Industrial Requirements on Component Technologies for Embedded Systems*

Anders Möller[1,2], Joakim Fröberg[1,3], and Mikael Nolin[1]

[1] Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Box 883, SE–721 23, Västerås, Sweden
[2] CC Systems, www.cc-systems.com
[3] Volvo Construction Equipment, www.volvo.com
{anders.moller, joakim.froberg, mikael.nolin}@mdh.se

Abstract. Software component technologies have not yet been generally accepted by embedded-systems industries. In order to better understand why this is the case, we present a set of requirements, based on industrial needs, that are deemed decisive for introducing a component technology. The requirements we present can be used to evaluate existing component technologies before introducing them in an industrial context. They can also be used to guide modifications and/or extensions to component technologies, to make them better suited for industrial deployment. One of our findings is that a major source of requirements is non-technical in its nature. For a component technology to become a viable solution in an industrial context, its impact on the overall development process needs to be addressed. This includes issues like component life-cycle management, and support for the ability to gradually migrate into the new technology.

1 Introduction

During the last decade, Component-Based Software Engineering (CBSE) for embedded systems has received a large amount of attention, especially in the software engineering research community. In the office/Internet area CBSE has had tremendous impact, and today components are downloaded and on the fly integrated into, e.g., word processors and web browsers. In industry however, CBSE is still, to a large extent, envisioned as a promising future technology to meet industry specific demands on improved quality and lowered cost, by facilitating software reuse, efficient software development, and more reliable software systems [1].

CBSE has not yet been generally accepted by embedded-system developers. They are in fact, to a large extent, still using monolithic and platform dependent software development techniques, in spite of the fact that this make software systems difficult to maintain, upgrade, and modify. One of the reasons for this

* This work is supported by Volvo Construction Equipment, CC Systems, and KKS (The Knowledge Foundation), within the project HEAVE.

status quo is that there are significant risks and costs associated with the adoption of a new development technique. These risks must be carefully evaluated and managed before adopting a new development process.

The main contribution of this paper is that it straightens out some of the question-marks regarding actual industrial requirements placed on a component technology. We describe the requirements on a component technology as elicited from two companies in the business segment of heavy vehicles. Many of the requirements are general for the automotive industry, or even larger parts of the embedded systems market (specifically segments that deal with issues about distributed real-time control in safety-critical environments), but there are also some issues that are specific for the business segment of heavy vehicles.

The list of requirements can be used to evaluate existing component technologies before introducing them in an industrial context, therefore minimising the risk when introducing a new development process. Thus, this study can help companies to take the step into tomorrow's technology today. The list can also be used to guide modifications and/or extensions to component technologies, to make them better suited for industrial deployment within embedded system companies. Our list of requirements also illustrates how industrial requirements on products and product development impact requirements on a component technology.

This paper extends previous work, studying the requirements for component technologies, in that the results are not only based on our experience, or experience from a single company [2, 3]. We base most of our results on interviews with senior technical staff at the two companies involved in this paper, but we have also conducted interviews with technical staff at other companies. Furthermore, since the embedded systems market is so diversified, we have limited our study to applications for distributed embedded real-time control in safety-critical environments, specifically studying companies within the heavy vehicles market segment. This gives our results higher validity, for this class of applications, than do more general studies of requirements in the embedded systems market [4].

2 Introducing CBSE in the Vehicular Industry

Component-Based Software Engineering arouses interest and curiosity in industry. This is mainly due to the enhanced development process and the improved ability to reuse software, offered by CBSE. Also, the increased possibility to predict the time needed to complete a software development project, due to the fact that the assignments can be divided into smaller and more easily defined tasks, is seen as a driver for CBSE.

CBSE can be approached from two, conceptually different, points of view; distinguished by whether the components are (1) used as a design philosophy independent from any concern for reusing existing components, or (2) seen as reusable off-the-shelf building blocks used to design and implement a component-based system [5]. When talking to industrial software developers with experience from using a CBSE development process [6], such as Volvo Construction Equipment[1], the first part, (1), is often seen as the most important advantage. Their experience is that the design philosophy of CBSE gives rise to good software architecture and significantly enhanced ability to divide the software in small, clearly-defined, development subprojects. This, in turn, gives predictable development times and shortens the time-to-market. The second part, (2), are by these companies often seen as less important, and the main reason for this is that experience shows that most approaches to large scale software reuse is associated with major risks and high initial costs. Rather few companies are willing to take these initial costs and risks since it is difficult to guarantee that money is saved in the end.

On the other hand, when talking to companies with less, or no, experience from component-based technologies, (2) is seen as the most important motivation to consider CBSE. This discrepancy between companies with and without CBSE experience is striking.

However, changing the software development process to using CBSE does not only have advantages. Especially in the short term perspective, introducing CBSE represents significant costs and risks. For instance, designing software to allow reuse requires (sometimes significantly) higher effort than does designing for a single application [7]. For resource constrained systems, design for reuse is even more challenging, since what are the most critical resources may wary from system to system (e.g. memory or CPU-load). Furthermore, a component designed for reuse may exhibit an overly rich interface and an associated overly complex and resource consuming implementation. Hence, designing for reuse in resource constrained environments requires significant knowledge not only about functional requirements, but also about non-functional requirements. These problems may limit the possibilities of reuse, even when using CBSE.

With any software engineering task, having a clear and complete understanding of the software requirements is paramount. However, practice shows that a major source of software errors comes from erroneous, or incomplete, specifications [7]. Often incomplete specifications are compensated for by engineers having good domain knowledge, hence having knowledge of implicit requirements. However, when using a CBSE approach, one driving idea is that each component should be fully specified and understandable by its interface. Hence, the use of implicit domain knowledge not documented in the interface may hinder reuse of components. Also, division of labour into smaller projects focusing on single components, require good specifications of what interfaces to implement and any constraints on how that implementation is done, further disabling use of implicit domain knowledge. Hence, to fully utilise the benefits of CBSE, a software engineering process that do not rely on engineers' implicit domain knowledge need to be established.

Also, when introducing reuse of components across multiple products and/or product families, issues about component management arise. In essence, each component has its own product life-cycle that needs to be managed. This includes version and variant management, keeping track of which versions and

[1] Volvo Construction Equipment, Home Page: http://www.volvo.com

variants is used in what products, and how component modifications should be propagated to different version and variants. Components need to be maintained, as other products, during their life cycle. This maintenance needs to be done in a controlled fashion, in order not to interfere aversively with ongoing projects using the components. This can only be achieved using adequate tools and processes for version and variant management.

3 A Component Technology for Heavy Vehicles

Existing component technologies are in general not applicable to embedded computer systems, since they do not consider aspects such as safety, timing, and memory consumption that are crucial for many embedded systems [8, 9]. Some attempts have been made to adapt component technologies to embedded systems, like, e.g., MinimumCORBA [10]. However, these adaptations have not been generally accepted in the embedded system segments. The reason for this is mainly due to the diversified nature of the embedded systems domain. Different market segments have different requirements on a component technology, and often, these requirements are not fulfilled simply by stripping down existing component technologies; e.g. MinimumCORBA requires less memory then does CORBA, however, the need to statically predict memory usage is not addressed.

It is important to keep in mind that the embedded systems market is extremely diversified in terms of requirements placed on the software. For instance, it is obvious that software requirements for consumer products, telecom switches, and avionics are quite different. Hence, we will focus on one single market segment: the segment of heavy vehicles, including, e.g., wheel loaders and forest harvesters. It is important to realise that the development and evaluation of a component technology is substantially simplified by focusing on a specific market segment. Within this market segment, the conditions for software development should be similar enough to allow a lightweight and efficient component technology to be established [11].

3.1 The Business Segment of Heavy Vehicles

Developers of heavy vehicles faces a situation of (1) high demands on reliability, (2) requirements on low product cost, and (3) supporting many configurations, variants and suppliers. Computers offer the performance needed for the functions requested in a modern vehicle, but at the same time vehicle reliability must not suffer. Computers and software add new sources of failures and, unfortunately, computer engineering is less mature than many other fields in vehicle development and can cause lessened product reliability. This yields a strong focus on the ability to model, predict, and verify computer functionality.

At the same time, the product cost for volume products must be kept low. Thus, there is a need to include a minimum of hardware resources in a product (only as much resources as the software really needs). The stringent cost requirements also drive vehicle developers to integrate low cost components from suppliers rather than develop in-house. On top of these demands on reliability and low cost, vehicle manufacturers make frequent use of product variants to satisfy larger groups of customers and thereby increase market share and product volume.

In order to accommodate (1)-(3), as well as an increasing number of features and functions, the electronic system of a modern vehicle is a complex construction which comprise electronic and software components from many vendors and that exists in numerous configurations and variants.

The situation described cause challenges with respect to verification and maintenance of these variants, and integration of components into a system. Using software components, and a CBSE approach, is seen as a promising way to address challenges in product development, including integration, flexible configuration, as well as good reliability predictions, scalability, software reuse, and fast development. Further, the concept of components is widely used in the vehicular industry today. Using components in software would be an extension of the industry's current procedures, where the products today are associated with the components that constitute the particular vehicle configuration.

What distinguishes the segment of heavy vehicles in the automotive industry is that the product volumes are typically lower than that of, e.g., trucks or passenger cars. Also the customers tend to be more demanding with respect to technical specifications such as engine torque, payload etc, and less demanding with respect to style. This causes a lower emphasis on product cost and optimisation of hardware than in the automotive industry in general. The lower volumes also make the manufacturers more willing to design variants to meet the requests of a small number of customers.

However, the segment of heavy vehicles is not homogeneous with respect to software and electronics development practices. For instance, the industrial partners in this paper face quite different market situations and hence employ different development techniques:

– CC Systems[2] (CCS) is developing and supplying advanced distributed embedded real-time control systems with focus on mobile applications. Examples, including both hardware and software, developed by CCS are forest harvesters, rock drilling equipment and combat vehicles. The systems developed by CCS are built to endure rough environments, and are characterised by safety criticality, high functionality, and the requirements on robustness and availability are high.
  CCS works as a distributed software development partner, and cooperates, among others, with Alvis Hägglunds[3], Timberjack[4] and Atlas Copco[5]. Experience from these companies are included in this paper, this makes our findings more representative for the business segment of heavy vehicles.

[2] CC Systems, Home page: http://www.cc-systems.com
[3] Alvis Hägglunds, Home page: http://www.alvishagglunds.se
[4] Timerjack, Home page: http://www.timberjack.com
[5] Atlas Copco, Home page: http://www.atlascopco.com

CCS' role as subcontractor requires a high degree of flexibility with respect to supported target environments. Often, CCS' customers have requirements regarding what hardware or operating systems platforms to use, hence CCS cannot settle to support only some predefined set of environments. Nevertheless, to gain competitive advantages, CCS desires to reuse software between different platforms.

– Volvo Construction Equipment (VCE) is one of the world's major manufacturers of construction equipment, with a product range encompassing wheel loaders, excavators, motor graders, and more. What these products have in common is that they demand high reliability control systems that are maintainable and still cheap to produce. The systems are characterised as distributed embedded real-time systems, which must perform in an environment with limited hardware resources.
  VCE develops the vehicle electronics and most software in house. Some larger software parts, such as the operating system, are bought from commercial suppliers. VCE's role as both system owner and system developer gives them full control over the system's architecture. This, in turn, has given them the possibility to select a small set of (similar) hardware platforms to support, and select a single operating systems to use. Despite this degree of control over the system, VCE's experience is that software reuse is still hindered; for instance by non-technical issues like version and variant management, and configuration management.

3.2 System Description

In order to describe the context for software components in the vehicular industry, we will first explore some central concepts in vehicle electronic systems. Here, we outline some common and typical solutions and principles used in the design of vehicle electronics. The purpose is to describe commonly used solutions, and outline the de facto context for application development and thereby also requirements for software component technologies.

The system architecture can be described as a set of computer nodes called Electronic Control Units (ECUs). These nodes are distributed throughout the vehicle to reduce cabling, and to provide local control over sensors and actuators. The nodes are interconnected by one or more communication bus forming the network architecture of the vehicle. When several different organisations are developing ECUs, the bus often acts as the interface between nodes, and hence also between the organisations. The communication bus is typically low cost and low bandwidth, such as the Controller Area Network (CAN) [12].

In the example shown in Fig. 1 on the facing page, the two communication busses are separated using a gateway. This is an architectural pattern that can be used for several reasons, e.g., separation of criticality, increased total communication bandwidth, fault tolerance, compatibility with standard protocols [13–15], etc. Also, safety critical functions may require a high level of verification, which is usually very costly. Thus, non-safety related functions might be

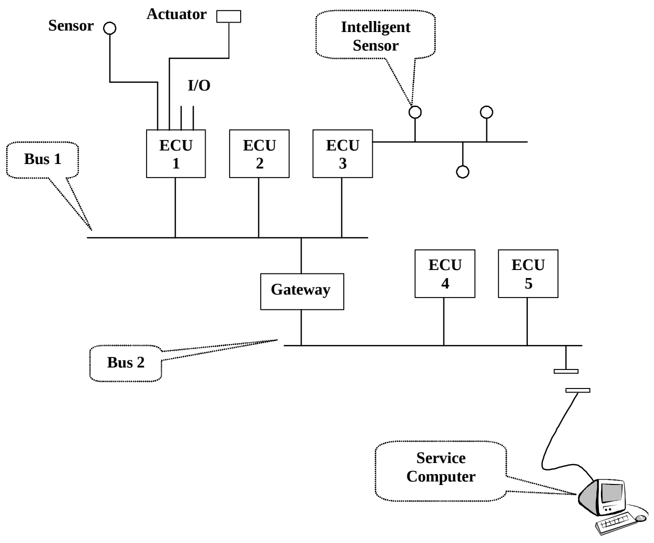

Fig. 1. Example of a vehicle network architecture

separated to reduce cost and effort of verification. In some systems the network is required to give synchronisation and provide a fault tolerance mechanisms.

The hardware resources are typically scarce due to the requirements on low product cost. Addition of new hardware resources will always be defensive, even if customers are expected to embrace a certain new function. Because of the uncertainty of such expectations, manufacturers have difficulties in estimating the customer value of new functions and thus the general approach is to keep resources at a minimum.

In order to exemplify the settings in which software components are considered, we have studied our industrial partner's currently used nodes. Below we list the hardware resources of a typical ECU with requirements on sensing and actuating, and with a relatively high computational capacity (this example is from a typical power train ECU):

Processor:	25 MHz 16 bit processor (e.g. Siemens C167)
Flash:	1 MB used for applications
RAM:	128 kB used for the runtime memory usage
EEPROM:	64 kB used for system parameters
Serial interfaces:	RS232 or RS485, used for service purpose
Communications:	Controller Area Network (CAN) (one or more interfaces)
I/O:	There is a number of digital and analogue in and out ports

Also, included in a vehicle's electronic system can be display computer(s) with varying amounts of resources depending on product requirements. There may also be PC-based ECU's for non-control applications such as telematics, and information systems. Furthermore, in contrast to these resource intense ECU's, there typically exists a number of small and lightweight nodes, such as, intelligent sensors (i.e. processor equipped, bus enabled, sensors).

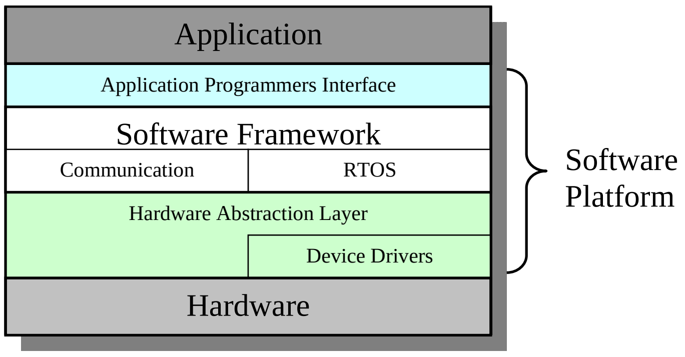

Fig. 2. Internals of an ECU - A software platform

Figure 2 depicts the typical software architecture of an ECU. Current practice typically builds on top of a reusable "software platform", which consists of a hardware abstraction layer with device drivers and other platform dependent code, a Real-Time Operating System (RTOS), one or more communication protocols, and possibly a software (component) framework that is typically company (or project) specific. This software platform is accessible to application programmers through an Application Programmers Interface (API). Different nodes, presenting the same API, can have different realisation of the different parts in the software platform (e.g. using different RTOSs).

Today it is common to treat parts of the software platform as components, e.g. the RTOS, device drivers, etc, in the same way as the ECU's bus connectors and other hardware modules. That is, some form of component management process exists; trying to keep track of which version, variant, and configuration of a component is used within a product. This component-based view of the software platform is however not to be confused with the concept of CBSE since the components does not conform to standard interfaces or component models.

4 Requirements on a Component Technology for Heavy Vehicles

There are many different aspects and methods to consider when looking into questions regarding how to capture the most important requirements on a component technology suited for heavy vehicles. Our approach has been to cooperate with our industrial partners very closely, both by performing interviews and by participating in projects. In doing so, we have extracted the most important requirements on a component-based technique from the developers of heavy vehicles point of view.

The requirements are divided in two main groups, the technical requirements (Sect. 4.1) and the development process related requirements (Sect. 4.2). Also, in Sect. 4.3 we present some implied (or derived) requirements, i.e. requirements that we have synthesised from the requirements in sections 4.1 and 4.2, but that are not explicit requirements from industry. In Sect. 4.4 we discuss, and draw conclusions from, the listed requirements.

4.1 Technical Requirements

The technical requirements describe the needs and desires that our industrial partners have regarding the technically related aspects and properties of a component technology.

4.1.1 Analysable Vehicle industry strives for better analyses of computer system behaviour in general. This striving naturally affects requirements placed on a component model. System analysis, with respect to non-functional properties, such as the timing behaviour and the memory consumption, of a system built up from well-tested components is considered highly attractive. In fact, it is one of the single most distinguished requirements defined by our industrial partners.

When analysing a system, built from well-tested, functionally correct, components, the main issues is associated with composability. The composability problem must guarantee non-functional properties, such as the communication, synchronisation, memory, and timing characteristics of the system [1].

When considering timing analysability, it is important to be able to verify (1) that each component meet its timing requirements, (2) that each node (which is built up from several components) meet its deadlines (i.e. schedulability analysis), and (3) to be able to analyse the end-to-end timing behaviour of functions in a distributed system.

Because of the fact that the systems are resource constrained (Sect. 3), it is important to be able to analyse the memory consumption. To check the sufficiency of the application memory, as well as the ROM memory, is important. This check should be done pre-runtime to avoid failures during runtime.

In a longer perspective, it is also desirable to be able to analyse properties like reliability and safety. However, these properties are currently deemed too difficult to address on a component level and traditional methods (like testing and reviewing) are considered adequate.

4.1.2 Testable and debuggable It is required that there exist tools that support debugging both at component level, e.g. a graphical debugging tool showing the components in- and out-port values, and at the traditional white-box source code debugging level. The test and debug environment needs to be "component aware" in the sense that port-values can be monitored and traced and that breakpoints can be set on component level.

Testing and debugging is by far the most commonly used technique to verify software systems functionality. Testing is a very important complement to analysis, and it should not be compromised when introducing a component technology.

In fact, the ability to test embedded-system software can be improved when using CBSE. This is possible because the component functionality can be tested in isolation. This is a desired functionality asked for by our industrial partners. This test should be used before the system tests, and this approach can help finding functional errors and source code bugs at the earliest possible opportunity.

4.1.3 Portable The components, and the infrastructure surrounding them, should be platform independent to the highest degree possible. Here, platform independent means hardware independent, RTOS independent and communication protocol independent.

Components are kept portable by minimising the number of dependencies to the software platform. Such dependencies are off course necessary to construct an executable system, however the dependencies should be kept to a minimum, and whenever possible dependencies should be generated automatically by configuration tools.

Ideally, components should also be independent of the component framework used during run-time. This may seem far fetched, since traditionally a component model has been tightly integrated with its component framework. However, support for migrating components between component frameworks is important for companies cooperating with different customers, using different hardware and operating systems, such as CC Systems.

4.1.4 Resource Constrained The components should be small and lightweighted and the components infrastructure and framework should be minimised. Ideally, there should no run-time overhead compared to not using a CBSE approach.

Systems are resource constrained to lower the production cost and thereby increase profit. When companies design new ECUs, future profit is the main concern. Therefore the hardware is dimensioned for anticipated use but not more.

Provided that the customers are willing to pay the extra money, to be able to use more complex software functionality in the future, more advanced hardware may be appropriate. This is however seldom the case, usually the customers are very cost sensitive. The developer of the hardware rarely takes the extra cost to extend the hardware resources, since the margin of profit on electronics development usually is low.

One possibility, that can significantly reduce resource consumption of components and the component framework, is to limit the possible run-time dynamics. This means that it is desirable to allow only static, off-line, configured systems.

Many existing component technologies have been design to support high run-time dynamics, where components are added, removed and reconfigured at run-time. However, this dynamic behaviour comes at the price of increased resource consumption.

4.1.5 Component Modelling

A component technology should be based on a standard modelling language like UML [16] or UML 2.0 [17]. The main reason for choosing UML is that it is a well known and thoroughly tested modelling technique with tools and formats supported by third-party developers.

The reason for our industrial partners to have specific demands in these details, is that it is belived that the business segment of heavy vehicles does not have the possibility do develop their own standards and practices. Instead they preferably relay on the use of simple and mature techniques supported by a welth of third party suppliers.

4.1.6 Computational Model

Components should preferably be passive, i.e. they should not contain their own threads of execution. A view where components are allocated to threads during component assembly is preferred, since this is believed to enhance reusability, and to limit resource consumption. The computational model should be focused on a pipe-and-filter model [18]. This is partly due to the well known ability to schedule and analyse this model off-line. Also, the pipes-and-filters model is a good conceptual model for control applications.

However, experience from VCE shows that the pipe-and-filter model does not fit all parts of the system, and that force fitting applications to the pipe-and-filter model may lead to overly complex components. Hence, it is desirable to have support for other computational models; unfortunately, however, which models to support is not obvious and is an open question.

4.2 Development Requirements

When discussing requirements for CBSE technologies, the research community often overlooks requirements related to the development process. For software developing companies, however, these requirements are at least as important as the technical requirements. When talking to industry, earning money is the main focus. This cannot be done without having an efficient development processes deployed. To obtain industrial reliance, the development requirements need to be considered and addressed by the component technology and tools associated with the technology.

4.2.1 Introducible

It should be possible for companies to gradually migrate into a new development technology. It is important to make the change in technology as safe and inexpensive as possible.

Revolutionary changes in the development technique used at a company are associated with high risks and costs. Therefore a new technology should be possible to divide into smaller parts, which can be introduced separately. For instance, if the architecture described in Fig. 2 is used, the components can be used for application development only and independently of the real-time operating system. Or, the infrastructure can be developed using components, while the application is still monolithic.

One way of introducing a component technology in industry, is to start focusing on the development process related requirements. When the developers have accepted the CBSE way of thinking, i.e. thinking in terms of reusable software units, it is time to look at available component technologies. This approach should minimise the risk of spending too much money in an initial phase, when switching to a component technology without having the CBSE way of thinking.

4.2.2 Reusable

Components should be reusable, e.g., for use in new applications or environments than those for which they where originally designed [19]. The requirement of reusability can be considered both a technical and a development process related requirement. Development process related since it has to deal with aspects like version and variant management, initial risks and cost when building up a component repository, etc. Technical since it is related to aspects such as, how to design the components with respect to the RTOS and HW communication, etc.

Reusability can more easily be achieved if a loosely coupled component technology is used, i.e. the components are focusing on functionality and do not contain any direct operating system or hardware dependencies. Reusability is simplified further by using input parameters to the components. Parameters that are fixed at compile-time, should allow automatic reduction of run-time overhead and complexity.

A clear, explicit, and well-defined component interface is crucial to enhance the software reusability. To be able to replace one component in the software system, a minimal amount of time should be spent trying to understand the component that should be interchanged.

It is, however, both complex and expensive to build reusable components for use in distributed embedded real-time systems [1]. The reason for this is that the components must work together to meet the temporal requirements, the components must be light-weighted since the systems are resource constrained, the functional errors and bugs must not lead to erroneous outputs that follow the signal flow and propagate to other components and in the end cause unsafe systems. Hence, reuse must be introduced gradually and with grate care.

4.2.3 Maintainable

The components should be easy to change and maintain, meaning that developers that are about to change a component need to understand the full impact of the proposed change. Thus, not only knowledge about component interfaces and their expected behaviour is needed. Also, information about current deployment contexts may be needed in order not to break existing systems where the component is used.

In essence, this requirement is a product of the previous requirement on reusability. The flip-side of reusability is that the ability to reuse and reconfigure the components using parameters leads to an abundance of different configurations used in different vehicles. The same type of vehicle may use different software settings and even different component or software versions. So, by introducing reuse we introduce more administrative work.

Reusing software components lead to a completely new level of software management. The components need to be stored in a repository where different versions and variants need to be managed in a sufficient way. Experiences from trying to reuse software components show that reuse is very hard and initially related with high risks and large overheads [1]. These types of costs are usually not very attractive in industry.

The maintainability requirement also includes sufficient tools supporting the service of the delivered vehicles. These tools need to be component aware and handle error diagnostics from components and support for updating software components.

4.2.4 Understandable

The component technology and the systems constructed using it should be easy to understand. This should also include making the technology easy and intuitive to use in a development project.

The reason for this requirement is to simplify evaluation and verification both on the system level and on the component level. Also, focusing on an understandable model makes the development process faster and it is likely that there will be fewer bugs.

It is desirable to hide as much complexity as possible from system developers. Ideally, complex tasks (such as mapping signals to memory areas or bus messages, or producing schedules or timing analysis) should be performed by tools. It is widely known that many software errors occur in code that deals with synchronisation, buffer management and communications. However, when using component technologies such code can, and should, be automatically generated; leaving application engineers to deal with application functionality.

4.3 Derived Requirements

Here, we present two implied requirements, i.e. requirements that we have synthesised from the requirements in sections 4.1 and 4.2, but that are not explicit requirements from the vehicular industry.

4.3.1 Source Code Components

A component should be source code, i.e., no binaries. The reasons for this include that companies are used to have access to the source code, to find functional errors, and enable support for white box testing (Sect. 4.1.2). Since source code debugging is demanded, even if a component technology is used, black box components is undesirable.

Using black-box components would, regarding to our industrial partners, lead to a feeling of not having control over the system behaviour. However, the possibility to look into the components does not necessary mean that you are allowed to modify them. In that sense, a glass-box component model is sufficient.

Source code components also leaves room for compile-time optimisations of components, e.g., stripping away functionality of a component that is not used in a particular application. Hence, souce code components will contribute to lower resource consumption (Sect. 4.1.4).

4.3.2 Static Configuration

For a component model to better support the technical requirements of analysability (Sect. 4.1.1), testability (Sect. 4.1.2), and light-weightiness (Sect. 4.1.4), the component model should be configured pre-runtime, i.e. at compile time. Component technologies for use in the office/Internet domain usually focus on a dynamic behaviour [8, 9]. This is of course appropriate in this specific domain, where powerful computers are used. Embedded systems, however, face another reality - with resource constrained ECU's running complex, dependable, control applications. Static configuration should also improve the development process related requirement of understandability (Sect. 4.2.4), since there will be no complex run-time reconfigurations.

Another reason for the static configuration is that a typical control node, e.g. a power train node, does not interact directly with the user at any time. The node is started when the ignition key is turned on, and is running as a self-contained control unit until the vehicle is turned off. Hence, there is no need to reconfigure the system during runtime.

4.4 Discussion

Reusability is perhaps the most obvious reason to introduce a component technology for a company developing embedded real-time control systems. This matter has been the most thoroughly discussed subject during our interviews. However, it has also been the most separating one, since it is related to the question of deciding if money should be invested in building up a repository of reusable components.

Two important requirements that has emerged during the discussions with our industrial partners are safety and reliability. These two are, as we see it, not only associated with the component technology. Instead, the responsibility of designing safe and reliable system rests mainly on the system developer. The technology and the development process should, however, give good support for designing safe and reliable systems.

Another part that has emerged during our study is the need for a quality rating of the components depending on their success when used in target systems. This requirement can, e.g., be satisfied using Execution Time Profiles (ETP's), discussed in [20]. By using ETPs to represent the timing behaviour of software components, tools for stochastic schedulability analysis can be used to make cost-reliability trade offs by dimensioning the resources in a cost efficient way to achieve the reliability goals. There are also emerging requirements regarding the possibilities to grade the components depending on their software quality, using for example different SIL (Safety Integrity Levels) [21] levels.

5 Conclusions

Using software components and a CBSE approach is, by industry, seen as a promising way to address challenges in product development including integration, flexible configuration, as well as good reliability predictions, scalability, reliable reuse, and fast development. However, changing the software development process to using CBSE does not only have advantages. Especially in the short term perspective, introducing CBSE represents significant costs and risks.

The main contribution of this paper is that it straightens out some of the question-marks regarding actual industrial requirements placed on a component technology. We describe the requirements on a component technology as elicited from two companies in the business segment of heavy vehicles. The requirements are divided in two main groups, the technical requirements and the development process related requirements. The reason for this division is mainly to clarify that the industrial actors are not only interested in technical solutions, but also in improvements regarding their development process.

The list of requirements can be used to evaluate existing component technologies before introducing them in an industrial context, therefore minimising the risk when introducing a new development process. Thus, this study can help companies to take the step into tomorrow's technology today. They can also be used to guide modifications and/or extensions to component technologies, to make them better suited for industrial deployment within embedded system companies.

We will continue our work by evaluating existing software component technologies with respect to these requirements. Our initial findings from this evaluation can be found in [22]. Using that evaluation we will (1) study to what extent existing technologies can be adapted in order to fulfil the requirements of this paper, (2) investigate if selected parts of standard technologies like tools, middleware, and message-formats can be reused, (3) make a specification of a component technology suitable for heavy vehicles, and (4) build a test bed implementation based on the specification.

Acknowledgements

A special thanks to Nils-Erik Bånkestad and Robert Larsson, at Volvo Construction Equipment, for fruitful discussions and for their helpfulness during our stay. We would also like to thank Jörgen Hansson at CC Systems for interesting discussions, new ideas, and for making this research project possible.

References

1. Crnkovic, I., Larsson, M.: Building Reliable Component-Based Software Systems. Artech House publisher (2002) ISBN 1-58053-327-2.
2. Winter, M., Genssler, T., et al.: Components for Embedded Software – The PECOS Apporach. In: The Second International Workshop on Composition Languages, in conjunction with the 16th ECOOP. (2002) Malaga, Spain.

3. van Ommering, R., et al.: The Koala Component Model for Consumer Electronics Software. IEEE Computer **33(3)** (2000) 78–85
4. Wallnau, K.C.: Volume III: A Component Technology for Predictable Assembly from Certifiable Components. Technical report, Software Engineering Institute, Carnegie Mellon University (2003) Pittsburg, USA.
5. Brown, A., Wallnau, K.: The Current State of CBSE. IEEE Software (1998)
6. Nordström, C., Gustafsson, M., et al.: Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry. In: Eigth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. (2001) Washington, USA.
7. Schach, S.R.: Classical and Object-Oriented Software Engineering. McGraw-Hill Science/Engineering/Math; 3rd edition (1996) ISBN 0-256-18298-1.
8. Microsoft Component Technologies: (COM/DCOM/.NET) http://www.-microsoft.com.
9. Sun Microsystems: (Enterprise Java Beans Technology) http://java.sun.com/-products/ejb/.
10. Object Management Group: MinimumCORBA 1.0 (2002) http://www.omg.org/-technology/documents/formal/minimum_CORBA.htm.
11. Möller, A., Fröberg, J., Nolin, M.: What are the needs for components in vehicular systems? – An Industrial Perspective. In: WiP Proc. of the 15th Euromicro Conference on Real-Time Systems, IEEE Computer Society (2003) Porto, Portugal.
12. International Standards Organisation (ISO): Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication (1993) vol. ISO Standard 11898.
13. CiA: CANopen Communication Profile for Industrial Systems, Based on CAL (1996) CiA Draft Standard 301, rev 3.0, http://www.canopen.org.
14. SAE Standard: (SAE J1939 Standards Collection) http://www.sae.org.
15. SAE Standard: (SAE J1587, Joint SAE/TMC Electronic Data Interchange Between Microcomputer Systems In Heavy-Duty Vehicle Applications) http://www.-sae.org.
16. Selic, B., Rumbaugh, J.: Using UML for modelling complex real-time systems (1998) Rational Software Corporation.
17. Object Management Group: UML 2.0 Superstructure Specification, The OMG Final Adopted Specification (2003) http://www.omg.com/uml/.
18. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall; 1 edition (1996) ISBN 0-131-82957-2.
19. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch or why it's hard to build systems out of existing parts. In: Proceedings of the Seventeenth International Conference on Software Engineering. (1995) Seattle, USA.
20. Nolte, T., Möller, A., Nolin, M.: Using Components to Facilitate Stochastic Schedulability. In: WiP Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS), IEEE Computer Society (2003) Cancun, Mexico.
21. SIL: Safety Integrity Levels – Does Reality Meet Theory? (2002) Report f. seminar held at the IEE, London, on 9 April 2002.
22. Möller, A., Åkerholm, M., Fredriksson, J., Nolin, M.: Software Component Technologies for Real-Time Systems – An Industrial Perspective. In: WiP Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS), IEEE Computer Society (2003) Cancun, Mexico.