# A Software Component Technology for Vehicle Control Systems

Mikael Åkerholm
*Mälardalen Real-Time Research Centre*
*Mälardalen University, Västerås, Sweden*
*CC Systems AB, Västerås, Sweden*
*mikael.akerholm@mdh.se*

## Abstract

*This research aims at developing a component technology for embedded control systems in vehicles. Such a technology would enable software engineers in the vehicular domain to practice component-based software engineering.*

## 1. Introduction

We addresses the problem of defining a component technology for embedded control systems in vehicles e.g., in passenger cars, trucks, and heavy vehicles. Control systems in vehicles are highly critical for the vehicles functionality, controlling, e.g., engine, brakes, and steering. They are characterized by high demands on safety, reliability, and hard real-time constraints. Furthermore, vehicles are produced in large volumes, meaning that the systems also must be cheap to produce. As a consequence software systems must be resource efficient, to allow the use of cheap hardware.

There are several challenges related to the problem, in an area relatively unexplored in comparison with component technologies for desktop- and web-applications. The existing commercial component technologies have been developed within the PC domain, which is a domain with requirements fundamentally different from those of vehicular systems.

In this report we describe our work so far, towards a component technology for embedded control systems in vehicles. Section 2, briefly describes our research focus by discussing the main questions. Section 3 introduces the SaveCCM Component Technology that is the result of the research. Conclusions and future work is discussed in section 4. More details are found in [1].

## 2. Research Focus

The main problem of component technologies for vehicular systems is broad. A component technology contains a *component model*, which defines a set of rules to be followed by users. It defines different component types that are supported by the technology, possible interaction schemes between components, and clarifies how different resources are bound to components. A component technology also contains a *component framework*, which provides the necessary run-time support for the components not provided by the underlying execution platform (i.e., operating system or similar). Furthermore a component technology often incorporates a set of *tools*, e.g., graphical modeling tools, compilers, component repositories, and analysis or testing tools.

To limit the research from all these different artifacts we have been addressing three main questions:

*Which quality attributes are the most important in the vehicular domain, and how do they relate to a component technology?*

Quality attributes define the quality of software. It is reasonable to expect that attributes can have different priorities in different software domains. The second part of the question addresses the relations between a component technology and the quality attributes that are important in the domain.

*How can a component model support predictability of important quality attributes and be suitable for expressing common functionality in vehicular control systems?*

This question calls for an approach by defining components and possibilities for component interaction, with respect to ease of implementing vehicular control systems, and support for prediction of quality attributes considered important in the domain

*How can an efficient utilization of resources be achieved in component based applications?*

By resources we mean shared limited run-time resources, e.g., processor and memory capacity. Resource-efficiency, (the consumption of a minimum of resources in achieving an objective) is a quality attribute that has received special attention in this work. This focus is based on the belief that poor resource efficiency is an

important reason for vehicular companies not choosing to utilize commercial component technologies.

## 3. Component Technology Overview

As shown in figure 1, the SaveCCM technology can be described by distinguishing three main phases of its utilization, *design-time- compile-time*, and *run-time*. The following sub-sections will describe this three phases.

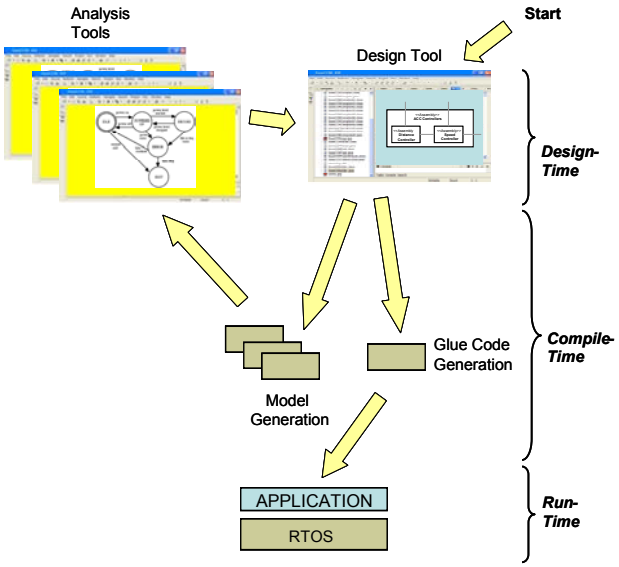


**Figure 1. Component Technology Overview**

### 3.1. Design-Time

During design-time developers use a component-based strategy, supported by the SaveCCM component model. The component model allows connecting components and expressing high level constraints.

The architectural elements are *components, switches,* and *assemblies*. Components are the basic units in a design, and encapsulate a portion of functionality. Switches are special components used to (re)configure component interconnections. Assemblies represent sub-systems and are aggregated behavior from a set of components, switches, and possibly other assemblies.

The component model has been designed to easily express common functionality in vehicular systems. Some specific examples of key functionality are: feedback control, system mode changes, and static configuration for product-line architectures.

### 3.2. Compile-Time

During compile-time, a set of tools are used to automatically produce necessary low level code for the run-time system (i.e. glue-code), and different specialized models of the application for analysis tools, e.g., finite state processes, timed-automata, and fixed priority scheduling models.

All low level code (i.e., hardware and operating system interaction) is automatically generated, meaning that components contains no dependencies to the underlying system and can be transferred between different execution platforms. Furthermore, the code generation step statically resolve resource usage and timing, with the strategy to resolve as much as possible during compile-time instead of depending of costly algorithms during run-time.

### 3.3. Run-Time

During run-time efficiency and predictability are achieved by systematic use of efficient and analyzable run-time mechanisms, provided by a fixed priority real-time operating system.

## 4. Conclusions and Future Work

We have presented our prototype component technology for vehicular software.

The key concept is clear distinctions between design-time, compile-time, and run-time. The compile-time techniques enable a component-based approach during design-time, combined with resource effective run-time models of real-time operating systems, by statically resolve resource usage and timing during compilation. It also enables automated analysis, and platform independent software components.

We have conducted initial evaluation of our prototype component technology in cooperation with industry, which indicates that the component technology is promising but need further development to be applicable in an industrial context.

In future work we will incorporate more supporting tools, e.g., for graphical modelling, configuration management, and analysis. New mechanisms like databases for structured handling of shared data, and run-time monitoring and test support are also targets for future work.

We will also continue to investigate other possible definitions of component technologies and component models, within the automotive domain as well as expand to a broader scope of embedded systems.

## References

[1] Mikael Åkerholm, A Software Component Technology for Vehicle Control Systems, Licentiate Thesis, Mälardalen University Press, February, 2005