COMPONENT BASED SOFTWARE DEVELOPMENT FOR EMBEDDED SYSTEMS

Bálint RÁKOSI
Advisor: András PATARICZA

I. Introduction

Recent embedded software development techniques tend to reach the limits of their scalability. For instance, the next generation of embedded automotive electronics will have several hundred of devices consisting total of 1 GB of code. Our environment will be dominated by ambient intelligent devices ranging from nanoscale devices up to the talking washing machine. Future embedded systems will coexist and cooperate in a complex and heterogenic system, so problems of software complexity will exceed the problems related to hardware design. Component-based development is a well proven method for hardware design, therefore its applicability for embedded software development should be examined.

II. Main Idea

Reuse of software is one of the most important aspects when developing new applications [1]. Component integration ensures a high level of IP reuse, therefore Component Based Software Development (CBSD) is the dominant design paradigm for desktop environments, enterprise systems, and web-based, distributed applications. However all basic component integration focuses only at the functional aspects of the system. The development of software for embedded systems necessitates the consideration of all the frequently complex non-functional and resource constrains in addition to the pure software quality aspects. A uniform specification and validation paradigm for software components covering non-functional constrains as well as functional aspects becomes to a vital factor from the point of view of the product quality.

III. Core Concepts

The first definition of a component [2] defines it as an executable entity, which can be composed and deployed at runtime. Accordingly, component-based software architecture defines a set of components, a set of component interfaces and their mutual interaction. A component could be seen as fragment of source code which can be derived from a high level abstract language and allows either compile or build time composition, or both. The component interface must include in case of embedded systems non-functional proprieties in addition to functional ones in order to support early, conceptual verification at the system design level.

The most pragmatic solution for small embedded systems suffering typical of a lack of resources is the exclusive use of design time composition and a complete avoidance of run time integration. This way interconnection of components can bee translated directly to function calls instead of using resource and power consuming dynamic event notifications.

Large-scale embedded systems with sufficient resources may rely on a reduced version of the component model on top of a real time operating system.

IV. Requirements

The most important requirement of embedded systems from the point of view of CBSD is the compliance to the required non-functional and extra-functional proprieties:

- Real Time Guarantees: violation of timing requirements even of in the case of a functionally correct reaction prohibits proper system functionality (latency, execution time, etc.)
- Dependability: represents the propriety of the system to function even in case of failures (avoidance of faults, fault tolerance) as characterized by other attributes like reliability, availability.
- Resource constraints: limitations in available resources like energy, computing power etc.
- Life-cycle properties: the system must be able to handle several generations of hardware and software technologies during its lifetime.

The development and maintenance of an embedded system requires a significant effort from the system designer as well as from the system programmer. The use of specification design languages at a high level of abstraction like UML complemented by techniques like Meta-Modeling, Model driven Architecture (MDA) [3], and Model Driven Software Development (MDSD) [4] lead to reduced costs both during system development and product life-cycle maintenance.

V. Component based modeling and design

An already proven method is presented in [5] to develop or design component-based software architectures. The system is partitioned to components and containers where the focus is on the containers which are responsible for aspects like: scheduling, interrupt handling, remote communication, generic driver interface, resource control, advanced error detection, meanwhile the components will contain only the functional aspects, like the application logic,. The source code of the components as well those of the containers can be generated from the abstract languages by the methods mentioned earlier.

VI. Challenges and future work

Currently we are developing a model-based approach for component-based software development, using tools openArchitectureWare [6] for code generation and Enterprise Architect [7] for modeling interfaces and complex data types.

OpenArchitectureWare uses an explicitly programmed metamodel (metaclasses describing the metamodel) implemented in Java. Additional templates (and extensions) define the mapping from the metamodel to the output source code. The Enterprise Architect is an arbitrary XML-based modeling tool. The output of the generator is the source code (skeletons) for the components, the complete container implementations, as well as a make file controlling the compilation and build process of the container and the components.

References

[1] Ivica Crnkovic *Component-based Software Engineering for Embedded Systems,* International Conference on software engineering, ICSE'05 ,ACM, St. Luis , USA, May, 2005

[2] C. Szyperski. Component Software: *Beyond Object-Oriented Programming. Second edition*, ACM, Press and Addison- Wesley, New York, N.Y., 2002.

[3] www.omg.org

[4] Tom Stahl, Markus Völter *Model-Driven Software Development Technology*, Engineering, Management 2006

[5] Markus Völter *Model-Driven Development of. Component Infrastructures. for Embedded Systems*. Version 2.0, Sept 29, 2004. Submission to MBEES Workshop 2005.

[6] http://www.openarchitectureware.org/

[7] http://sparxsystems.com.au/products/ea.html