Introducing Component Based Software Engineering at an Embedded Systems Sub-Contractor

Mikael Åkerholm, Kristian Sandström and Ivica Crnkovic
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
{mikael.akerholm, kristian.sandstrom and ivica.crnkovic}@mdh.se

Abstract

Attractive benefits with successful implementation of component-based principles include managing complexity, reduction of time-to-market, increased quality, and reusability. Deployment of component-based development is however not simple - it depends on many strategic, technical, and business decisions. In this paper we report experiences from our attempts with finding a correct implementation of component-based principles for the business situation of sub-contractors of embedded systems.

Findings related to suitable component models, component technologies, and component management are presented. Overall the results confirm the suitability of component-based principles for the domain, but also show the need (and potential) in further development of CBSE theory and technology for embedded systems.

1 Introduction

In this paper we report our experiences from introducing Component-Based Software Engineering (CBSE) [11, 14] at the company, CC Systems[1], acting as sub-contractor and Commercial Off The Shelf (COTS) components supplier for embedded systems.

CBSE can be seen as analogous to engineering approaches in other engineering domains. For examples, mechanical engineers build systems using well-specified components such as nuts and bolts, and the building industry uses components as large as walls and roofs (in turn assembled from smaller components). CBSE has proven to be effective for desktop and web-applications, however, not yet for development of software for embedded systems. Implementation and deployment of CBSE for development of embedded systems is not trivial. As its success depends on many factors where some of them are selection (or development) of a component technology that can be efficiently used in the development and maintenance process, and satisfy the run-time requirements of the particular domain.

[1]Cross Country Systems, http://www.cc-systems.com

We present experiences from four cases in this paper. One case, SaveCCT, is a study where a group of researches demonstrates a prototype component technology in a real industrial environment. The second case, CrossTalk, utilises CBSE principles for realizing a software platform supporting "any" system consisting of the company's hardware. The third case, CC Components, make use of a component repository when possibilities to create or reuse components arise in the development projects. Finally, the fourth study is an evaluation of a method supporting the sometimes necessary work with adaptation of components to fit usage in different development projects.

Our findings indicate that CBSE principles are suitable for embedded systems sub-contractors, but it might be harder to practice CBSE as sub-contractor than product owning company. The necessary technical needs of, e.g., expressiveness in the component models, resource efficiency of component based applications, and analysis possibilities can be considered met.

The following section (section 2) presents the motivation and goals with the research presented in this paper. Section 3 presents data for the different cases we have studied. The findings from our experiences when working with the different cases are reported in section 4. A discussion of the results is provided in section 5. Finally, section 6 concludes the paper.

2 Goals and Motivation

The primary goal of this experience paper is to contribute to the overall understanding of the needs of component technologies and processes for practicing CBSE in the domain of embedded control systems for vehicles and machines.

The studies have all been performed at CC Systems engineering sites in Finland and Sweden. CC Systems develops electronics targeting vehicles and machines in rough environments. From software and hardware controlling safety critical by-wire functions, to software and hardware for powerful on-board display based information systems with back-office connections.

The studies focus on the control related part of the systems. The focus is chosen because CBSE as approach have had a limited success for development of such systems, in comparison to the domain of PC applications where the approach has emerged. The major reason to this is a number of important qualities that leaven all through the software life-cycle, e.g., safety, reliability, timing, and resource efficiency. It known that these qualities are not addressed by most existing commercial component technologies, and consequently these systems cannot be developed with such component technologies. However, many component technologies that might be suitable exists within academia and some are to a limited extent used within industry, e.g., Koala [24] used internally at Philips[2], Rubus [18] used by some Swedish vehicle manufacturers, and different implementations of the IEC61131-3 standard [16]. However, as pointed out in [9], there is currently no de-facto standard component technology within the domain of vehicular systems; although CBSE seams to get a lot of attention from industry, e.g., East[3] and Autosar[4]. This leads us to the goal of assessing if the limited success of CBSE in the domain is depending on an inability of existing commercial technologies to support the requirements of embedded vehicular applications.

Another very interesting question is how an ideal component model for the domain should take the trade-off between supporting predictability, and ease to express common functionality in vehicular control systems? The core part of a component model is related to defining what a component is, and possibilities for component interaction. There are several important design decisions that have to be made when defining a component model and one is the trade-off between flexibility and predictability. It is the trade-off with respect to ease of implementing vehicular control systems (high degree of flexibility), and support for prediction of quality attributes considered important in the domain (high degree of predictability). A design choice of our suggested component model, SaveCCM [12], is in contrast to many of the current component models to sacrifice some flexibility to facilitate analysis and predictability.

Finally resource-efficiency (the consumption of a minimum of resources in achieving an objective) is important in the domain since products are typically produced in high volumes. Poor resource efficiency in component frameworks might be an important reason for not choosing CBSE [9]. At the same time the basic ideas of CBSE has been driven from the needs of PC/Internet applications where resource efficiency typically is not an issue. A reusable component should according to CBSE theory be general. An obvious method to create a general component is to implement support for many methods that might suit different purposes. Using this approach imply that you might end up with using only a small part of the component, and the rest is "dead code" in the application. This might be a problem for resource constrained embedded systems, and highly recommended to avoid for systems with high Safety Integrity Level (SIL), i.e., SIL 2 and above in IEC61508 [15].

3 CBSE Activities

Four CBSE activities are summarized in the following sub-sections, denoted *Case 1-4*. *Case 1* is a study by a group of researchers in a real industrial environment; the study evaluates technical properties of a component technology. In *Case 2*, the company utilises CBSE principles for realizing a Product-Line Architecture (PLA) [7] for platform software, here the suitability of CBSE and the component technology are evaluated. In *Case3* component-based reuse is practiced when opportunities arise, here the CBSE principles in this context are evaluated. Finally, *Case 4* is an evaluation by researchers in an industrial environment of a method related to component adaptation, here the method itself is evaluated. The experiences and lessons learned of the cases are summarised in section 4.

3.1 Case 1, SaveCCT

Case 1 is a demonstration of the component technology SaveCCT [1], by usage on a fictive but representative application in a real industrial environment at CC Systems. The main purpose with this case is evaluation of the technical properties of the component technology.

The research prototype of the SaveComp Component Technology (SaveCCT) used during this study is visualized in Figure 1. SaveCCT is described at the

[2]Philips, http://www.philips.com/
[3]East, http://www.east-eea.net/
[4]Autosar, http://www.autosar.org

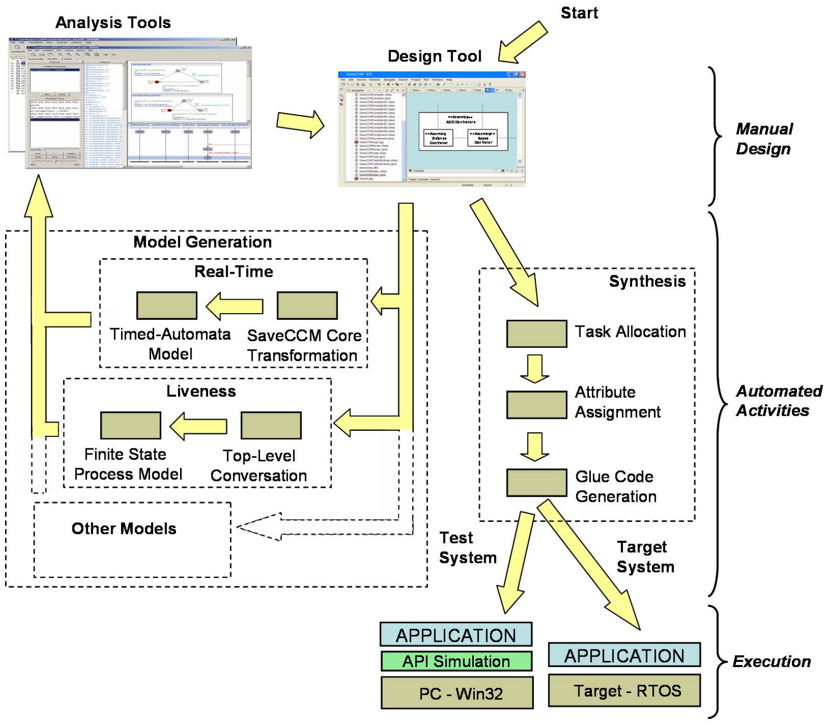

Figure 1. An overview of our research prototype SaveCCT

top level, by distinguishing manual design, automated activities, and execution.

Manual design is the entry point for the development; here a component-based strategy is used, supported by a set of tools for design and analysis. The SaveCCT design tool provides support for graphical assembly of applications from existing components. The tool allows designers to specify the component interconnection logics, and express high level constraints on the resulting application. Assembling components is done with respect to the rules of the SaveComp Component Model (SaveCCM) [12]. The component model defines different component types that are supported by SaveCCT, possible interaction schemes between components, and clarifies how different resources are bound to components. As shown in the figure, SaveCCT incorporates a number of analysis tools, which can be used for verifying specific attributes of the application, e.g., related to timeliness and safety. To efficiently incorporate an analysis tool, as much as possible of the translation from the model created with the design tool to the model required by the desired analysis tool should be automated. In this study we incorporated LTSA [19], and Times [3].

Automated activities produce necessary code for the run-time system (i.e., glue-code), and different specialized models of the application for analysis tools. The synthesis activity generates all low level code (i.e., hardware and operating system interaction), meaning that components are free from dependencies to the underlying platform. Furthermore, the code generation step statically resolves resource usage and timing, with the strategy to resolve as much as possible during compile-time instead of depending on costly run-time algorithms.

To achieve efficient and predictable run-time behaviour, and reliable support for pre-runtime analysis, SaveCCT assumes a real-time operating system (RTOS) as underlying platform. The prototype used the Quadros[5] RTXC operating system, which is a standard fixed-priority pre-emptive multitasking RTOS used in some applications by CC Systems. The supported target hardware in the current version is CrossFire MX1 from CC Systems, which is an electronic control unit intended for control systems running in rough environments. To facilitate testing and debugging we incorporate CCSimTech [20], which is a simulation framework that offers generic hardware emulation components for common hardware in embedded systems, e.g., I/O (digital and analogue), network technologies, and memories. This environment represents a typical platform used in development projects

[5]Quadros, http://www.quadros.com/

by CC Systems, and thus serves as an example of an industrial environment for the SaveCCT prototype.

3.2 Case 2, CrossTalk

Case 2, CrossTalk [6], is an initiative driven by CC Systems, which have taken influences from the research demonstrated in *Case 1*. The main goal with this initiative is to take advantage of the support for product-line architectures that component-based approaches give. The goal is rapid and cost effective assembly of platform software, through enabling addition and/or replacement of components to a baseline platform depending on the needs from a certain application. The CrossTalk platform has been used in numerous real development projects by CC Systems.

A CrossTalk based system is built on an open-ended component-based CrossTalk platform, the concept is to have *one* platform to build *any* system consisting of the company's own hardware. Figure 2 illustrates the concept, which we describe here with the following list:

1. System architecture, in terms of computer nodes and their responsibility is established. The hardware for a CrossTalk system are selected among more than ten different nodes, e.g., control modules, communication gateways, and display units. The communication between the different nodes on the machine is based on CANopen[6] this means that it is also possible to integrate any third-party node in the system that uses the CANopen protocol, but treatment of such nodes is beyond the scope of this paper.

2. Based on the functionality designated to each computer node, platform components from the CrossTalk repository are selected to constitute software platform for each node in the system. The repository is based on a standard version control system, and the components are IEC61131-3 [16] components. The components are assembled using the CoDeSys tool from 3S[7].

3. The open-ended platform software is deployed on each of the nodes in the system; the application is then built by the customer or by CC Systems in a separate project, by continuing the work in the CoDeSys tool.

3.3 Case 3, CCComponents

CC Components is another initiative with influences from *Case 1*. Here the intention is to package reusable parts of applications into software components, and to reuse components when suitable. Notice here that the intention is not to build entirely component-based systems; systems are built through a combination of components and non-component-based software.

[6]CiA, CANopen, http://www.can-cia.org/

[7]3S CoDeSys, http://www.3s-software.com/

To efficiently take advantage of CBSE, development processes for system development, component assessment, and component development are separated, as proposed in e.g., [10].

As demonstrated in Figure 3, CC Systems has a system development process based on Rational Unified Process (RUP)[8], thus this process needs no further description here. In the inception phase all development projects should initiate a component assessment process with the intention to find suitable components to reuse, and suggestions for specifications of components to develop. The component assessment process has the following steps:

Find, component assessment starts with finding components that might provide the required functionality for some part in the project.

Select, if candidate components for reuse are found, the selection of which components to reuse in the project is documented and motivated.

Specify, if no candidate components are found, but the project finds a certain part of the system very suitable to be packaged as a reusable component a specification for such a component is created.

Evaluate, specifications are evaluated by the CC Components board (a group responsible for the repository). Generally, the board must be convinced that the suggested component will be target for reuse in other projects, before a separate component development project is initiated.

Verify, this step is required to test that the component really fits the intended purpose as soon as possible to avoid that the component is assumed to be fit and well-tested until the very late stages in the system development project.

Component development is guided by the same RUP-based process as system development, but the target to develop reusable components is made clear through lifting the development from the process of a particular project to a separate process ending with delivery to the common company-wide CC Components repository. There is no formal component definition. The only technical requirement is that the components

[8]IBM Rational, http://www-306.ibm.com/software/rational

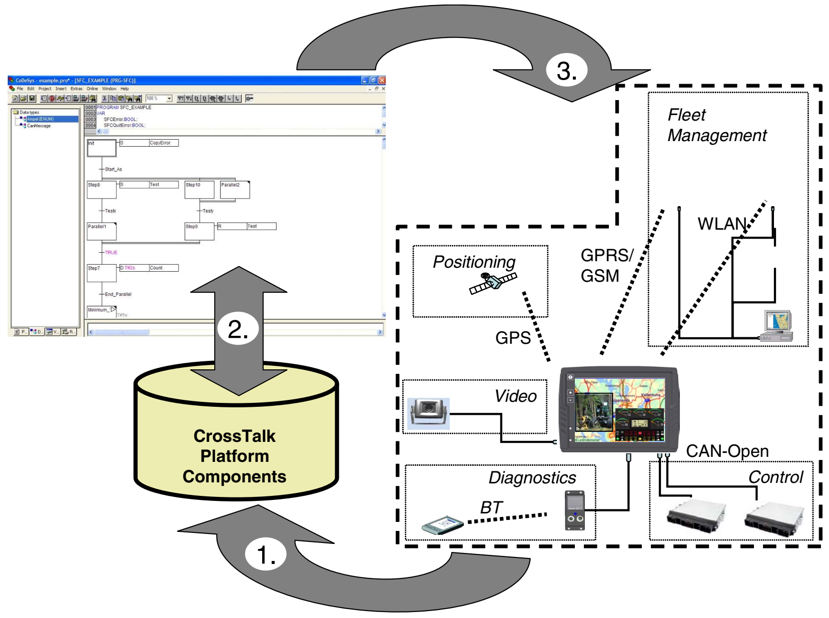

Figure 2. Workflow when assembling a software platform for a CrossTalk-based system

in the repository must have all their dependencies specified in the interfaces, combined with requirements on standardized documentation.

3.4 Case 4, Component Metadata for Traceability

Case 4, is an evaluation of a prototype implementation [22] of a method supporting component assessment, and component development. The theory behind the method is described in [2], it is based on work by Orso et.al. [21] suggesting to (re)use component metadata to support software engineering tasks. The need is based on experiences from *Case 3*, where component assessments often results in needs for component adaptations, this will be further discussed in section 4. This case is based on a prototype demonstration on a representative software component, for a group consisting of two project managers, four developers, and one sales manager.

Figure 4, gives an overview of the method. The method affects the processes of component assessment, and component development, and there is a metadata associated with all components which is central in the method. The purpose of the metadata is to maintain traceability of requirements through design and testing during component development.

The metadata is collected during component development, and the arrows in the metadata in the figure illustrates that the metadata contains references between requirements, design, and test cases associated with the component. This information is traceability information of how the requirements fulfilled by the component are related to the internal realization of the component, and how the different test cases relate to different parts of the realization and different requirements.

The component assessment process has, in comparison to *Case 3* Figure 3, a new activity after Find called Modify:

Modify, here modification requests of existing components are created, when modifications are necessary for the system development project. These requests are then evaluated by the board of CC Components in the following Evaluation step, as previosly desribed in *Case 3*.

During the component assessment the metadata with traceability is utilized for performing impact analysis of a desired modification of a component. The purpose of the impact analysis is to estimate the amount of work and consequences of performing the desired modification. A prototype tool has been developed that automatically produces analysis of which parts of a components design and test cases that are affected by a modification. Where the modifications are defined through giving the desired change of the requirements. The output is used as input to the evaluation step where the board decides whether the component modification should be performed or dismissed.

During the component development process where a new variant of a component is developed, the impact

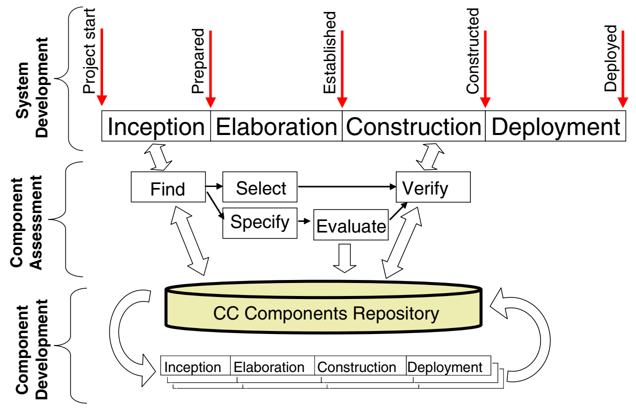

Figure 3. CC Components development processes

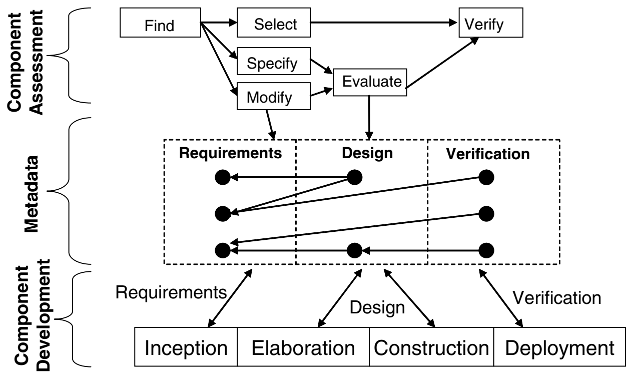

Figure 4. Using metadata in component assessment and component development

analysis gives guidance to the work. It gives information of which parts of the component that should be modified. It also specifies which test case that should be used for regression testing after the change, i.e., points out which test cases that must produce the same results.

The need to adapt software components have been known in the CBSE community, e.g., a survey on the topic in 1999 [13]. Common for many of the proposed techniques is the support for configuration of components, e.g., [8, 4]. However, the flip-side with these techniques is that future scenarios must be predicted, and that the configuration code increase complexity and thereby resource usage. The other main principle for existing techniques is to apply external adaptation through wrappers [5], adaptors [25], or connectors [17]. The main limitation here is that optimization of the component's internal realization is not possible, e.g., it is not possible to remove functionality. Thus, none of these techniques is perfect for the problem we have encoutered with resource constrained embedded systems.

4 Experiences

In this section we summarize the findings from the above reported cases. We do this case by case.

4.1 Case 1

The component model is based on data-flow (or pipes-and-filters) interaction, this has been chosen to give good support for expressing the key functionality of control systems. Designing the fictive application according to component-based principles was relatively straight-forward, and SaveCCM proved sufficiently expressive for this type of system.

The close integration of analysis tools, exemplified by LTSA and Times, enabled the researchers to derive a number of non-trivial properties automatically or with little manual intervention. In particular, the high predictability imposed by the SaveCCM semantics allowed analysis of properties crucial to ensure correct real-time behaviour, such as end-to-end response times. Likewise, the integration of CCSimTech provided good support for testing.

The resulting system was sufficiently resource efficient. It utilizes only a small part of the available capacity in the target hardware, which is approximately the utilization expected for this application in combination with state-of-practice programming methods (i.e., C and C++). The explicit triggering allows the synthesis mechanism to minimize communication overhead by identifying static triggering patterns.

We should also make clear that the demonstrated component technology was considered unusable in real development projects, since the quality and usability of the included prototype tools was considered below tolerable levels. Mature tools is a basic need in practice.

4.2 Case 2

The component model, i.e., IEC61131-3 with its roots in the automation domain, is based on data-flow (or pipes-and-filters) interaction. Studying this case fortify that this is a suitable component interface also in the vehicular domain for control related systems. Numerous control systems have successfully been built for vehicles and machines based on the CrossTalk concept.

Reuse of low-level components (i.e., CrossTalk platform components) through component based principles is successfully practiced in this case. The experience is that product-line architectures for platform software can be efficiently created with component-based principles. The company also stresses that the component-based principles in this case results in short and predictable development projects, and higher software quality.

Regarding resource efficiency, it is known that the CoDeSys run-time framework requires additional processing compared to realizing the system with lower level programming of the hardware. Thus, it is a trade-off between resource efficiency and development efficiency.

4.3 Case 3

Overall the introduction of a company-wide component repository is a success, every time a well-tested component can be reused "as is" without modification there is a good return of the investments made in the component development. Reuse was practiced before, but more unstructured, depending of the knowledge about reusable software assets inside the project team. The evaluation step of component specifications is seen as promising for the future to ensure that only reusable components are developed.

However, problems with reusing components have also been identified, which might be similar for other sub suppliers to customers with high volumes or safety critical applications. The basic foundation of CBSE, to build general components that can be (re)used in many applications, is harder to practice for a sub-supplier, especially in the domain of embedded control systems. Components may require functional additions or adaptation associated with reuse. The adaptation needs seems to be higher for the sub-suppliers business case. At the same time, due to domain requirements, components can often not include any extra functionality. Instead of being based on general components, applications must be dedicated and specialized to its task for high volume products. Safety critical applications are even worse since no "dead code" is allowed in source files or on target for certification according to the higher SIL levels of, e.g., IEC61508 [15]. Practicing reuse under these circumstances often require adaptations.

4.4 Case 4

The conclusion from *Case 4* is that the method is promising to support the adaptation needs. The method causes no overhead in the internal realisation of software component itself, and the components can be highly specialized for every scenario. If this can be efficiently implemented in practice, it supports the adaptation needs identified at the company.

Another positive side is reuse of the documentation of the traceability information given through the graphs in the metadata. This is becoming more and more important for all companies in the domain, due to legislation of using system safety standards in development. The IEC61508 standard [15], which is the main standard for functional safety of electronic programmable systems, requires traceability of requirements through the different stages of development. CMMI [23], a process improvement approach, even requires bidirectional traceability. This is the ability to trace requirements both forward and backward, i.e., requirements through the development process into the product and from the product backwards to requirements. This becomes possible, reusable, and well-documented, through the metadata.

On the negative side were the developers' concerns about using one tool more, when you ideally would like as few tools as possible to work efficiently. Another concern was the effort to create the dependency graphs, which might be time consuming and complex for big components.

5 Discussion

Here we discuss the findings related to the goals stated in section 2.

One of the purposes with the studies was to evaluate CBSE as engineering approach for the domain. To start with, the continuous interest and investments from the company in these activities indicates that CBSE is an attractive engineering approach. The success with both increased efficiency and quality in *Case 2* and *Case 3*, generally demonstrates the potential of CBSE and especially when it comes to PLA. It also shows that component technologies with tool-suites that are mature enough for industrial needs exist, here manifested through CoDeSys. Whether the functional blocks of the IEC61131-3 standard qualify as real software components according to some definition remains unsaid, but it is here proven that it is possible to treat them as components. The conclusion is that the use of CBSE in the domain should not be limited by lack of existing commercial technologies, even if we cannot dismiss this as a reason.

We could also observe that it can be problematic to reuse components for sub-contractors, primary from *Case 3*. Sub-contractors, as CC Systems in this case, might take contracts on realizing similar functions with slightly different requirements for different customers. If the target systems are safety-critical or produced in high volumes, general components (with the side-effect of being bigger) must often be discarded in favour of solutions tailored for the particular system.

It is interesting that within the same company it was possible to observe the usage of CBSE as product owner in *Case 2*, and as sub-contractor *Case 3*. The experiences from this is that it is definitively easier to take advantage of CBSE being a product-owner, in fact being a product owner you actually plan for reuse. The next generation of the system will most likely be an improvement of the existing system; it becomes natural to take reuse of existing components into consideration when planning for the next generation.

Next important concern from section 2 is the trade-off between flexibility and predictability in component models. Both in *Case 1* and *Case 2*, where more formal component models were used, the component model was based on data-flow (or pipes-and-filters) interaction, this has been chosen to give good support for expressing the key functionality of control systems. Designing the fictive application according to component-based principles was relatively straightforward, and SaveCCM proved sufficiently expressive for this type of system. CrossTalk has been used for numerous real control systems and it has proven to be suitable in every case. The basic interaction mechanism is thus well proven in practice, but important to stress is that we cannot dismiss other component interaction approaches from our studies. The analysis of real-time and reliability properties demonstrated in *Case 1* shows that is possible to create a component model that is expressive enough for the applications and at the same restrictive enough to allow this type of predictions. However this has not been proven in real projects.

The experiences also justify apprehensions concerning risks of poor resource efficiency of component-based applications. However, it is also demonstrated in *Case 1* that it is possible to resolve resource usage and timing statically during compile-time without costly run-time mechanisms, but this is not yet common in commercial mature technologies. This might actually be one of the reasons for the limited usage of CBSE in the domain today. Furthermore, *Case 4* demonstrated a method supporting the adaptation needs, which got positive feedback to address the specialization problem, but it has not been used in real projects.

6 Conclusions and Future Work

In this paper we have reported experiences from four cases where we have introduced/demonstrated CBSE principles at CC Systems.

Overall our findings indicate that CBSE principles are suitable for embedded systems sub-contractors, but also that it might be harder to practice CBSE as sub-contractor than product owner. The most technical needs of expressiveness in the component models, resource efficiency of component based applications, and analysis possibilities can be considered possible to fulfil with a combination of the contents in the different cases. According to our studies the most important need is related to resource efficiency. Resource efficient component frameworks with mature tools together with support for adaptation of software components themselves are needed.

For future work it would be interesting to explore more about the impact from the business situation on CBSE. In the domain of control systems for vehicles and machines we can identify three major business situations *sub-suppliers on contract basis*, *COTS suppliers*, and *product owners*. Note that it might be possible to study all these within a single company, as e.g., CC Systems, hopefully with increased possibilities to limit influences from other differences. Different goals with practicing CBSE would also be interesting to explore in combination with the different business models.

7 Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via PROGRESS and Save, and the KK-Foundation (KKS) via Save-IT.

References

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The save approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667, May 2007.

[2] M. Åkerholm, J. Fröberg, K. Sandström, and I. Crnkovic. A model for reuse and optimization of embedded software components. In *29th International Conference on Information technology Interfaces, (ITI 2007)*. IEEE, June 2007.

[3] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In

In Proceedings of 1st International Workshop on Formal Modeling and Analysis of Timed Systems. LNCS Springer, 2003.

[4] J. Bosch. Superimposition: A component adaptation technique. *Information and Software Technology*, 5(41), 1999.

[5] J. Brant, B. Foote, R. e. Johnson, and D. Roberts. Wrappers to the rescue. In *Proceedings of 12th European Confernece on Object-Oriented Programming (ECOOP98)*, July 1998.

[6] CC Systems. Crosstalk generic control system platform. Technical report, CC Systems, 2007.

[7] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* ISBN 0-201-70332-7. Addison-Wesley, 2001.

[8] K. Cooper, J. Zhou, H. Ma, I. L. Yen, and F. Bastani. Code parameterization for satisfaction of qos requirements in embedded software. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2003.

[9] I. Crnkovic. Component-based approach for embedded systems. In *9th International Workshop on Component-Oriented Programming*, Oslo, June 2004.

[10] I. Crnkovic, M. Chaudron, and S. Larsson. Component-based development process and component lifecycle. In *Proceedings of the International Conference on Software Engineering Advances, ICSEA'06*. IEEE, October 2006.

[11] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems.* Artech House publisher, 2002. ISBN 1-58053-327-2.

[12] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. SaveCCM – a component model for safety-critical real-time systems. In *Proc. 30th Euromicro Conference*, pages 627–635, 2004.

[13] G. T. Heineman. An evaluation of component adaptation techniques. In *2nd ICSE Workshop on Component-Based Software Engineering*, 1999.

[14] G. T. Heineman and W. T. Councill. *Component-based Software Engineering, Putting the Pieces Together.* Prentice-Hall, 2001. ISBN: 0-201-70485-4.

[15] International Electrotechnical Commission IEC. Standard: IEC61508, Functional Safety of Electrical/Electronic Programmable Safety Related Systems. Technical report.

[16] International Electrotechnical Commission IEC. *International Standard IEC 61131, Programmable controllers*, 1992.

[17] K.-K. Lau, L. Ling, and Z. Wang. Composing components in design phase using exogenous connectors. In *Proceedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications.* IEEE, 2006.

[18] K.-L. Lundbäck, J. Lundbäck, and M. Lindberg. Development of dependable real-time applications. Arcticus Systems, Dec. 2004.

[19] J. Magee and J. Kramer. *Concurrency: State Models & Java Programs.* John Wiley & Sons, Inc., New York, NY, USA, 1999.

[20] A. Möller and P. Åberg. A Simulation Technology for CAN-based Systems. *CAN Newsletter*, 4, December 2004.

[21] A. Orso, M. J. Harrold, D. Rosenblum, G. Rothermel, M. L. Soffa, and H. Do. Using component metacontents to support the regression testing of component-based software. In *Proceedings of the International Conference on Software Maintenance*, November 2001.

[22] Q. Tien Le. Component design tool for embedded system components. Technical report, Masters Thesis, MRTC, Mälardalen Univ., 2008.

[23] SEI. CMMI for development, version 1.2. Technical report, Technical Report CMU/SEI-2006-TR-008, 2006, 2006.

[24] R. van Ommering, F. van der Linden, K. Kramer, and J. Magee. The Koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, march 2000.

[25] D. M. Yellin and R. E. Strom. Protocol specification and component adaptors. *ACM Trans. on Programming Languages and Systems*, 2(19):292–333, March 1997.