use or be prepared to use components of the shelf (both HW and SW). Important activities related to technology investigation for a new architecture are: Technology scouting, Prototyping and Education.

For the system *evolution* and *technology*, it is important to predict (guess) the variation points in the system - the subsystems that will likely change in the future and take this into account in the architecture. An example is UI – Teach Pendant Unit, that has changed from a terminal like device in generation 4, to a rich graphical UI client in generation 5.
New technology is constantly considered, in the sense that it is investigated if new technology can be used to improve system quality and increase development efficiency.

Organization
In the system architecture generation shift from 3rd to 4th generation, a *new organization* has been created to support the new architecture and development process. Generally speaking, o*rganization* changes definitely much more often than the architecture.

Distributed development did not have much impact, even though parts of the system are developed in geographically different locations. It is more important that many developers can give good contribution, without knowing and understanding the whole system.
Subcontracting had some limited influence on the architecture. Organizationally, a person with "proxy" role was responsible for coordination with each of the outsourced subsystems. Examples of third party components used in the system are: real-time OS, network communication software, field bus communication, UI-framework etc.
To balance *project and competence organization* internal organization with *subject areas* was created. Subject area is an owner of specified technical areas, responsible for design issues. Ownership of each system architectural component is explicitly assigned to a subject area. Subject area is also responsible for local planning, improvements and new solutions. It has been noticed that significant effort is required from subject areas to balance between: development of new functions, maintenance of previous releases and long-term quality improvements.

Process issues
Development process issues were not found to be much relevant in the architectural design. Many processes would be possible for the same architecture (or a domain of architectures). Processes and their changes are more related to organizational changes. In general, it can be said that the development process focus is on system integration.

Resources in architectural activities
S4 generation controller architectural design was done by a small core team of about five people, during the calendar period of about two years (total effort estimated to ~10my). This pre-study resulted in the selection of system architecture, hardware and software architecture, key technologies, tools and methods for the product development project.

Merger of companies
Not applicable.

Other issues
If a single the most important factor for in a design of an architecture needed to be selected, that would be to answer the question: where are the most changes/variations expected in the future.

Architecture is very tightly bound to the domain knowledge. Domain knowledge was described in the following way. It is not a single thing, but can rather be viewed as a combination of multiple things, among which, the following are very important:

- What will be the subject of changes in the system's environment
- What are the possible system configurations
- Different deployment scenarios
- Packaging
- Likely key scenarios for future requirements

2.2 Ericsson

This section contains summary of an interview with an architect in R&D System Management group, a group which is tasked with maintaining an overall technical view of how the Ericsson product portfolio and platform technology is evolving.
In this interview, the architect answered most of the questions from Ericsson system perspective, but some more general comments were also given using previous experiences with software architecture. The interviewed architect was the principal software architect of shipboard command and control system, which is described in [1].

A short description of the system.

In this interview, subject of discussion were telecom systems in general. These systems have a long history and have been standardized to enable a global communication system. Standards are the dominating factor for the systems functionality in this domain. It is a sign of the maturity of the telecom industry. The packet switching networks are more recent and these systems still experience more dynamic and changes in requirements. A telecom system has high demands on reliability and availability. For example, a switch should only be started once, meaning that almost all maintenance operations should be possible to do during operation. Further, interoperability is another key property of a telecom system. A telecom system is a complex system. An example of a telecom network is shown in figure 2.

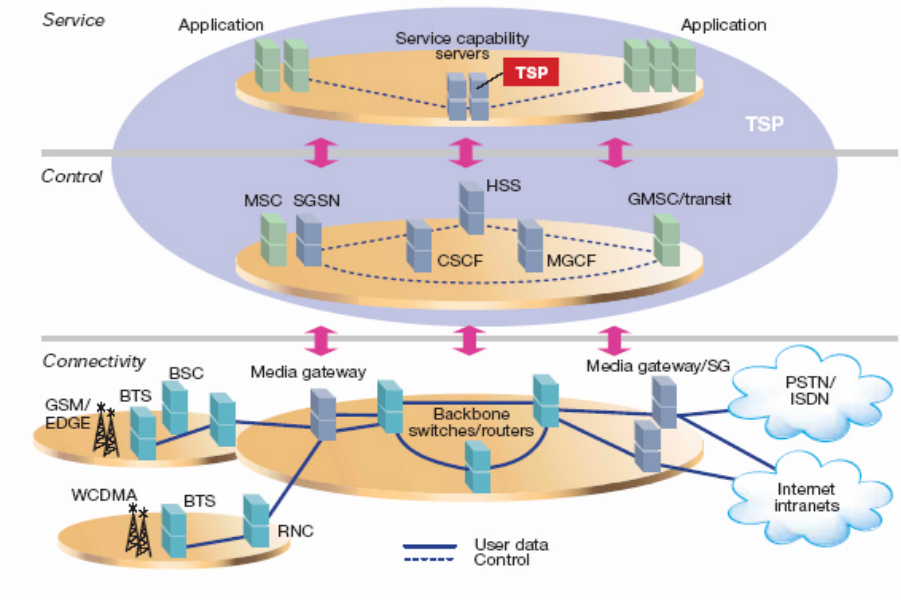

Figure 2. *Ericsson's horizontally layered network architecture* (source: Ericsson review no3, 2002 http://www.ericsson.com/about/publications/review/2002_03/files/2002032.pdf)

Relationship of software and system architecture

The *system architecture* dominates in a way that it optimizes the most important properties of the *system* as a whole. For example it defines how the system is built to enable efficient maintenance such as the possibility to change a board during operation. It also defines how a system can be configured, the scalability and requirements on reliability.

Hardware (HW) design follows system design and sets additional constraints for *Software (SW)* design. HW must be built so that it can easily be configured. SW design comes last. Special care needs to be taken for SW-HW version handling and their compatibility as HW is expected to change during the system lifetime.

Requirements are described in text written in natural language. The company has a well-established process for capturing and breaking down requirements from system level through nodes down to individual design items. This means that separation in HW and SW design happens quite late in the development. The architecture must be possible to describe in such a way that all involved parties understand the most important properties of the system, in particular for nodes built on newer technology. The well-known platforms (AXE being the dominant one) represent a stable architecture and can therefore expected to be known. .

Inheritance/legacy issues

In the creation of architecture, *experiences* are dominant. Reusing code and subsystem is something you expect in the architecture evolution, but not when you create a new architecture.

Business/application domain

Standards dominate the design in telecom domain. This is related to very high efforts required in interoperability testing.

Type of customer has very much influence on the architecture. The systems are deployed in over 140 countries around the world with many different operators. They set much more constraints than one would wish to have in the architecture design. "Just like builders of the systems have much opinion how the system should be used, the same way users (their technical representatives) have much opinions in how system should be built".

Volumes have high impact on architecture, especially on HW (cost reduction) but also on SW (troubleshooting and upgrades). Systems are by their purpose deployed geographically in a large area and troubleshooting a few systems compared to thousands makes a big difference.

Length of life is also important. In some cases systems will live that long that during their evolution HW will change (e.g. processor unavailability) OS will change etc. Layering approach is used to isolate the elements of the platform that may change in the future.

The most important *non functional* requirements are

- availability
- scalability (system must be configurable in SW and HW for different capacities according to different customer needs)
- performance, especially concurrency
- interoperability
- understandability (see organization section for clarification)

Choice of technologies

This is a difficult question to answer in general terms. Ideally, it would be just requirements on the system, that influence the creation of an architecture. In the practice, it is a mix of requirements and technology influences that together determine the system architecture.

When designing a new architecture it is important to not only "base" the system on the state of the art, since the system will live under such long time that it must be adaptable to new technologies.

Organization

Basic research in the company is conducted by the research group. The research is also the basis for influencing new standards. Product groups decide the architecture of different products.

In a large size company (more than 10 research and development centers) an architecture must take into account that implementation will take place in geographically different places.

This is related to the fact that organization sometimes mirrors in the architecture or vice versa.

When developing an architecture one has to take into consideration the size of the development organization that will be needed to implement the system. Partitioning of the system into subsystems for easy integration and verification needs to be done.

In such a large company as Ericsson organizational changes are much more frequent than architectural changes. Further, one should count for 5-10% new employees per year. Thus, the understandability of the system is an important factor in how quickly the newly employed people can be productive. An extreme example when *understandabilty* is of utmost importance is when the product responsibility for a specific product is organizationally transferred to a different group. Further, it is completely impossible to achieve the level of quality if the developers do not understand how the unit is used and operates with other units. All developers have to have the big picture.

Process issues

Platform and applications are developed in parallel but to avoid performance and quality problems several integrations are done during the development.

Resources in architectural activities

It is hard to say when architecture work is done because it is an iterative process that sometimes goes on late in the development cycle. The basic reason for that is that it is very hard to assure that all architecture relevant considerations have been taken care off in the early phase of the design. The initial architecture for successful systems is designed by a small group of people (<10), which preferably also builds the basic infrastructure (or at least are directly available during early implementation of key elements).

The initial development of the architecture is done by a small group of people which both understands the demands of the system and the technology. The success of architecture requires that the small group can provide a consistent architecture (single mind consistency). The small group has to have "correct" experience otherwise the architecture will not deliver what is required.

2.3 Tieto-Enator

This is interview summary with a senior specialist within the area of UMTS RBS software architecture at TietoEnator Telecom & Media.

A short description of the system

TietoEnator Telecom & Media develops 3G-base stations, in partnership with Ericsson, for the new mobile telecommunication systems UMTS. One of the most significant aspects of 3G base stations is that they are sold in very large volumes (each 3G provider in Sweden needs around 12000 base stations according to PTS estimations) and that they have very high requirements on availability and ease of maintenance.

Figure 3 shows the Radio Access Network in which the Radio Base Stations basically acts as the radio modem converting digital information to analog radio signals and vice versa.

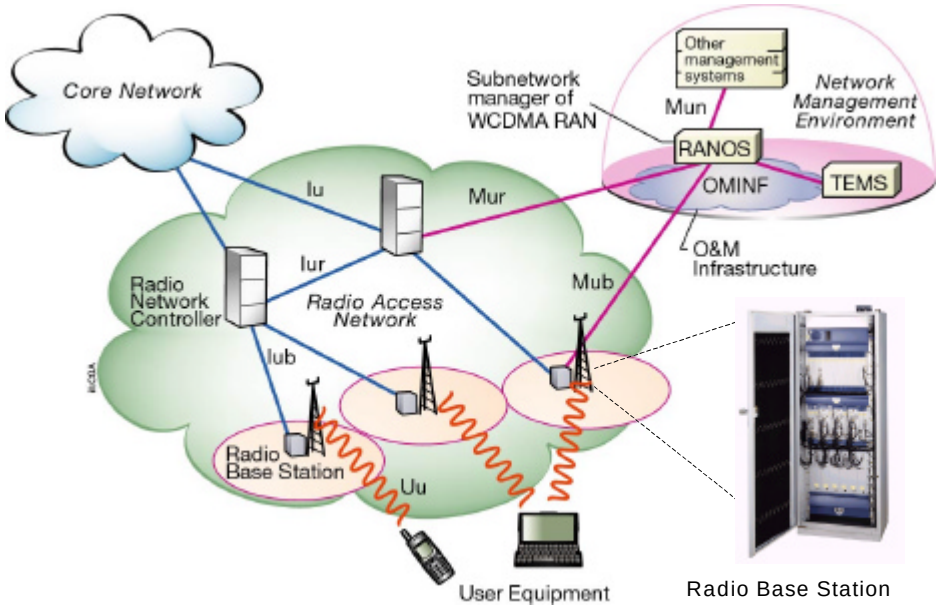

Figure 3 RBS in the Radio Access Network

The system currently being developed is the first version of the new 3G radio base stations. The development of the radio base station system was preceded by an initial prototype and experimental implementation, gathering experience and evaluating architectural solutions. In the actual product development, experience from the prototype development was reused but not the software.

Figure 4 shows the functional architecture and the major functional components of the Radio Base Station. The functional components can be realized both in SW and HW, where the Traffic Control and Operation & Maintenance are the most SW centric components.

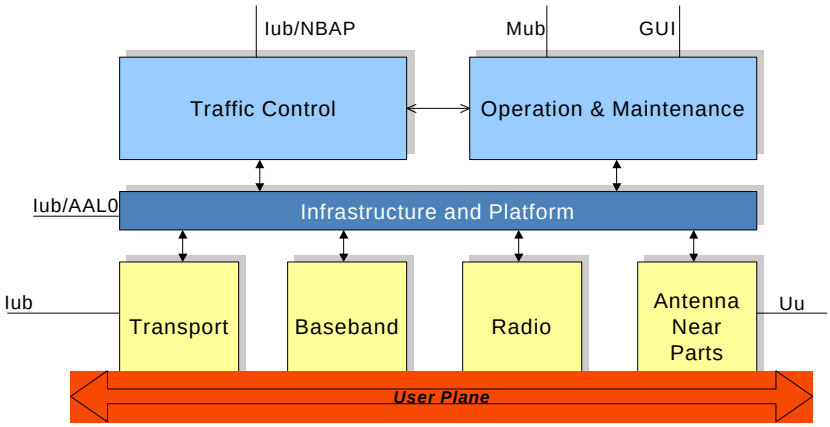

Figure 4 RBS Functional Architecture

The main characteristic of the SW architecture is that the functional architecture has been arranged in a layered structure. The layering can be seen in three distinct dimensions: From Traffic Control point of view, from Operation & Maintenance point of view and from Platform point of view. The main focus in the layered structure from the Traffic Control point of view is the hardware abstraction layer, which decouples the higher layers from the actual HW realization in the Transport, Baseband, Radio and Antenna Near Parts. The decoupling is achieved by having reusable components that provides the User Plane functionality in the same way independently of how the HW is being realized, e.g. depending of which kind of radio base station that is being built.

The radio base station Traffic Control software, consists of approximately 2000 KLOC of code organised in around in an object oriented UML-RT model with around 5000 model elements, i.e. capsules, protocols and classes. About 80% of the code is generated automatically from UML-RT-models. The base stations are built on a platform delivered by Ericsson (CPP, Connectivity Packet Platform). The platform is based on the OSE-Delta real-time operating system.

Relationship of software and system architecture

The *system architecture* in the base station is to a large extent given by the platform and by domain standards. The system architecture and hardware resources put restrictions on the software, but besides that, the *software architecture* is rather independent from the system architecture.

Packaging of software modules is important and in some sense independent from the architecture. The packaging structure defines an architecture in its own that is tailored for the structure of the development organization/project. Managing the dependencies among packages is important, e.g. minimize dependencies so that one software module is not dependent on another module that will be developed later on in a project. The relationship between the “packaging architecture” and the operational/runtime architecture shall be very loosely coupled. Of course there exists a relationship, but if the “packaging architecture” and the runtime architecture are too coupled, for example with a too strong one-to-one mapping, then you are heading for problems. A much greater freedom of mapping the “things” in the “packaging architecture” to entities and instances in the runtime architecture must exist.

Another important aspect of software architecture is to establish a common *vocabulary* and a terminology describing different parts and aspects of a system. A common vocabulary helps the introduction of a common system view, it makes communication between people easier, and it helps in preserving the architecture by making it visible. It is important to communicate principles and decisions that govern the architecture.

A special circumstance was that the prototype system and the platform were developed in parallel. Hence, there were problems with changes in the platform during prototype development. On the other hand, they were given possibilities to influence the platform during the prototype development, which were a clear benefit for the final product development.

The causality in architecture development was in the order; *system, followed by hardware, and finally software*. That is, the system architecture was provided initially and influenced the development of the hardware architecture. The system and hardware architecture then formed the base for the software development.

System requirements are today captured in ordinary documents with no specific notation. Functional requirements are specified in so called function specifications. These function specifications describing the black box view of the system are being transformed into a function description that further decomposes the requirements on the system into functional requirements on the subsystems. The main tool for this decomposition of functional requirements is done in sequence diagrams, where the derived requirements are allocated on the subsystems. Non-functional requirements are handled in characteristics requirement specifications, but are not completely allocated on all subsystems. Although some subsystems, especially subsystems containing HW and signal processing have a more strict allocation of both functional and non-functional requirements with requirement specifications on subsystem level. For the control software the main derived functional requirements are found in the sequence diagrams.

Architectural design is mainly communicated using design rule documents towards the subsystems. Currently investigations are conducted where a model based system engineering process are being looked into where more formal UML notation will be used to describe the system, both with respect to functional requirements as well as architectural design principles.

Inheritance/legacy issues

As this was the first version of the new 3G-system there was no legacy code in the application software, although the system platform from Ericsson was (re)used. Apart from this, the only legacy and reuse was in the form of experiences from building similar systems (GSM) and the preceding prototype. As stated before, there was no reuse of code from the prototype system.

Business/application domain

The traffic provided by the 3G systems is regulated by the 3G Partnership project (3GPP). Most functionality is defined in *standards* but there are many options in the standards with which companies compete. The main competing factors are characteristics (output power, capacity, size, weight and so on) as well as the maintainability of the base stations.

Aspects of the business domain that have heavy impact on production of this system are the *very large volumes* (each 3G provider in Sweden needs around 12000 base stations according to PTS estimations) and the high requirements on *availability* and *maintainability*. Maintenance operations on site are generally not acceptable due to the amount of base stations and their geographical locations (e.g., remote or located on roof tops).

Choice of technologies

The choice of the *technologies* CORBA, Java and code generation using UML-RT and RoseRT has influenced the software architecture to some extent. CORBA was chosen before it became a part of the standards for 3G. Both Java and RoseRT was chosen by TietoEnator to inspire the developers with

enthusiasm and to hopefully find implementation technologies that was more efficient during the development phase and that increased the quality of the code.
Future changes in the technologies are not probable and possible due to the large installed base of base stations out in the field .

Organization
Since the *organization* develops the architecture it has great influence on the architecture. One example is the partitioning of system functionality, where the organisational structure has to be considered in order to minimise dependencies and to get distinct interfaces between different organisational units. Another important aspect with respect to organisation and architecture is the ability to keep the architecture intact when the organization changes.

There are around 400-500 people involved in developing the entire 3G base station and about 60 to 70 of those were involved in developing the radio base station controller software. There have been no sub-contractors in building the system; since it is considered being too complex to outsource a part of the application.

Process issues
The *process* is rather vague and no specific architectural activities and no architect role is explicitly defined. Integrating software and hardware for the very first time is usually a source of friction, so also in this project. One potential remedy of this problem could be software/hardware co-design. There are ongoing activities that are investigating what can be improved in the process area, which currently is very vague and not very well documented.

Resources in architectural activities
There have been 5 to 6 architects involved in developing the radio base station control software. It should be noticed that the architect role have not been defined in all sub-systems and at all system levels. This and the fact that the organisation is very hardware oriented has been a source of some problems with respect to sub system interfacing.

The architects are situated within the functional organisation supporting the projects. It should be noted that the architect has no regulatory role, thus most control over the architecture is at design level.

One problem has also been the lack of continuity among the architects. Due to changes in the organisation architects have been replaced and thus all the knowledge that you cannot put on paper have vanished. This is a major concern regarding how to keep architectural principles throughout the lifetime of the product. There are always things that only exist in people's head, no matter how much you document. We see this as an emerging problem, especially in hard times when people get fired our outsourced.

Other
Q: If software architecture is rather independent from the system architecture, how much system and HW knowledge do you as a SW specialist need to do a good job? What do you find more valuable in your work - a good system/domain knowledge, or good general SW knowledge?
A: I would say that "good enough" knowledge, or at least understanding, of the system and the HW is needed. Especially in a basestation, which to a very large extent is a HW, problem, i.e. with respect to manage, supervise and control all the different HW that exist in the base-station. Personally, however, I find that my experience in SW engineering and good general SW knowledge is much more vital in getting a good SW architecture. I rely on others to have the specific system and domain knowledge (even though I consider myself to have a pretty good overview and basic understanding of it anyway).

2.4 Ericsson

This is a summary of interview with two software technology specialists at Ericsson Core unit Core Network Development.

A short description of the system

Ericsson Core unit Core Network Development develops platforms, which provide the basic hardware and software in different types of telecommunication systems. Examples of platforms are CPP platform, AXE platform, TSP platform and the WPP platform. Platforms include both hardware and software. Examples of systems built on these platforms are nodes in telecom networks:

- Digital switching system (e.g. AXE108)
- Mobile Base Station (e.g. RBS 3000-family)
- SGSN (GPRS Support Node)

Relationship of software and system architecture

The *system architecture* (where system is the telecom system, i.e. the complete telecom infrastructure) is the dominant parameter when designing the hardware architecture, which in turn dominates the software architectural design. The system architecture also influences the software architecture since it is a distributed system. An example of such an influence is software for handling data replication on the nodes.
Software architecture is described rather informally, as the rules and the principles for how software in the system shall be constructed, or "what is left when you take away all functions".
Requirements - The functionality in each node is defined by the system architecture. Because of strong standardization impact, there is little room for competition concerning functional features among the suppliers of telecommunication infrastructure. It is the non-functional properties that can constitute distinguishing features.
Requirements are described using natural text, in Word or Framemaker documents. Requirements tracing usually stops quite early. At least the programmer very seldom gets any input on the actual requirement that is behind a particular feature in his design description.

Legacy

Little or no legacy code is part of the development of a new product. To some extent a complete sub systems may be reused. The most important and substantial form of reuse is the reuse of experiences from developing similar systems.

Business/application domain

Standards have very big influence on the system architecture, which determines the top-level system architecture, i.e. which nodes should exist and their responsibilities. However, the architecture within nodes is determined by each manufacturer. No new products are released until there is a standard. Standards are usually finished several years before the first equipment that implements them are being sold. Consequently, it is important to participate in the standardization activities. The participation in standardization enables a high speed in product development since the company can influence the standards and knows what will be coming in the next release of the standard. Being able to influence the standards makes it possible to develop new products in parallel with the standardization activities, which is important in order to have products on the market early.
There exist *local standards* and regulations in some countries that the systems must be open for.

The *lifetimes* of the products are long, between 15-20 years. However, the long lifetime was not given explicit attentions when designing the architecture.
The most important *non-functional requirements* on the systems are: availability, maintainability, evolvability, and performance (especially concurrency).
The most important non-functional requirements on the software platform are: fault-tolerance (especially fault isolation) and concurrency. Support for event logging and event tracing is important. It is the core of the software platform that implements the support for non-functional attributes and prescribes the rules to be followed by application level code so that non-functional property goals are met.

Choice of technologies
The choices of different technologies have had very little influence on the architecture. The requirements on performance, concurrency, maintainability are the same independent of technology chosen. Some technologies makes it easier to implement these requirements, some make it harder. In the end, an architecture has to be designed, irrespective of what technology we have chosen. i.e. the technology has little influence on the architecture but it has big influence on the cost associated with implementing the architecture.

Organization
The *organization* influences the architecture. Functions are developed in different teams with their own special domain competences. The teams may be organizationally and geographically separated in the company. Each function has resulted in a sub-system in the architecture. However, an ideal architecture would have been slightly different.
Components, or self-contained blocks, have shown to have a positive impact on the *distributed development*, even though this was not explicitly taken into account when the architecture was created.
The distributed development is a problem when trying to implement an architecture for the first time
The *size* of the development organization, which is needed to implement the systems, is not taken into account in the architecture design, but there is a belief that it would be beneficial if it were.

Process issues
Generally, there is a conflict of interests between the projects and the architectural work Projects tend to have tight deadlines and thus, working on a short-term basis, while architectural related activities are strategic and long term.

Resources in architectural activities
Between 2-3 years for architectural work is not unusual at Ericsson Core unit Core Network Development. The architectural work includes prototyping and development of the basic principles and infrastructure. Moreover, the importance of having few key persons is stressed. This is very important in order to keep the system consistent and to make the most important and correct tradeoffs.
A system architect is defined to be a person who could implement the whole system by himself/herself, if time (to market) was not an issue. The architects may use the help from different quality experts for individual quality attributes, but it is the architect responsible for the managing the tradeoffs.

Merger of companies
N/A

Other
Interviewed software specialists do not work with colleagues HW specialists on frequent basis. But they stated that they needed a good knowledge of what processors were used, their properties and how their platform multiprocessor systems were built. Those things have impact on how the whole system and software will function.
There are other SW specialists that work with HW close design, e.g. FPGA, ASICS, DSP, NSP, microcontrollers etc. In that kind of SW role, deep HW knowledge is a must.

2.5 Volvo Construction Equipment (VCE)
The interview was done with a technical specialist in electronic systems, who is involved in the design of both hardware and software systems for on-board use.

A short description of the system

Volvo Construction Equipment (VCE) develops and manufactures a wide variety of construction equipment vehicles, such as articulated haulers, excavators, graders, backhoe loaders, and wheel loaders. Some examples of the vehicles are shown in figure 6.

Figure 6*. Examples of construction equipment developed by Volvo CE*

The products are to be used at construction sites, and the most important aspect of the vehicle is to provide a reliable machine to increase production. In-vehicle electronic systems and networks are an important part of the construction equipment product and are crucial to provide end-user functionality, such as automatic gearbox, engine and differential lock control, as well as providing diagnostic and service functions.

The vehicle control system developed by VCE consists of several distributed nodes, electronic control units (ECUs), connected to two busses. One bus is the SAE J1939, CAN, bus and the other is a SAE J1587 bus. This is illustrated in figure 7.

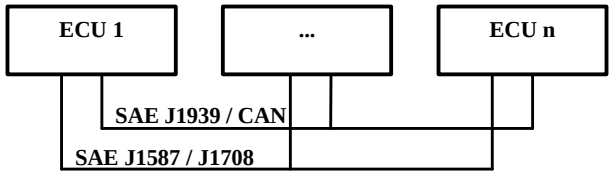

Figure 7 Components in electronic system of Volvo CE equipment.

Each ECU consists of hardware (16-bit CPU at 33 MHz) and two conceptually different software layers. The top layer is an application layer with specific functionality for different ECUs and products. The lower layer is the interface between the application and the hardware. In this interface layer, the following can be found: communication layer, which handles communication between different nodes, the I/O layer that handles the in and output on the affected node and the real-time operating system Rubus, which handles task scheduling and synchronization.

The system has gone through three generations:

1. The first generation system consisted of 1 ECU and the software was developed by one person and the code size was about 10KLOC.
2. For the second generation the corresponding numbers are five person and 50KLOC and
3. The third generation was developed by 35 persons and contains about 140KLOC. From the first system that mainly controlled the automatic gearbox, a number of control and diagnostic functions have been added. Examples are differential lock control, retarder control, and test functions accessible from service tools (external computers).

Relationship of software and system architecture

There are many changes within every generation of equipment and generations of the control system follow generations of the products they are a part of. The Rubus component concept is used to isolate the SW from the HW. The complexity of HW is relatively simple compared to SW, which contains the most logic. Investment in SW has become more important to preserve than the investment in HW. Goals are to decouple HW from SW evolution. Also *easy partitioning* of software functions is an important goal because VCE like other vehicle developers work with requirements on variant design (many different variants of a vehicle model).
Requirements are propagated from system architecture down to the control system in form of text written in natural language. In the past, it was common to "work closer to the system" (the construction equipment system), but it is getting more and more common to work from the requirement specification.

Inheritance/legacy issues
In evolution of system between system generations I, II and III, was it not important to keep any of the existing code or HW. Neither HW nor SW had any requirements of reuse between generations. Complete HW and SW were redesigned. We expect that this will change. As SW becomes more complex and costly, it will have to be partly reused (evolved rather than revolutionized.)
The trend is to specify rather than design HW. This means that the designers of the system (VolvoCE) specifies how many I/O and of what type, how much flash memory, E2, RAM etc, but board and circuits are chosen by subcontractors. Cost wise, HW could be revolutionized between system generations, but again its relation to the increasingly costly SW yields that we expect evolution.

Business/application domain
Standards do not have yet big impact, but that is changing (ISO 15998, ISO 1508). Coming standards set requirements on the whole development chain.
Types of customers are described more under the *organization* section further in this text.
Volumes have big impact on HW design (cost reduction) and very little impact on SW.
Lifetime issues are very important. About 80% of the systems are estimated to still be in use after 20 years. As the control system value in the value of the product is only about 1%, customers count on beeing able to replace broken ECUs even after 20 years. If HW is going to be replaced this also means that there has to be SW that can run on this new HW, so this also becomes a SW issue.
Non-functional requirements that are important and addressed explicitly are:

- safety
- reliability
- real-time requirements

Choice of technologies
The Rubus component concept is central *technology* in this system. Architecture of the system is dominated by this technology. There are very positive experiences since introduction of this technology in generation 3 of the product. The software architecture of the ECUs is defined in a graphical editor where the in- and out-ports of the different tasks are connected with other components in a graphical editor which then calculates real-time attributes of the tasks and generates code.
The following can be defined in the tool: precedence, communication, and real-time attributes (deadline, period, release time, WCET). Rubus operating system is used, that supports hard and soft real-time tasks. A tool named Rubus Configuration Compiler, CC, supports communication, synchronization, and temporal constructs as described earlier. A scheduler that produces an off-line schedule and communication infrastructure for the operational mode uses the configuration data for the tasks. If the temporal properties of the task set cannot be fulfilled, the scheduler rejects the configuration. Static scheduling is used within Rubus, and the execution times of the tasks are monitored for later analysis.
The base platform, i.e. hardware and non-application software, are reused on all vehicle types, but the number of ECUs and the application layer are different.
The use of standard technologies becomes more and more common.

Organization

Organizationally, Volvo CE is divided into a number of product companies and other supportive companies. The product companies focus on their specific products and are responsible for cost, deliveries, service etc to the end-customer. End-customers range from single vehicle owners to rental companies with hundreds of vehicles. Typically, a product company manufactures a product line of similar, but differently sized vehicles.
The development department holds expertise on and develops on-vehicle electronic systems for various products to the product companies. The electronic development includes control systems, other on-vehicle software, and hardware. The product companies can also decide to buy electronics development from vendors external to Volvo CE. Hence, the final products can include hardware and software components from many vendors both from Volvo CE and external to Volvo CE. From the product companies' perspective, Volvo CE electronic development is essentially equal to any subsystem vendor and must develop control systems and software at competitive prices.

Process issues
As number of people grows in the development organization, the need for having a good development *process* has increased. Most of the development time is spent on the development of SW.
Software development at VCE has not yet been affected by standards, but that is changing. Coming standards (ISO 15998, ISO 1508) set requirements on the whole development chain.

Resources in architectural activities

In the early design of e.g. most recent generation of the system only few people were involved (<5). These people were responsible for the system design and also for the implementation of the base platform framework. More people were added to the project when application specific work started and basic principles were in place. The initial work was not considered to be a pilot-project, but was a part of the product development with a scheduled release date for the product/system (i.e. the construction equipment).

HW and SW are developed in the same group; with the difference that HW is only designed but not implemented "in the house". In terms of design responsibility, there are the following roles: HW design responsible, SW platform design responsible, one per SW application responsible, and responsible for SW tools.

Merger of companies
N/A

Other issues
.

2.6 Volvo Car Corporation

This section contains summary of an interview with a "Program Manager, Research & Advanced Engineering" at "Electrical and Electronic Systems Engineering".

A short description of the system
Volvo Cars produces several families of cars, each family consisting of several models and many variants. Due to different laws in different countries many different variants are delivered to different countries. Volvo Cars is a typical company that produces a large amount of products where efficiency, production costs and production quality plays a dominant role. The electronic control system is usually the same in different models of the same family. It is important that Volvo Cars does not develop and does not produce components, neither physical units nor software embedded in them, but Volvo does design the control system that determines how components are integrated and work together.

An example of a contemporary car electronic architecture is that of the Volvo XC90 (see Figure xxx). The boxes in the figure represent electronic control units (ECUs) and the lines represent communication buses. The maximum configuration contains about 40 ECUs. They are connected by two CAN networks; one for powertrain and one for body functionality. From some of the nodes, LIN sub networks are used to connect slave nodes into a subsystem. The other main structure is a MOST (MOST Specification Framework, 1999) ring, connecting the infotainment nodes together, with a gateway to the CAN network for limited data exchange. Through this separation, the critical powertrain functions on the CAN network are protected from possible disturbances from the infotainment system. The diagnostics access to the entire car is via a single connection to one ECU. The figure shows approximately how the ECUs are placed in various locations in the car. The partitioning of functionality is decided by the location of the sensors and actuators used, but also by the combinations of optional variants that are possible.

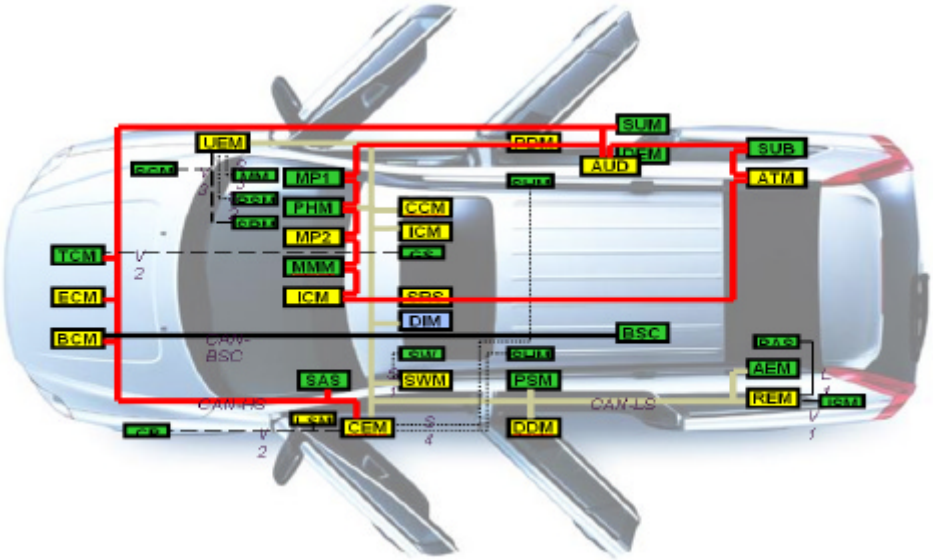

Figure. 8. *The electronic architecture of Volvo XC90.*

The software architecture of a computer node has a layered style with several basic layers:

- application layer,
- diagnostic and network communication layer, and
- the lowest layer with RTOS and I/O modules.

Diagnostic kernels provide an implementation of the diagnostic services that each node must implement to act as a client towards the off-board diagnostic tool.

Network communication software provides a layer between the hardware and the application software, so that communication can be described at a high level of abstraction in the application, regardless of the low-level mechanisms employed to send data between the nodes. *Volcano* is a communication concept used throughout the Volvo Car Corporation for managing network traffic. Currently Volcano supports the CAN and LIN busses. The basic concept in Volcano for communication between software components is signals, where a signal typically represents some engineering data. The Volcano concept also addresses vehicle manufacturer controlled integration of components developed by suppliers. This is done through the use of the Volcano API and by separate specification of the signals used by a component and the network configuration.

Relationship of software and system architecture

The focus of Volvo Cars electronic *architecture* effort is mainly concerned with assuring system properties including scalability to support product variation, reusability and partitioning of SW components to lower development cost, as well as safety and reliability. Software is an integrated part of hardware and it is treated as a part (identified with a part number) of hardware – for example a part of an ECU (electronic control units). Volvo Cars does not specify and does not develop software or hardware of ECUs, but rather focuses on system level requirements and specification.

Requirements are described and communicated by using natural text and UML (class diagrams, state charts). There are no industry standards on how to specify functionality delivered by suppliers.
Vulcano technology plays an important role here in a way that ECU suppliers have to document which signals the unit communicates with, what are the timing requirements, etc using a special format. Volvo puts all this information together in a signal database and verifies that all requirements can be met in within the limitation of the bus bandwidth. Also a SW module is generated that converts CAN messages to signals.

Inheritance/legacy issues
The compatibility requirements spread through a family of products, and are not requirements for new generation of products. The expected lifetime of a car is more than 25 years; during that period the company guarantees the support and supply of spare parts. This leads to a development that is very much of evolutionary type. The company does not perceive serious problems with legacy systems due to extensive outsourcing.

Business/application domain
Standards influence on the architecture of the control system architecture is considered to be rather low.
A large number of cars (in order of 400.000) are produced each year and are distributed in many countries. *Volumes* play rather important role in relationships with sub-suppliers.
The business vision of Volvo Cars is to produce cars for modern families, which implies safety, care for environment and quality. In the control system design, safety, reliability are important for both HW and SW. For HW, there are many other *non-functional requirements*, related to its operational requirements (e.g temperature, humidity, vibrations, etc).

Choice of technologies
There are many constraints, which determine the use of *technology*. The technology is much related to the systems characteristics and system constraints – such as space, temperature and other environmental conditions. Further, different safety and environmental requirements, often regulated by local authorities, determine the choice of technology. In that sense software technology is of minor importance for the company. The basic requirement is that well-known and proved technologies are used.
One technology that has significant impact on the control system is Volcano (http://www.volcanoautomotive.com/).

Organization
The control system is designed by Volvo in cooperation with significant sub-contractors. After the system is designed, all nodes in the control systems are developed by sub-contractors.

Process issues
Volvo Cars development and production process is integration-oriented. The company does not develop the components but integrate them in the final production process. For this reasons the process is very much component-based. This approach enables better flexibility and time-to market. Versions and variant management is rigorous. A basic configuration item is identified by an article number. Introduction of new article numbers are very strict, which implies high costs for introduction of new components.
A development process of a completely new model (or family) is a long process divided in several phases: (i) Preparation phase in which the requirements are captured and elicited. (ii) The conceptual development phase in which the concepts are developed on prototypes and (iii) Production development that includes

preparation for the production. The vendors of the components usually participate in the prototype development.

Integration process - In order to leverage control in integrating supplier software and hardware components, VCC uses methods and tools to assist in this effort. The suppliers are developing the software using whatever tools and structure, but VCC as an OEM specifies communication, power consumption, diagnostics, and software download procedure. The volcano concept allows VCC to keep a list of all signals on the network and give suppliers precise specifications on which signals to use and how much bandwidth to use. Also, worst case timing is predicted by the volcano tools. Power consumption of ECUs is specified for the different states of the system e.g. key positions together with behavior in cases of voltage drop such as crank. Moreover, VCC gives specifications on how diagnostic functions should behave and how software is to be downloaded to the ECU.

Even though there are no industry standards on how to specify functionality delivered by suppliers, there are established procedures that work well for suppliers of ECUs. The ability to influence suppliers is heavily dependent of product volume, why car OEMs might have to adapt to the ways of the largest OEMs. Car industry is aware that chips will not be available and that the 15-year support of products is unclear. A way to address this by trying hard to keep number of article numbers and configurations low. OEMs who succeed in having less complex maintenance and refitting of electronics will have a great business advantage.

Resources in architectural activities

Volvo Cars primarily considers the system requirements, system specification and system architecture. System architects constitute a relatively large team focused on different domains. The architect's most intensive activities are related to evaluation – evaluation of different ideas; their suitability for new products, the constraints and the costs.

The basic architecture of cars are changed slowly – mostly when new models are developed.

Merger of companies

Volvo Cars has relatively recently become a part of Ford company. The company has not seen significant changes in their development and production process, although there are many activities related to coordination of development of different Ford companies.

Other

Generally speaking, those who work with architecture are experienced people that understand well both HW and SW. Equally important for them is to understand the car industry domain and its prerequisites. Additionally, there are architects that focus on different aspects of the system, like: mechanics and cabling, electronics, software (i.e. the structure of SW in ECUs), but a wide general knowledge is expected from them as well.

2.7 Bombardier Transportation

At Bombardier Transportation, we had two interviews; one interview with a system architect and another with a technology specialist.

A short description of the system.

Bombardier Transportation is the global leader in the rail equipment, manufacturing and servicing industry (http://www.bombardier.com). Examples of products are: passenger rail vehicles and total transit systems, locomotives, freight cars, propulsion & controls, signaling equipment and systems. The development site in Sweden, which we have visited, was previously a part of Adtranz, which acquired it from ABB.

The role of the development group, in which the interviewees work, is shown in Figure 9. This group develops:

- components (of the control system),
- propulsion and
- control systems

and delivers them to the system groups delivering complete solutions to the end customer, and in particular sub-domains.

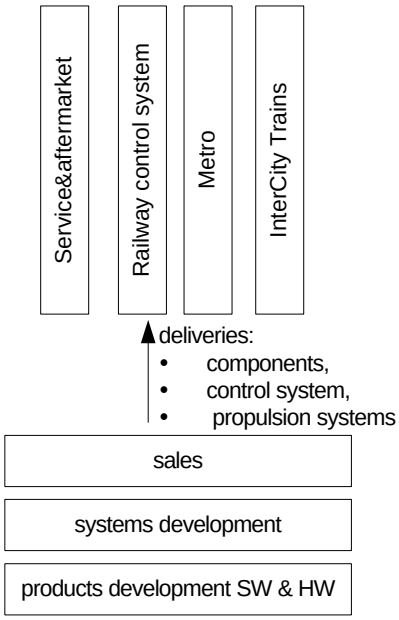

Figure 9. Products and systems of the interviewed group

A very simple model of a train with the most important elements of its control system is shown in figure 2. Standard called TCN (Train Communication Network) defines network interconnections between vehicles and within vehicles. The train bus is called WTB (wired train bus). A vehicle bus connecting devices within a vehicle is called MVB (Multifunction Vehicle Bus) and is also defined by the TCN architecture. A common time-triggered protocol is used on both WTB and MVB bus.

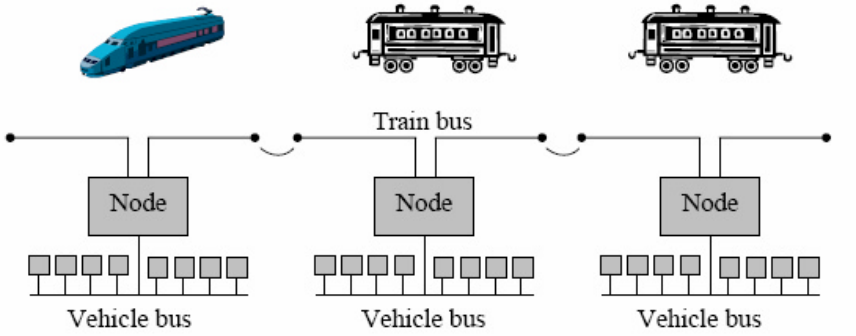

Figure 10. *Train communication network*

Within nodes of the control system, Bombardier uses programming languages defined by the standard IEC 61131-3. The standard defines five languages, and out of those *function blocks* (FCB) is the mostly used. Apart from this traditional control system related functionality, comfort related functionality is also getting more and more focus. So, each of the computer based electronic equipment that is developed by this organization, can be classified into one of the three "domains":

- Comfort (e.g. connection system, passenger information, entertainment, etc)
- Safety (control system that is a part of the safety control)

- Control systems

Relationship of software and system architecture

System architecture is definitely the most important in the architectural design. The impact of computer HW on the architectural design used to be much higher, but this is changing in such a way that SW design is becoming more important than HW (related to increasing SW functionality and cost). Design of SW and HW follows each other, and it is hard to say which one precedes the other. Because of the fact that the HW is subject of degradation faults, and operational environment condition vary a lot (e.g. temperature of environment may be –40C or 60C; ….), it seems that more attention is needed to be devoted to HW than SW, in order to achieve satisfactory system properties. There are examples where it was very important to make new HW compatible to run multiple old SW. For example, this influenced choice of e.g. CPU for new HW.

Requirements are described in natural language, in text and drawings. No particular standard notations are used for describing system models. Common standardized templates are used for requirements specifications.

Inheritance/legacy issues

Experiences are the most important in architectural design but it is hard to give any general answers – answers may differ from case to case. There is a case that new HW architecture was developed with requirements to be able to run old SW. This is related to long product lifetime.

There is a wish to reuse code for reliability and safety reasons.

IEC-1131 provide a good platform for reuse but to be really efficient it must be incorporated in a product line architecture philosophy.

Business/application domain

Standards have rather significant impact, e.g.:

- TCN fieldbus for trains is standardized
- UIC standard that defines how wagons are assembled in a train
- Safety standards
- For HW different standards (e.g. vibrations, temperature etc)

Types of customers can be grouped in domains. Domains have different requirements on the system, e.g. requirements from ICT (InterCity Trains) differ significantly from Metro (ICT is much more advanced and demanding). A intention is to do less development explicitly for one customer order, otherwise there can be a conflict with the desire to develop and maintain a common platform base for the product line.

Volumes for a product are explicitly considered – it is a part of a business case study before each project enters the development phase, e.g. volumes are important in the sense that they are related to the cost of licenses.

Length of life is explicitly considered both for HW and SW. For SW, there are examples where Linux was preferred than Windows, because it was easier to plan long term maintenance (10-20 years) if the company could own the source code. Systems (products in use) life-time is expected to be about 20-30 years.

NFR (non-functional requirements):

- Safety – more focus on process than product
- Environmental requirements (e.g. temperature, moisture)
- Maintainability – e.g. a requirements that HW can easily be changed; requirement that SW update can be done easily (and remotely) by non-experts. There is one new trend in deals with customers – customers by the system and *maintenance package*, such that lifecycle cost is calculated in the deal. The importance of non-functional system properties then obviously becomes a first-concern issue for the manufacturer.

Choice of technologies

It is hard to give a general answer. There are influences in both directions. E.g. commercial OS are used in the products, but system quality requirements and length of life of the products, have dominating role when choices are made between different alternatives.

It is expensive to certify HW and SW (safety related) for trains.

There is a trend to replace field-busses with more other industrial standards, like Ethernet, TCP/IP.

Layered architecture and OS isolation layer is used to preserve investment and guard from future OS changes.

Requirements for hardware changes (for example monitors) may imply changes in technology, including software technology.

In the terms of new technology in software, the company does not position itself as the leader, but a follower (compared to e.g. other pure SW companies). This reduces product development costs.

SW cost as a part of the product cost is increasing, and therefore SW is getting more and more attention; e.g. diagnostics functionality is growing.

There are positive initial examples (project finished before time and under its budget) of using UML for modeling of application design, instead of IEC 61131 function blocks. But, what makes a technology more superior to another is very relative depending on the previous experiences of its users (in this case application developers), costs for training and education etc.

Future HW changes are explicitly considered in system design.

Organization

Company organization mirrors the system architecture, but organizational changes are more often than architectural. There are cases (related to previous mergers) that responsibility for development of a subsystem of product is moved from one country to another.

Experiences with third party SW components can be summarized as: it works well to order development of system components from a reliable subcontractor, but experiences with using black-box ready made third party components are mixed.

CMM level 2 is a goal.

Process issues

Waterfall model is used.

Because of safety requirements, there are differences between the domains a subsystem is a part of, with three domains being recognized: Comfort, Safety, Control. Processes for development of safety related products are much more strict and regulated, than comfort domain.

Resources in architectural activities

Design of a new generation control system can be estimated to about 3-4 year time with about 25 people involved (initially fewer). The next step towards product development involves more people and implementation of smaller functions.

Other issues

There is an example where UML was tried to be used to model all elements of the control system, but was not successful. There is a limited scale success of using UML on application level development, instead of using 1131 languages.

3. Interviewers comments about the interviews

What do we think about the interviews?

- They were a good way to meet knowledgeable people with many years of practice and also learn about the state of the practice in different domains.
- Very interesting, but it is hard to guide the interviews so that the essentials are captured.

Was the time too short/long?

- Several of the interviews were actually longer than two hours. The time varied because of different sizes and complexity of the systems investigated.
- The time was appropriate

Were the questions appropriate?

- Yes, even though some of the questions should have been made more clear even to us.
- The questions provided a good base, although the interview took the form of a discussion (which is natural and desired).

Was it good to have a list of questions?

- Yes, they were used to guide the discussions and as a checklist.
- Yes – to keep the discussion on track

What problems did you notice during the interviews?

- Quite a lot of time was spend on getting a domain overview and get familiar with terminology of the domain.
- It is hard to capture all relevant information

Suggestions for improvement, if we were to do the interviews again.

- If possible a material should be obtained in advance, to educate the interviewers about the basics of the domain, its terminology. Also a short questionnaire (5-6 simple questions – one sentence answer) could have been sent to the interviewed, to learn about their role and responsibility in the system.
- Record the interviews?
- Maybe one could provide a more clearly defined goal with the interview

4. Summary and future work

In this paper, we document the data collected as a part of the case study of Software Architecture. The setup of the case study is described and the questions used to conduct the interview. A summary of the each interview is given in a subsection of section 2, without any comparison of the results between different interviews. Finally, we summarize our own experiences about the interviews that can be used to improve the case study if a similar case study is going to be done in the future.
Our future work will be the analysis of the case study data and organization of a workshop where results of the analysis will be discussed with the architects interviewed.

5. References

[1] Bass L., Clements P., and Kazman R., *Software Architecture in Practice*, Addison-Wesley, 2003.

[2] Pfleeger S.L., *Software Engineering - Theory and Practice*, Prentice Hall, 2000.