Component-based Software Engineering for Embedded Systems

Ivica Crnkovic
Mälardalen University, Department of Computer Science and Electronics
Box 883, 721 23 Västerås, Sweden
+46 21 1031 83
ivica.crnkovic@mdh.se, http://www.idt.mdh.se/~icc

ABSTRACT

Although attractive, CBD has not been widely adopted in domains of embedded systems. The main reason is inability of these technologies to cope with the important concerns of embedded systems, such as resource constraints, real-time or dependability requirements. However an increasing understanding of principles of CBD makes it possible to utilize these principles in implementation of different component-based models more appropriate for embedded systems. The aim of this tutorial is to point to the opportunity of applying this approach for development and maintenance of embedded systems. The tutorial gives insights into basic principles of CBD, the main concerns and characteristics of embedded systems and possible directions of adaptation of component-based approach for these systems. Different types of embedded systems and approaches for applying CBD are presented and illustrated by examples from research and practices. Also, challenges and research directions of CBD for embedded systems are discussed.

Categories and Subject Descriptors
D.2.2[**Software Engineering**]: Design Tools and Techniques

General Terms: Design, Reliability.

Keywords
Component-based software engineering, embedded systems.

1. INTRODUCTION

Component-based development (CBD) is of great interest to the software engineering community and has achieved considerable success in many engineering and application domains. CBD has been extensively used for several years in desktop environments, office applications, e-business and in general in Internet- and web-based distributed applications. In many other domains, for example embedded and real-time systems, CBD is utilized to a lesser degree. It has been experienced that it is difficult to use the same component technology in different domains because of different system requirements and constraints. The latest trends show that different component technologies are being developed for different domains. Similarly to object-oriented paradigm that is exploited in different OO languages, a component-based paradigm based on certain common principles is slowly built and used in different component technologies [1,2,3].

The aim of this tutorial is give an overview of principles of component-based software engineering and their utilization in development of embedded systems. The intention is to show that the component-based approach can successfully be used in development of embedded systems although the different concerns, requirements and limitations are valid then for systems that successfully have used CBD. Direct use of general-purpose component-based technologies is usually not feasible; rather specific adoptions are required, or new component models that explicitly address the main concerns of embedded systems must be developed.

2. EMBEDDED SYSTEMS

Embedded systems are computer systems that are part of larger systems and they perform some of the requirements of these systems. Some examples of such systems are automobile control systems, industrial processes control systems, mobile phones, or small sensor controllers. Embedded systems cover a large range of computer systems from ultra small computer-based devices to large systems monitoring and controlling complex processes. The overwhelming number of computer systems belongs to embedded systems: 99% of all computing units belong to embedded systems today [4].

Most of such embedded systems are also characterized as real-time systems, which means that the real-time properties such as response time, worse case execution time, etc., are important design concerns. These systems usually must meet stringent specifications for safety, reliability, availability and other attributes of dependability. Due to small size and requirements for mobility, but also extremely low production costs these systems require small and controlled resource consumption, and have limited hardware capacity [5]. The increased complexity of embedded real-time systems leads to increasing demands with respect to requirements engineering, high-level design, early error detection, productivity, integration, verification and maintenance, which increases the importance of an efficient management of life-cycle properties such as maintainability, portability, and adaptability.

3. BASIC CONCEPTS FOR COMPONENT-BASED EMBEDDED SYSTEMS

In classic engineering disciplines a component is a self-contained part or subsystem that can be used as a building block in the design of a larger system. In CBSE, there are many different suggestions for a definition of components. The best understanding of component in the software industry world is based on Szyperski's definition [2]. From this definition it can be assumed that a component is an executable unit, and that deployment and composition can be performed at run-time. In the domains of embedded systems this definition is largely followed; this is in particular true for the separation between component implementation and component interface. However the demands on the binary or executable from is not directly followed. A component can be delivered in a form of a source code written in a high-level language, and allows build-time (or design-time) composition.

The component interface summarizes the properties of the component that are externally visible to the other parts of the system. As for embedded systems extra-functional properties are as important as functional there is a tendency to include specification of extra-functional properties in the component interface (for example timing properties). This allows more system properties to be determined when the system is designed, i.e. such interface enables verification of system requirements and prediction of system properties from properties of components.

In general-purpose component technologies, the interfaces are usually implemented as object interfaces supporting polymorphism by late binding. While late binding allows connecting of components that are completely unaware of each other beside the connecting interface, this flexibility comes with a performance penalty and increased risk for system failure. Also the predictability of the system's performance or other properties decreases since the composition of the components occurs at run-time. Therefore the dynamic component deployment is not feasible for small embedded systems.

Due to the constraints for real-time and limited resources there are several reasons to perform component deployment and composition at design time rather than run-time: This allows composition tools to generate a monolithic firmware for the device from the component-based design and by this achieve better performance and better predictability of the system behavior. This also enables global optimizations of a static component composition, connections between components could be translated into direct function calls instead of using dynamic event notifications and verification and prediction of system requirements can be done statically from the given component properties. A characteristic example of a component model fro embedded systems is Koala developed and used at Philips [3].

For large embedded systems the resource constraints are not the primary concerns. The complexity and interoperability play much more important role. Also due complexity the development of such system is very expensive and cutting the development costs is highly prioritized. For this reason general-purpose component technologies are more interesting than in a case for small systems. The systems using these technologies belong to the category of soft-real time systems. Often a component technology is used as a basis for additional abstraction level support, which is specified either as set of standards or proprietary solutions. One successful example of adoption of a component-based technology is the initiative OPC Foundation (OLE process control Foundation), an organization responsible for a specification that defines a set of standard interfaces for process automation based upon OLE/COM and recently .NET technology. Another example of a component-based approach is development and use of the standard IEC 61131 [7]. IEC 61131 defines a family of languages in which function blocks can be viewed as components and interfaces between blocks are released by connecting in-ports and out-ports.

Large embedded systems that must fulfill hard real-time requirements usually do not use general-purpose component-based technologies. However in some cases, a reduced version of a component model is used on a top of a real-time operating system.

4. THE NEEDS AND PRIORITIES IN RESEARCH

In order to fully utilize advantages of CBD for the embedded systems, there are needs for a number of issues that must be solved or improved [4]. Today there is a lack of widely adopted component technology standards which are suitable for embedded systems. For smaller-size embedded systems, it is important that a system composed of components can be optimized for performance and memory consumption, and such a support is still missing in most of the component technologies available today. Most current component technologies do not support specification and analysis of extra-functional properties important for embedded systems. In particular composition theories of certain extra-functional properties and predictability of system properties from component properties must be developed [6]. Further, in order to support more advanced features in component-based embedded systems, the run-time platform must provide certain services, which however must use only limited resources. There are a number of other issues such as platform and vendor independence, component certification, component noninterference and similar, which must be solved for a successful adoption of component technologies. Finally, tools that support all the features lists above must be implemented.

5. REFERENCES

[1] Ivica Crnkovic and Magnus Larsson (editors), *Building Reliable Component-Based Software Systems*, Artech House Publishers, ISBN 1-58053-327-2

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

[3] R. van Ommering, F. van der Linden, and J. Kramer. The *Koala component model for consumer electronics software*. IEEE Computer, 33(3):78–85, March 2000.

[4] Roadmaps on Selected topics in Embedded Systems Design from the EU IST ARTIST project, http://www.artist-embedded.org/Roadmaps/index.html

[5] Ivica Crnkovic, *Component-based approach for Embedded Systems*, Ninth International Workshop on Component-Oriented Programming, Oslo, June 2004.

[6] Ivica Crnkovic, Magnus Larson, *Classification of quality attributes for predictability in component-based systems*, DSN 2004 Workshop on Architecting Dependable Systems Florence, Italy, June 2004. IEEE

[7] IEC. Application and implementation of IEC 61131-3. Technical report, IEC, Geneva, 1995.