A component-based development framework for supporting functional and non-functional analysis in control system design*

Johan Fredriksson
Mälardalen University
Mälardalen Real-Time Research Centre
Västerås, Sweden
johan.fredriksson@mdh.se

Massimo Tivoli
University of L'Aquila
Computer Science Department
L'Aquila, Italy
tivoli@di.univaq.it

Ivica Crnkovic
Mälardalen University
Mälardalen Real-Time Research Centre
Västerås, Sweden
ivica.crnkovic@mdh.se

Abstract

The use of component-based development (CBD) is growing in the software engineering community and it has been successfully applied in many engineering domains such as office applications and in web-based distributed applications. Recently, the need of CBD is growing also in other domains related to dependable and embedded systems, namely, in the control engineering domain. Control systems constitute the core functionality of modern embedded systems such as vehicles and consumer electronics. However, the widely used commercial component technologies are unable to provide solutions to the requirements of embedded systems as they require too much resource and they do not provide methods and tools for developing predictable and analyzable embedded systems. There is a need for new component-based technologies appropriate to development of embedded systems.

In this paper we briefly present a component-based development framework called SAVEComp. SAVEComp is developed for safety-critical real-time systems. One of the main characteristics of SAVEComp is syntactic and semantic simplicity which enables a high analyzability of properties important for embedded systems. By means of an industrial case-study, we show how SAVEComp is able to provide an efficient support for designing and implementing embedded control systems by mainly focusing on simplicity and analyzability of functional requirements and of real-time and dependability quality attributes. In particular we discuss the typical solutions of control systems in which feedback loops are used and which significantly complicate the design process. We provide a solution for increasing design abstraction level and still being able to reason about system properties using SAVEComp approach. Finally, we discuss an extension of SAVEComp with dynamic run-time property checking by utilizing run-time spare capacity that is normally induced by real-time analysis.

*This work is an extended and revisited version of [13].

1 Introduction

Due to the increasing complexity of control systems, they are often constructed performing a modular approach by means of libraries of building blocks with high functionality and a high degree of flexibility. This has lead to a need of a component-based approach for building control systems out of a set of already implemented *"control modules"* [11]. The control module concept has been implemented in *ABB's new control system, Control IT* as a more reliable and *easy-to-use* generalization of a traditional IEC61131-3 function block[1] [1]. A control module might be considered a control system component and hence it is the mean to build control systems by adopting a component-based approach supported by a suitable component technology. Although component models that support predictability of the system behaviour exist, they are often not able to support the requirements of embedded systems. For example, software components for embedded systems should provide an interface specification that points out specific resource requirements or other properties of interest for the target application, e.g., timing, memory usage and dependability-related attributes such as reliability and safety. Specific architectural constraints should be imposed on the system design in such a way that predictability of properties that are relevant for the domain can be supported. Even a component framework for embedded systems should use predictable mechanisms and be light weight. Thus, a component-based development framework which supports the requirements of embedded systems is highly needed in order to be able to predict functional and non-functional behaviour of control systems during design-time.

In this paper, we present a component-based development framework, called *SAVEComp*[2] that supports predictability of control system behaviour during design-time. The main purpose of SAVEComp is to provide efficient support for designing and implementing embedded control applications by mainly focusing on simplicity and analysability of functional requirements and real-time and dependability properties. Our reference component model is *SaveComp Component Model* (SaveCCM) [6] which is designed for safety-critical real-time systems. SaveCCM has been thought to support predictability of the real-time behaviour of embedded systems. We show how to extend the current version of SaveCCM in order to incorporate the control module concept in SAVEComp in such a way that we are able to predict the system behaviour. The design of a SaveCCM control module can be enriched with infor-

[1] In the reminder of the paper, we will use the term "function block" to identify a "IEC61131-3 function block" and all its further extensions (e.g., IEC61499 function blocks [7]).

[2] SAVEComp is developed in the project *SAfety critical components for VEhicular systems - http://www.mrtc.mdh.se/SAVE*.

mation about the module quality attributes by providing the ground support for the system analysis. By means of both the extended capabilities of SaveCCM and the analysis tools provided by SAVEComp, we show how the developer is able to build control systems by composing already implemented components in such a way that both functional requirements and real-time attributes can be analyzed in control systems design. We also discuss an extension of SAVEComp with dynamic run-time property checking by utilizing run-time spare capacity that is normally induced by real-time analysis.

The remainder of the paper is organized as follows. Section 2 discusses background notions of our work by referring to control modules as a solution for an "easy-to-make" component-based design of control systems. In Section 3 the main features of SaveCCM are summarized. In Section 4 we first outline the overall structure of SAVEComp and then - by means of an explanatory example - we discuss its relevant aspects in more detail. Section 5 concludes and discusses future work.

2 Background: Control Modules

In Section 1, we said that many modern control systems are designed by using a modular approach in which its constituent function blocks are combined together.

Function blocks are very complex and have many configuration parameters because the rapid development of control algorithms has lead to a tremendous increase of the function block's functionalities. There are two main disadvantages due to the increased complexity of the function blocks. The first one is that there are a lot of parameters to be set and interface points to be connected and, hence, the developer should have a deep knowledge of the different function blocks. The second one is the obvious risk to make mistakes when the developer has to deal with a large amount of parameters and interface points. In [11], a component-based solution to overcame these disadvantages has been proposed. The main idea is to reduce the complexity of control systems by defining a standard interface for the signals between the building blocks.

In Figure 1.**A** we show an example of a control system made of a cascade control loop [10] where its building blocks are traditional function blocks. In Figure 1.**B** we show the same cascade control loop where its building blocks are connected by means of a graphical connection of ControlConnection type. Note that a control system is configured in a much simpler way if the blocks are connected with a ControlConnection structure.As showed in the figure, we will hereafter refer to the simpler configuration as the *top-level* design of the control system and to the other one as its *internal design*. In order to deal with connections of ControlConnection type, all the building blocks of the loop have to be able to transmit information forwards as well as backwards, with low delays. For this reason, in [11], the concept of control module has been introduced as a generalization of a traditional function block. The control module contains two parts of code for transmitting information forwards and backwards respectively. Although the control module concept considerably reduces the complexity of control loops by providing a component-based approach, current component technologies do not allow one to realize a control module in order to provide the developer with facilities for supporting predictability of the control system behaviour. This leads to a real need of a component-based approach for designing and composing control modules in such a way that such a support can be provided. Our aim is to provide a mean that will make it possible to use a component-based approach and predict the system behaviour.

3 The SaveCCM component model

In this section we briefly describe the main characteristics of our reference component model called SaveCCM. Refer to [6] for a detailed description of it.

The SAVEComp Component Model (SaveCCM) [6, 2] is a restrictive component model for control software development.

The interface of an architectural element is defined by a set of ports, i.e., points of interaction between the element and its environment. SaveCCM distinguishes between input and output ports, and there are two complementary aspects of ports: the data that can be transferred via the port and the triggering of component executions. SaveCCM distinguishes between these two aspects, and allows three types of ports: (i) *data-only* ports, (ii) *triggering-only* ports, and (iii) *data and triggering* ports. An architectural element emits trigger signals and data at its output ports, and receives trigger signals and data at its input ports. Systems are built by composing architectural elements. This composition is obtained by connecting input ports to output ports.

Since predictability and analysisability are of primary concern for the considered application domain, the SaveCCM execution model is rather restrictive. The basis is a control-flow (i.e., pipes-and-filter) paradigm in which executions are triggered by clocks or external events, and where components have finite, possibly variable, execution time. At the beginning a component is in an *idle* state where it waits for the activation of all its triggers. Once all component triggers have been activated, the component reads its input ports (*reading* state), performs its computations (*executing* state) based on the inputs read and its internal state, writes the result of the execution on its output ports (*writing* state) and finally goes back to the *idle* state. A list of quality attributes and (possibly) their value and credibility (i.e., a measure of confidence of the expressed value) is included in the specification of components and assemblies. Actions are abstract specifications of the externally visible behaviour of the component. Components are specified by their interfaces, behaviour and quality attributes.

A subset of the UML2 component diagrams[3] is adopted as graphical specification language[4]. The symbols showed in Figure **??** are used.

4 The SAVEComp development framework

In this section we outline the overall structure of the SAVEComp development framework[5] (see Figure 2). SAVEComp has been thought to be an extensible component-based development framework for design-time analysis (both functional and non-functional) and development of safety-critical embedded real-time systems. A part of it is the AutoComp technology [12] which is intended only for predicting the real-time behavior of the system.

As showed in Figure 2, SAVEComp can be described by distinguishing three main phases of its utilization. During design-time, developers may exploit the capabilities of SaveCCM [6, 2] to specify the top-level design of the control system by adopting a component-based software engineering process. Moreover, the extended version of SaveCCM allows the developer to enrich the system design with: (i) functional properties of the system expressed in terms of sequences of actions performed on component ports and/or possible values of data ports of interest for the analysis (e.g., the set of possible values of a data port expressing different op-

[3] UML2.0 specification - http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.

[4] In [6], the complete textual syntax (i.e., BNF specification) of the specification language is reported.

[5] The framework is under construction

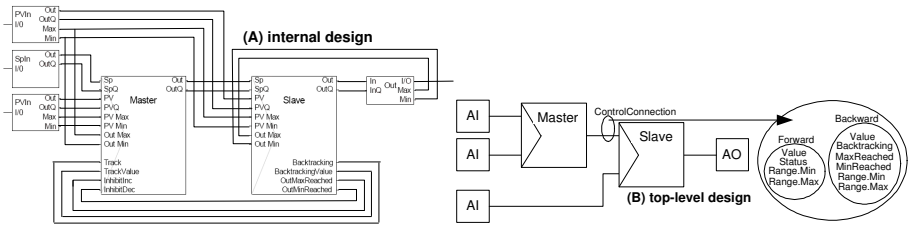

Figure 1. Two different designs of the same control system

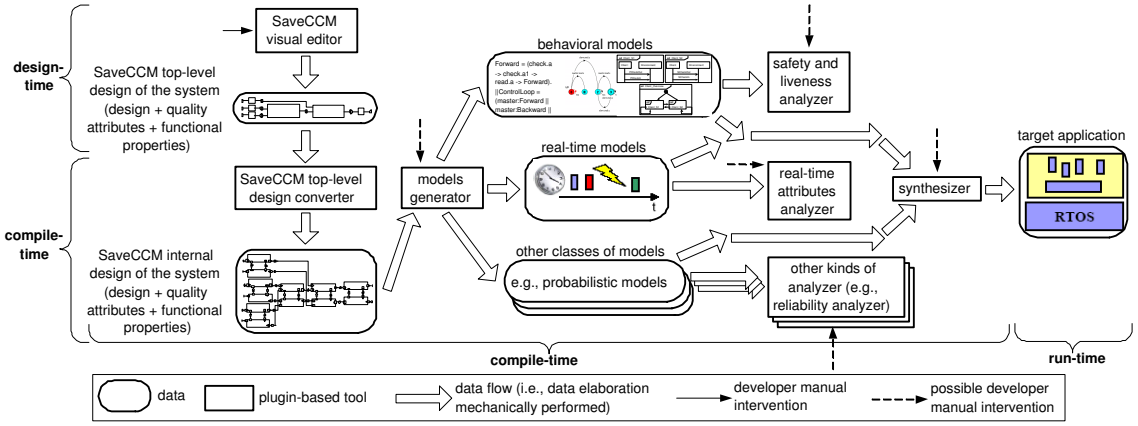

Figure 2. The SAVEComp development framework

erational modes of the control system); and (ii) high level temporal constraints in form of end-to-end deadlines and jitter supplied with their credibility values. During compile-time, SAVEComp automatically produces the SaveCCM internal design corresponding to the top-level and derives different views of the designed system intended to support both different kinds of specific functional/non-functional analysis and the mapping process to a real-time operating system (RTOS). In the figure, we show two possible classes of system views/models: (i) behavioral models (e.g., Process Algebras, LTSs, state machines, MSCs, UML2 interaction diagrams); and (ii) real-time models (e.g., Worst-case execution time analysis and Response-time analysis). The first class is intended to perform functional analysis (i.e., checking safety and liveness properties[6]), the second one to perform non-functional analysis in the specific case of guaranteeing real-time attributes. The plug-in based nature of SAVEComp allows us either to add new classes of system models - whenever it is needed to perform other specific kinds of analysis - or to extend an existent class to contain other model notations that are needed to support/integrate other processes for the same kind of analysis. For example, as sketched in the figure, we might need to add a probabilistic models view (e.g., Markov Chains, Stochastic Process Algebras) to perform reliability analysis by taking into account, e.g., the credibility value of each real-time attribute. Each specific kind of analysis/transformation is supported by a plug-in based tool within SAVEComp. Each "plug-in" might be either an existent tool suitably integrated with SAVEComp or built from scratch. By looking at the result of each particular analysis, the developer can either refine the top-level design since a functional or non-functional requirement has not been met or - if the design matches every requirement - execute a synthesis step. In each utilization phase, the developer has the possibility to interact with a particular plug-in based tool to set specific configuration parameters of it or to apply refinements (that are dictated by the analysis results) directly on the generated data/models rather than being forced to go back to the original design. We choose *Eclipse* platform[7] as implementation environment since it provides us with all the integration features we need to build SAVEComp. Eclipse facilitates the integration of different tools, that usually manipulate different content types. SAVEComp is built on a XML-based core which is the substrate providing an intermediate XML-based representation of system models that may work as a common ground to apply functional and non-functional analysis. To make SAVEComp as extensible as possible the XML core is kept general enough to allow its further extensions needed to manage new system model notations and new analysis processes and tools. In the reminder, we will only focus on the parts of SAVEComp that implement the approach presented in this paper. We consider the following SAVEComp plug-ins:

SaveCCM Visual Editor. A visual editor supporting the SaveCCM graphical specification language for designing the system architecture and for specifying functional properties and real-time attributes that must be analyzed. It is also responsible for generating XML code.

SaveCCM top-level (to internal) Design Converter. Control loops are often used to deliver the response for the time-critical computation as fast as possible. This plug-in is responsible for automatically deriving from the top-level design its corresponding internal design consistent of SaveCCM components, switches and their connections. Since top-level components, as SaveCCM assemblies, do not reflect the execution model of a basic component, this translation is required in order to perform functional and non-functional analysis of the system. The translation algorithm exploits the implicit internal structure and semantics of a ControlComponent and a ControlConnection.

[6] As usual, for safety and liveness we mean *nothing bad happens* and *something good eventually happens*, respectively.

[7] The Eclipse project. Eclipse platform technical overview. Technical report, 2001 - http://www.eclipse.org.

Functional behaviors Models Generator. A part of the *models generator* plug-in based tool. It is responsible for generating models of the functional behavior of the designed system. The kind of generated model (e.g., Process Algebras, LTSs, state machines, MSCs, UML2 interaction diagrams) depends on the XML template used. the model is generated by taking into account the system's internal design, the execution model of the SaveCCM components forming the system, the set of possible actions performable on a SaveCCM port and its possible values. Furthermore, a consistent model (with respect to both the notation used to model the system's functional behavior and the analyzer that will be used) of the functional properties is generated.

Safety and Liveness Analyzer. It is a plug-in based tool integrating an analyzer for each kind of model of the system's functional behavior that can be generated. In order to mechanically verify the specified safety and liveness properties. For example, the developer can verify that deadlocks do not occur or that the system always progresses (i.e., can every action eventually be performed?) or other specific functional properties of the system (e.g., a specific component must be disabled if the system is running in a specific operational mode).

Component to Task Converter. A part of the *models generator* plug-in based tool. In cooperation with the *Task Attribute Assignment*, it is responsible for generating a real-time model. The algorithm strives to reduce the number of operating system tasks by allocating components to the same task according to a set of rules.

Task Attribute Assignment. It is part of the *models generator* plug-in based tool. In cooperation with the *Component to Task Converter*, it assigning attributes considering platform and analysis goal.

Real-Time Analyzer. The analysis step is dependent on the underlying platform, e.g., schedulability analysis is limited to the algorithms available in the OS used. In the current prototype implementation, response-time analysis according to FPS theory is performed.

Code Synthesizer. The code generation module of the compile-time activities generates all source code that is dependent on the underlying operating system. Each operating system needs to have a transformation API where platform independent system calls can be translated to OS specific.

4.1 Extending SaveCCM to design and use control modules

The control module concept [11] can be implemented in SaveCCM by means of a new type of assembly which composes two components. We denote this new assembly type as "ControlComponent" type. One component within a ControlComponent is denoted as "Forward", the other one is denoted as "Backward". Forward and Backward are for transmitting information forwards and backwards (within a loop in a control system), respectively. In other words, Forward is responsible - given input values and taking into account the state of its ControlComponent - for calculating the output value of the ControlComponent. Analogously, Backward is responsible for updating the state of its ControlComponent depending on the feedback signals. Forward exports an interface made of input and output data-and-triggering ports and, possibly, other ports explicitly specified by the developer for specific purposes depending on the system functionality. The same is for Backward. ControlComponent, in turn, exports the same interface of Forward and Backward. As it is usual in SaveCCM, the ports of ControlComponent are connected to the corresponding ports of Forward and Backward through delegation. The type of a data transmitted through a port of the ControlComponent is a structured data type as defined by the ControlConnection structure. The triggering data are used for activating a Forward or Backward component depending on the control flow of the system. The information required to update the state of all the ControlComponents in a loop is not available until all the Forward components have executed their code. This is required for a correct functioning of the control system. Note that a ControlComponent can handle outer control loops as well as inner loops. These inner connections are internally generated - after the generation of Forward and Backward - by the *"SaveCCM top-level design converter"* (see Figure 2).

4.2 Analyzing functional requirements

In this section we formalize the execution model of a ControlComponent. This formalization is intended to support functional analysis of control systems during design-time. We are interested in proving safety and liveness properties. To formalize the execution model of a ControlComponent we look at (i) its internal design; (ii) the execution model of a SaveCCM component; (iii) the set of possible actions performable on a SaveCCM port and, in some cases[8], (iv) its possible values. By referring to Section 3, the execution model of a component may be expressed as a combination of actions that can be executed on its ports. The only action that can be performed on an input (output) data port is a reading (writing) action. We denote it as "read" ("write"). "read" and "write" are non-blocking actions (i.e., there will always be a value on a data port and it will always be possible to overwrite that value). On an input (output) triggering port we can perform a checking (activating) action that we denote as "check" ("activate"). "check" is a blocking action, that is it makes a component waiting for the activation of an input triggering port. "activate" simply activates the trigger associated to an output triggering port. On an input (output) data-and-triggering port a component executes "check" followed by "read" ("write" followed by "activate"). These rules can be combined in the obvious way in order to specify the execution behavior of a component, with an arbitrary number of ports of different type, by means of a process algebra. Note only that if a component C has $p_1, ..., p_n$ input data-and-triggering ports then - during the initial part of its execution - C will execute a sequence of n "check" (each of them for each p_i) followed by a sequence of n "read". We choose FSP [8] (*Finite State Processes*) as process algebra to model the execution behavior of components and assemblies at design level. FSP fits our purposes because it is notoriously easier to use than other more expressive process algebras and it is supported by LTSA [8] (*Labeled Transition System Analyser*). LTSA is a plug-in based verification tool for concurrent systems. It mechanically checks that the specification of a concurrent system satisfies required properties of its behavior. In addition, LTSA supports simulation to facilitate the interactive exploration of the system behavior. Thus the FSP specification of a SaveCCM system represents the mean to integrate SAVEComp with LTSA in order to support functional analysis. The FSP specification is mechanically derived by the *"functional behaviors model generator"* taking into account the loop's internal design, the execution model of a SaveCCM component and by combining the above mentioned rules (defining the set of actions that can be performed on a port) in the obvious way. Every functional property - that must be checked - has been included in the system top-level design (in a XML format) and it has been mechanically translated in the LTSA property notation by the *"functional behaviors model generator"*. Integrating SAVEComp with LTSA (i.e., a possible *"safety and liveness analyzer"*) allow us to easily verify functional properties of the system's FSP specification. For example, we can

[8]This is required only for specific data ports of interest, e.g., boolean data ports used to set different operational modes of the control system.

mechanically verify that deadlocks do not occur in the execution of the control system (i.e., safety).

4.3 Analyzing Real-Time properties

In this section we will discuss the non-functional model of SaveCCM. We will show how we can analyze SaveCCM considering real-time properties in an automated way, and discuss a *resource reclaiming*-extension to SaveComp that utilizes spare capacities introduced by pessimistic real-time predictions. Further, we will also discuss analysis techniques and synthesis. In order to reason about real-time behaviour we need to transform the design-time components into tasks conforming to a real-time model. The tasks can then be analyzed considering the design requirements. The process is performed in the steps:

Model transformation: Model transformation involves the steps *(i) component to task allocation* and *(ii) attribute assignment* which are necessary in order to transit from the component model, to a run-time model enabling verification of temporal constraints and usage of efficient and deterministic execution environments.

Real-Time Analysis: To show that the run-time tasks will meet their stipulated timing constraints, schedulability analysis must be performed. We assume a fixed-priority systems (FPS) (the predominant scheduling method in today's real-time operating systems).

Synthesis: Synthesis involves mapping the tasks to operating system specific entities, mapping data connections to an OS specific communication, generating glue code, compiling, linking and bundling the program code.

4.3.1 Model transformation

When designing a control system with components, the designer is not required to consider the schedulability of the system, but should rather focus on the functionality. The components should be annotated with non-functional information corresponding to the control performance, e.g., periods and jitter constraints. Transformation of components to tasks and scheduling the tasks on a real-time operating are automated processes.

In order to reason about, e.g., real-time each component must be annotated with appropriate quality attributes. These quality attributes are; A finite worst-case execution time (**WCET**). A nominal period (**T**), and in the case it is appropriate jitter constraint (**Jitter**). The SaveCCM model also has transactions that can be used for defining timing constraints of data and/or control paths. Transactions has end-to-end deadlines (**E2ED**), that define the longest allowed latency between two components in the system.

Components can be mapped to tasks in numerous ways and when constructing systems the developer is often required to manually set task attributes such as priorities. Since the priorities directly decides how the tasks are scheduled, this is a hard task. In our approach this process is automated in the task attribute assignment plug-in.

The context-switch time is increasing with the number of tasks, and the ideal mapping considering stack usage and task switch-overhead is to map all components to one task. However, in most cases this is not feasible due to the real-time constraints of the system.

A common approach to preserve the notion of components also after deployment is to use a one-to-one mapping between components and tasks, i.e., map one component to one task. However, the one-to-one mapping often implies worse resource usage than necessary.

In [5] a framework is proposed to facilitate the mapping between components and tasks by setting up mapping rules and exploit *Genetic Algorithms* (GA) to find feasible mappings that is optimized considering stack usage and context-switch overhead. This framework also constitutes the proposed plug-ins *Component to Task Converter*, *Task Attribute Assignment* and *Real-Time Analyzer*.

4.3.2 Real-Time Analysis

An important issue in obtaining high resource utilization is to deploy an efficient and tight schedulability analysis. The analysis need to faithfully model the complex execution behaviour that arises in control systems. Especially, the analysis should be able to handle arbitrary large jitter and deadlines , task synchronization and shared resources, and operating systems overhead.

For fixed-priority systems (the predominant scheduling method in today's real-time operating systems), the recent fast and tight response-time analysis (RTA) for tasks with offsets provides a suitable efficient and tight analysis [9]. This technique can model the precedence relations between tasks and hence gives a very accurate model of the system behaviour. Furthermore, the execution speed of this technique widely outperforms previous methods and is hence highly suitable for deployment in an optimization algorithm.

4.3.3 Synthesis

For synthesizing an assembly, platform specific API calls have to be inserted in the code. SaveCCM uses a general API and an API-translator (Code generator) The code generator resolves communication within and between tasks by translating platform-independent system calls with platform-specific system calls and adds platform specific glue code.

4.3.4 Resource Reclaiming Extension

Real-Time Analysis is based on worst-case behaviour in order to guarantee correct behaviour in all situations. Due to this, the analysis often becomes pessimistic because the worst-case scenario does not always reflect the actual case. Thus, when the worst case does not occur, there are left over resources in terms of processing time, i.e., residual time. The residual can be dynamically reclaimed and used for, e.g., dynamic property checking or other types of monitoring in low priority tasks.

The resource-reclaiming strategy is performed with an on-line service scheduler that uses hybrid scheduling to choose appropriate actions considering a residual time and. Low priority monitoring-tasks can use the residual time. The residual time can also be used for scheduling higher quality versions of the normal tasks as described in [4].

5 Conclusion and future work

Although component models that support predictability of the system behaviour there exist, they are found to be inappropriate for the control systems application domain since they do not support the requirements of embedded systems and, hence, are not able to predict the behaviour of control systems. The approach presented in this paper represents a possible solution to this problem. By means of it, we can build/compose control systems components (i.e., in designing the control system we can use a component-based approach by exploiting all its notorious advantages) and - in the same time - predict the functional/non-functional behaviour of the composed system. Although extending SaveCCM with the possibility to specify a top-level design of the system considerably simplify the developer tasks, it internally adds complexity at level of system implementation. To validate the real feasibility of our approach, as future work, we plan to apply SAVEComp to real-scale case studies. Moreover, SAVEComp, as it is currently structured, still lacks of integration between functional and non-functional analysis. That is, functional and non-functional analysis are separately performed.

We also plan to incorporate SAVEComp into TOOL•ONE framework [3] which supports functional and non-functional analysis integration, and implement the SAVEComp parts that go beyond the approach presented in this paper.

Acknowledgements

This work is supported by SSF within both SAVE (*SAfety critical components for VEhicular systems - http://www.mrtc.mdh.se/SAVE/*) and FLEXCON (*FLEXible embedded CONtrol systems - http://www.control.lth.se/FLEXCON/*) project.

6 References

[1] *International Electrotechnical Commission, IEC 61131 Programmable Controllers. Part 1 - 5*, January 1992.

[2] M. Akerholm, A. Möller, H. Hansson, and M. Nolin. Towards a Dependable Component Technology for Embedded System Applications. In *Proceedings of the 10th IEEE International Workshop on Object-oriented Real-Time Dependable Systems (WORDS05)*, February 2005. Sedona, Arizona, USA.

[3] V. Cortellessa, A. Marco, P. Inverardi, F. Macinelli, and P. Pelliccione. A framework for the integration of functional and non-functional analysis of software architectures. In *TACoS*, 2004.

[4] J. Fredriksson, M. Akerholm, K. Sandström, and R. Dobrin. Attaining flexible real-time systems by bringing together component technologies and real-time systems theory. In *Proceedings of the 29th Euromicro Conference, Component Based Software Engineering Track*, September 2003. Belek, Turkey.

[5] J. Fredriksson, K. Sandström, and M. Akerholm. Calculating resource trad-offs when mapping components to real-time tasks. In *In the 8th International Symposium on Component-Based Software Engineering (CBSE8), St.Louis, USA*, May 2005.

[6] H. Hansson, M. Akerholm, I. Crnkovic, and M. Törngren. SaveCCM - a Component Model for Safety-Critical Real-Time Systems. In *Proceedings of 30th Euromicro Conference, Special Session Component Models for Dependable Systems*, September 2004.

[7] B. Lewis. IEC 61499 Function Blocks: A new way to design control systems? *Control Engineering Europe*, April 2002.

[8] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.

[9] J. Mäki-Turja and M. Nolin. Fast and Tight Response-Times for Tasks with Offsets. In *17th EUROMICRO Conference on Real-Time Systems*. IEEE, July 2005. Accepted for publication.

[10] E. Parr. *Programmable Controllers - An Engineer's Guide (2nd Edition)*. Butterworth-Heinemann Ltd, 2001.

[11] L. Pernebo and B. Hansson. Plug and play in control loop design. In *Preprints Reglermöte 2002*, Linköping, Sweden, May 2002.

[12] K. Sandström, J. Fredriksson, and M. Akerholm. Introducing a component technology for safety critical embedded real-time systems. In *Springer - LNCS 3054*, May 2004.

[13] M. Tivoli, J. Fredriksson, and I. Crnkovic. A component-based approach for supporting functional and non-functional analysis in control loop design. *Malardalen University, Malardalen Real-Time Research Centre. Technical Report*, May 2005.