Beating the Automotive Code Complexity Challenge

Components, Models and Tools

Hans Hansson, Mikael Nolin and Thomas Nolte
Mälardalen Real-Time Research Centre, Västerås, SWEDEN

I. Problem

The last decade, cars and other vehicles have become increasingly dependent on software to realize novel and innovative functions. The time when mechanics and electronics accounted for the majority of value growth in the automotive industry is long gone - today the majority of inventions and value growth comes from functions realized with software.

This sudden shift in technology from mechanics/electronics to software has created a whole new set of problems for the automotive industry. A modern high-end car can contains more than 2.000.000 lines of code, distributed over 80 nodes, using 5 different networks. Development of a computer system of this magnitude is a daunting task regardless of commercial sector. For the automotive industry exceptional requirements regarding safety, reliability, serviceability, platform reuse, time-to-market, product sustainability, and more, makes the task almost infeasible. Add to this gloom picture the fact that the achievements within the general software-engineering community are seldom applicable to the safety critical, real-time, resource constrained environments of the automotive electronic platforms. To summarize state-of-affairs we conclude that automotive software, despite being one of the most safety critical and value adding software sectors, are currently developed using inadequate tools, techniques and processes.

II. Solution

To remedy these ills, the software complexity must be simultaneously tacked from four different perspectives:

- **Design** - We must find proper abstraction mechanisms to allow the designer to model functional and non-functional properties and structure of the software.
- **Analysis** - Using the design models we need to be able to predict behaviors and properties of the design. Analysis tools should give feedback to the developer and provide information to be used by synthesis tools.
- **Synthesis** - Advanced compilation techniques and code generation needs to be developed, in order to map abstract model-behaviors to executable entities. Mappings need to result in efficient resource utilization and predictable execution patterns.
- **Runtime system** - Novel runtime mechanisms to execute, efficiently and predictable, the automatically generated code from the synthesis step are needed. Many of the primitives of contemporary runtime systems will become obsolete and new mechanisms will be needed.

The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

III. The PROGRESS approach

PROGRESS is a national Swedish research centre (5 year initiative, ∼$7.5M, ∼30 people) working on component based software engineering with a particular focus (and history) on automotive and vehicular applications. PROGRESS' hypothesis is that building embedded software (and systems) from reusable components allows for a more cost efficient and scalable handling of complexity, integration and quality assurance. PROGRESS research aims to provide a mature engineering discipline for development of embedded software. This includes theories, methods, and tools for (i) predictable embedded-software development from software components and legacy code, (ii) interfacing components with the underlying platform, and (iii) adopting and applying real-time modeling and analysis techniques across all stages of the component-based design and development chain. Specifically, PROGRESS approaches the code complexity problem in the following concrete ways:

- **Design** - We propose the use of software components and model based development. The components are carrier of functionality, requirements and models. Components can be assemblies to perform advanced functions and their models can be analyzed so the important non-functional properties can be verified against the requirements.
  In order to support the needs of safety, predictability and resource efficiency we propose the use of design-time component models. Contrary to popular component models, such as .NET and Corba Component Model, a design-time component model focuses on design-time composition and design-time analysis of components and assemblies. The goal is to make a priori predictions of system properties and by using advanced synthesis technologies to pre-run time construct a highly efficient and predictable executable.
- **Analysis** - Analysis techniques in PROGRESS extend beyond the commonplace techniques, used in e.g. UML-tools, to validate completeness and connectedness of the design. We also include prediction (or bounding) of runtime properties such as end-to-end delays, memory consumption, deadlock absence, and more. These tools take their input from the design model and the models associated to the components used. From these models, analysis-specific models can be derived for different types of analysis (e.g. timing models for schedulability analysis and state machines for deadlock-analysis). The analysis tools are used both by (1) the developer and (2) by the synthesis tools. (1) The developer directly uses analysis to establish feasibility of the design before commencing with realization of the system and to explore different design decisions. (2) During synthesis many potential configurations of the runtime system can be explored; the analysis tools are then be used to evaluate configurations for feasibility and/or cost in order to generate a system that fulfils all requirements and possible some optimization criteria.
- **Synthesis** - The synthesis tools fulfill a series of functions. First, they generate glue code to tie components together. In our design-time component technology, components themselves cannot bind to each other. Instead it's the role of the compiler to create bindings for runtime. Second, the tool removes any unused functionality. Since components may be designed for reuse, it is likely that they contain functionality that is not used in a particular context. To preserve resource and increase predictability, unused portions of components should be removed. Third, in order to further optimize the system, components should be mapped to runtime objects in an efficient manner. This includes coalescing as many components as possible into each task and as many data-items as possible into each message. Fourth, selection of runtime parameter for runtime objects is performed. This includes activities as assigning priorities and/or deadline to tasks, assigning identifiers to messages, assigning ceiling values to shared resources, allocating memory to stacks, etc.

- **Runtime system** - The runtime system need to support the abstractions used by the synthesis tools. Furthermore, it needs to support runtime monitoring and control of the different functions in the vehicle. Both automated and manual surveillance needs to be supported. Automated surveillance include making sure that subsystems does not consume more resources than allocated (in order not to interfere with other subsystems), and various logging and error reporting. Manual surveillance includes monitoring of important data-items both internally in nodes and transmitted over the networks. Ideally the runtime system should also support runtime intervention of test-engineers, allowing them to change data- and control-flow.

IV. About the authors

Hans Hansson (born Aug 8, 1957) is professor in Computer Engineering, specializing in real-time systems, at Mälardalen University since 1997. He is director of research at the School of Innovation, Design and Engineering, Mälardalen Real-Time Research Centre, the PROGRESS national strategic research centre, and the national graduate school SAVE-IT. He received an M.Sc. (Engineering Physics), a Licentiate degree (Computer Science), a B.A. (Business Administration), and a Doctor of Technology degree (Computer Science) from Uppsala University, Sweden, in 1981, 1984, 1984 and 1992, respectively. He is associate editor of the IEEE Transactions on Industrial Informatics and Kluwers Journal of Real-Time Systems.

Hans Hansson, MRTC/Mälardalen University, P.O. Box 883, SE-72123 Västerås, SWEDEN
email: hans.hansson@mdh.se, tel: +4621103163

Mikael Nolin is a professor in real-time systems, specializing in vehicular software. Mikael received his PhD in 2000 from Uppsala University, Sweden. Since then he has been working in both academia and in industry with embedded systems, real-time systems, and embedded communications. He is currently sharing his time between the company CC-Systems, where he is working with embedded communications solutions and architectures for distributed systems, and Mälardalen Real-Time Research Centre (MRTC) where he is focusing his research on new methods to construct software for the vehicular industry.

Mikael Nolin, MRTC/Mälardalen University, P.O. Box 883, SE-72123 Västerås, SWEDEN
email: mikael.nolin@mdh.se, tel: +46702882829

Thomas Nolte received his B.Eng., M.Sc., Licentiate, and Ph.D. degree in Computer Engineering from the Department of Computer Science and Electronics at Mälardalen University (MDH), Sweden, 2001, 2002, 2003, and 2006 respectively. Currently, Thomas holds a position as the Project Leader of the PROGRESS Centre for Predictable Embedded Software Systems at Mälardalen Real-Time Research Centre (MRTC), at MDH. Moreover, he is working as a Researcher in the Real-Time Systems Design group, active in the HISCORE project, several PROGRESS projects and the SAVE++ project.

Thomas Nolte, MRTC/Mälardalen University, P.O. Box 883, SE-72123 Västerås, SWEDEN
email: thomas.nolte@mdh.se, tel: +46708657551