A Component-based Development Framework For Supporting Functional and Non-functional Analysis In Control System Design

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

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

Ivica Crnkovic
Mälardalen Real-Time Research Centre, Mälardalen University
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. However, the widely used commercial component technologies are unable to provide solutions to the requirements of embedded systems as they require too much resources 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. We discuss 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.

Categories and Subject Descriptors: D.2 [Software Engineering]: Miscellaneous

General Terms: Design, Algorithms.

Keywords: Real Time Embedded Systems, Control Systems, Functional Analysis, Non Functional Analysis.

1. INTRODUCTION

Due to the increasing complexity of control systems, they are often constructed performing a modular approach by means 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"* [7]. 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. 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. 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. 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* 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) [4] which is designed for safety-critical real-time systems. SaveCCM has been thought to support predictability of 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. 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.

This paper is a short version of an existent technical report [3] where all the details missing here (about SAVEComp and SaveCCM) are reported. In [3] we also report and compare related work and validate the approach presented in this paper by means of an industrial case-study, which is concerned with an adaptive cruise controller.

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

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 [7], a component-based solution to overcome 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 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 [7], the concept of control module has been introduced as a generalization of a traditional function block.

3. THE SAVECCM COMPONENT MODEL

The SAVEComp Component Model (SaveCCM) [4] is a restrictive component model for control software development. The interface of an architectural element is defined by a set of ports 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.

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. 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.

4. THE SAVECOMP DEVELOPMENT FRAMEWORK

In this section we outline the overall structure of the SAVEComp development framework[1] (see Figure 2).

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 [4] 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 operational 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, 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. 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. 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 as implementation environment since it provides us with all the integration features we need to build SAVEComp. Eclipse facilitates the integration of different tools,

[1]The framework is under construction

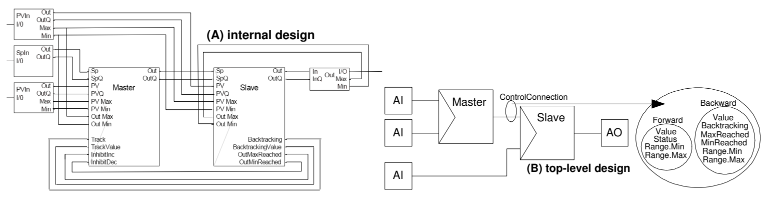

Figure 1: Two different designs of the same control system

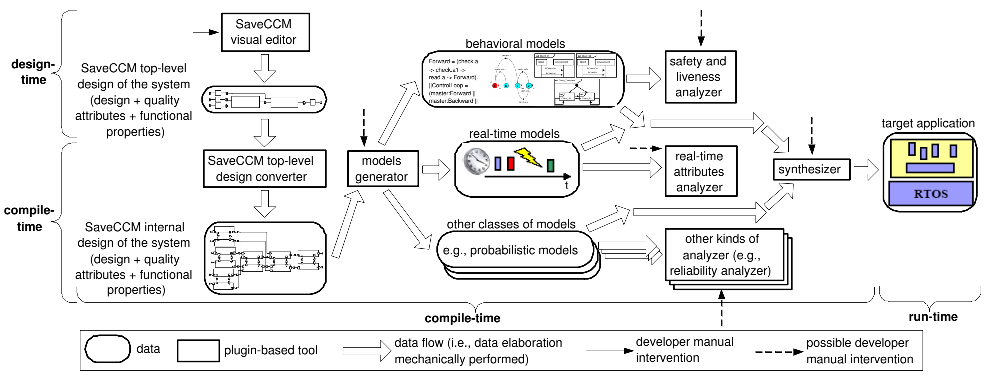
design-time
compile-time
SaveCCM visual editor
SaveCCM top-level design of the system (design + quality attributes + functional properties)
SaveCCM top-level design converter
SaveCCM internal design of the system (design + quality attributes + functional properties)
models generator
behavioral models
Forward = (check.a -> check.a1 -> read.a -> Forward).
||ControlLoop = (master:Forward || master:Backward ||
real-time models
t
other classes of models
e.g., probabilistic models
safety and liveness analyzer
real-time attributes analyzer
other kinds of analyzer (e.g., reliability analyzer)
synthesizer
target application
RTOS
compile-time
run-time
data
plugin-based tool
data flow (i.e., data elaboration mechanically performed)
developer manual intervention
possible developer manual intervention

Figure 2: The SAVEComp development framework

that usually manipulate different content types. In the reminder, we will only focus on the parts of SAVEComp that implement the approach presented in this paper. We consider the following plug-ins:

SaveCCM Visual Editor. A visual editor supporting the SaveCCM graphical specification language.

SaveCCM top-level (to internal) Design Converter. This plug-in is responsible for automatically deriving from the top-level design its corresponding internal design.

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 and of the functional properties that must be checked.

Safety and Liveness Analyzer. It is used 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.

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.

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

Real-Time Analyzer. The underlying platform, e.g., schedulability analysis is limited to the algorithms available in the RTOS 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.

4.1 Extending SaveCCM to design and use control modules

The control module concept [7] 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. The type of a data transmitted through a port of the ControlComponent is a structured data type as defined by the ControlConnection structure. 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 (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. This combination of possible actions can be described by means of a process algebra. We choose FSP [5] (*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 tool-supported, e.g., by LTSA [5] (*Labeled Transition System Analyser*). LTSA is a plug-in based verification tool for concurrent systems. 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"*. 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"*.

4.3 Analyzing Real-Time properties

To reason about real-time behaviour the design-time components have to be transformed into tasks conforming to a task-model. The tasks can then be analyzed considering the design requirements. The process is performed in the steps *(I) Model transformation* which involves *(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. *(II) Real-Time Analysis* for analyzing that the stipulated timing constraints are met. We assume a fixed-priority systems (FPS). *(III) Synthesis* which 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.

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 are 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. 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. However, the one-to-one mapping often implies bad resource usage. In [2] 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*.

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. For fixed-priority systems the recent fast and tight response-time analysis (RTA) for tasks with offsets provides a suitable efficient and tight analysis [6]. The execution speed of this technique widely outperforms previous methods and is hence highly suitable for deployment in an optimization algorithm.

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.

Resource Reclaiming Extension. Real-time analysis is based on worst-case behaviour in order to guarantee correct behaviour in all situations. Threfore the analysis often becomes pessimistic. Thus, as the worst case does not occur there are left over resources in terms of processing time (residual time). The residual time 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 as described in [1].

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. To validate the real feasibility of our approach, as future work, we plan to apply SAVEComp to real-scale case studies. Moreover, SAVEComp still lacks of integration between functional and non-functional analysis. We also plan to integrate them and implement the SAVEComp parts that go beyond the approach presented in this paper.

Acknowledgements

This work is supported by SSF within both SAVE and FLEXCON project.

6. REFERENCES

[1] 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 *Proc. of the 29th Euromicro Conference, CBSE Track*, 2003.

[2] J. Fredriksson, K. Sandström, and M. Akerholm. Calculating resource trad-offs when mapping components to real-time tasks. In *CBSE8*, 2005.

[3] J. Fredriksson, M. Tivoli, and I. Crnkovic. A component-based development framework for supporting functional and non-functional analysis in control systems design. Technical report, Tech. rep., Dep. of Computer Scienc and Electronics, Mälardalen University, 2005.

[4] H. Hansson, M. Akerholm, I. Crnkovic, and M. Törngren. SaveCCM - a Component Model for Safety-Critical Real-Time Systems. In *Proc. of 30th Euromicro Conference*, 2004.

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

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

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