Component-based approach for embedded systems

Ivica Crnkovic
Mälardalen University, Department of Computer Science and Engineering,
Box 883, 721 23 Västerås, Sweden
ivica.crnkovic@mdh.se, www.idt.mdh.se/~icc

Abstract: This paper addresses component-based approach for embedded systems. Due to the specific characteristics of embedded systems the general-purpose component technologies such as COM, .NET, or EJB have not been the most appropriate choices for their development. Although attractive, component-based approach has not been successful in this domain as in other domains. However in recent years the interest for component-based approach in embedded systems increases. The experience has shown that existing technologies cannot be used, or at least used directly. On the other hand an increasing understanding of principles of component-based approach makes it possible to utilize these principles in implementation of different component-based models, more appropriate for embedded systems. This paper gives an overview of basic characteristics of embedded systems, their requirements and constraints, and the implications to component models for these systems.

1 Introduction

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: 98% of all computer systems belong to embedded systems today. IEEE has the following definition for embedded systems:

An Embedded Computer System: A computer system that is part of a larger system and performs some of the requirements of that system; for example, a computer system used in an aircraft or rapid transit system. (IEEE, 1992).

Most of such embedded systems can also be characterized as real-time systems, (i.e., systems in which the correctness of the system depends not only on the logical result of the computations it performs but also on time factors [9]). Embedded real-time systems contain a computer as a part of a larger system and interact directly with external devices. They must usually meet stringent specifications for safety, reliability, limited hardware capacity etc. 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.

Component-based development is an attractive approach in the domains of embedded systems. In particular for the development of many variants of products the component-based approach is attractive. In spite of this attractiveness the adoption of component-based technologies for the development of real-time and embedded systems is significantly slower. Major reasons are that embedded systems must satisfy requirements of timeliness, quality-of-service, predictability, that they are often safety-critical, and can use severely constrained resources (memory, processing power, communication). The widely used component technologies such as EJB, .NET, CORBA component models are inherently heavyweight and complex, incurring large overheads on the run-time platform; they do not in general address timeliness, quality-of-service or similar extra-functional properties that are important for real-time systems. In their present form they start to be deployed in large, distributed, and not safety critical systems, e.g., in industrial automation, but are not suitable for deployment in most embedded real-time environments.

This paper gives a short overview of basic characteristics of embedded software and the problems and challenges the developers met. Further it describes a current state of the practice in different industries and research trends in applying component-based principles for embedded systems. The paper is based on work and results of several research projects, EU IST ARTIST[1], CBSENet Network[2] Swedish SAVE[3] project, and on communication and cooperation with several companies such as ABB, Volvo, Ericsson and Philips.

The rest of the paper is organized as follows. Section 2 gives and general characteristics and the most important requirements of embedded systems. Section 3 gives some examples of embedded systems in which a component-based approach has been used or has recently being developed. Chapter 4 lists the basic characteristics of component-based approach for embedded systems and finally chapter 5 summarizes the current challenges and need for further research.

2 Specific requirements of embedded systems

In most of the cases embedded systems are real-time systems. In many cases embedded systems are safety or mission critical systems. Embedded systems vary from very small systems to very large systems. For small systems there are strong constrains related to different recourses such as power or memory consumption. For these as well as for large embedded systems the demands

[1] http://www.systemes-critiques.org/ARTIST
[2] http://www.cbsenet.org
[3] http://www.mrtc.mdh.se/SAVE

on reliability, robustness, availability and other characteristics of dependable systems are important. Finally, in many domains, the product life cycle is very long – in can stretch to several decades.

All these characteristics have strong implications on requirements. Most of the requirements of embedded systems are related to non-functional characteristics (better designated as extra-functional properties or quality attributes). These properties can be classified in run-time and life cycle extra-functional properties. The most important are:

- *Real-time properties*. The real-time system functions are time-related; a violation of time requirements even of a proper functional response violates the system functionality. There are numbers of real-time properties: Response time or latency, execution time, worst case execution time, deadline etc.
- *Dependability.* Dependability is defined as an ability of a system to deliver service that can justifiably be trusted and an ability of a system to avoid failures that are more severe and frequent than is acceptable to the users. The main means to attain dependability are related to avoidance of faults: fault prevention, fault tolerance, fault removal and fault forecasting [1]. Dependability is characterised by several attributes and according to [1], these are the following: Reliability, availability, integrity, safety, confidentiality and maintainability. All these attributes are run-time properties except maintainability. These attributes are combination of several properties. For example availability is dependent on real-time properties, but also on reliability and even maintenance; safety is strongly related to reliability, etc.
- *Resource consumption.* Many embedded systems have strong requirements for low and controlled consumption of different resources. The reasons may be the size of the systems and/or the demands on lower production costs. Examples of such restrictions and constraints are power and memory consumption, execution (CPU) time, computation (CPU) power, etc.
- *Life cycle properties.* In general embedded systems are tightly coupled with their environment and the absence of their services can have large consequences on the environment. In many domains the embedded systems have very long life time running round the clock year after year. During the lifetime of a system several generations of hardware and software technologies can be used. The long life systems must be able to cope with these changes introduced either into the surrounding environment or into the systems themselves.

We can conclude that many of the most important requirements of the embedded systems are related to extra-functional properties. This has an implication that development and maintenance of such systems are very costly. In particular activities related to verification and guaranteed behavior (formal verification, modeling, tests, etc.) and maintenance (adaptive maintenance, debugging, regressive testing, etc.) require a lot of efforts. For these reasons the technologies and processes that lead to lower costs for these activities are very attractive and desirable.

3 State of the practice and experience for Embedded Systems

Embedded systems comprise a scale from ultra small devices with simple functionality, through small systems with sophisticated functions, to large, possibly distributed systems, where the management of the complexity is the main challenge. Further we can distinguish between systems produced in large quantities, in which the low production costs are very important and low-volume products in which the system dependability is the most important feature. All these different requirements have impact on feasibility, on use, and on approach in component-based development. A common characteristic of all systems is increasing importance of software. For example, software development costs for industrial robots make today about 75% of total costs, while in car industry it is today about 30%. Some ten, fifteen years ago this number was about 25% for robots and neglectable for cars. A second common characteristic is increasing interoperability. While previously the embedded systems were mainly used for controlling different processes today they are integrated with information systems of infotainment technologies.

In this section we give a short overview of some typical embedded systems.

3.1.1 Automotive Industry

Cars are typically manufactured in volumes in the order of millions per year. To achieve these volumes, and still offer the customer a wide range of choices, the products are built on platforms that contain common technology that have the flexibility to adapt to different kinds of cars by adding different components or different variants of the components.

The components are to a large extent provided by external suppliers, who work with many different car companies (or OEMs, original equipment manufacturers). The role of the OEM is thus to provide specifications for the suppliers, so that the component will fit a particular car, and to integrate the components into a product. Traditionally, suppliers have developed physical parts, but in modern cars they also provide software.

Within the automotive industry, the component-based approach has a relatively long tradition, as these systems are typically built from system components that are either developed in-house or provided by external suppliers. Today, the entire control system of an advanced car includes a number of Electronic Control Units (ECUs) equipped with software that implements vehicle functions. ECUs are treated as system components that can be developed and build independently of each other and of the entire system. A system control architecture is shown in Figure 1. The ECUs are connected to the system (the

car) through sensors and actuators and between themselves via one or several buses. Usually the buses are integrations points and their protocols specify the communications between the ECUs.

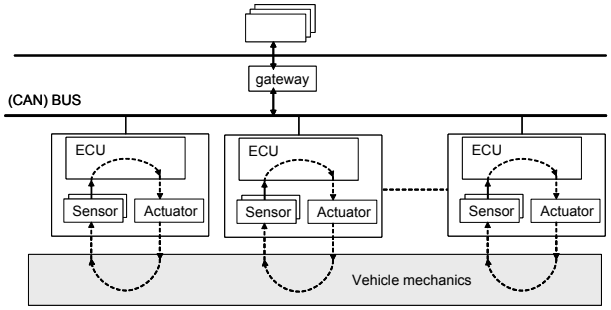

Figure 1. Component-based architecture of vehicular systems

Modern vehicular systems contain several tens of computer nodes. As the number of ECUs increases, the entire system becomes more complex. This requires sharing different types of resources (sensors, actuators, time, communication, memory, and CPU consumption). With increasing complexity, system reliability and safety become major problems. A satisfactory handling of safety-critical functions, such as emerging brake and steer-by-wire systems, require the integration of methods for establishing components and compositions of different aspects: functional and temporal correctness, safety and reliability, etc.

Today ECUs include proprietary software, mostly owned by subcontractors. This makes the entire system inflexible and inefficient in utilizing resources, makes it difficult to implement complex functions, and expensive to add new ECUs. The next major step in designing these systems is to go from the current situation with one node one supplier to a situation with one node several suppliers, i.e. there will be several software components of different origins executing on a typical node. Also to enable delivery of more complex applications, it should be possible to spread out software components through several nodes.

3.1.2 Industrial Automation

Typical application domains of industrial automation are in control of industrial processes, power supply, industrial robots. Industrial automation domain comprises a large area of control, monitoring and optimization systems. They typically include large pieces of software that have been developed over many years (often several decades). Most control systems are manufactured in rather large volumes, and must to a large extent be configurable to suit a variety of customer contexts. They can be classified according to different levels of control [4]: (i) *Process level* (for example, a valve in a water pipeline, a boiler, etc.), (ii) *Field level* that concerns sensors, actuators, drivers, etc. (iii) *Group control level* that concerns controller devices and applications which control a group of related process level devices in a closed-loop fashion, (iv) *Process control level* i.s. operator stations and processing systems with their applications for plant-wide remote supervision and control and overview the entire process to be controlled, (v) *Production or manufacturing management level* that includes systems and applications for production planning.

Notice that, even if the higher levels are not embedded, they are of uttermost importance as they need to be interoperable with the lower level which greatly influences the possible choices of the component model and in fine the design choices. The integration requirements have in many cases led to a decision to use component technologies which are not appropriate for embedded systems but provide better integration possibilities.
Depending on the level, the nature of the requirements and the implementation will be quite different. In general, the lower the level, the stronger are the real-time requirements (including timing predictability) and the resource limitations. Also, the component based approach will include different concepts at different levels. While at the lowest levels availability, timeliness, and reliability are the most important quality requirements, at higher levels it will be performance, usability, and integrability.

3.1.3 Consumer Electronics

Consumer electronics products, such as TV, VCR, and DVD, are developed and delivered in form of product families which are characterized by many similarities and few differences and in form of product populations which are sets of products with many similarities but also many differences. Production is organized into product lines - this allows many variations on a central product definition. A product line is a top-down, planned, proactive approach to achieve reuse of software within a family or population of products. It is based on use of a common architecture and core functions included into the product platform and basic components. The diversity of products is achieved by inclusion of different components. Because of the requirements for low hardware and production costs, general-purpose component technologies have not been used, but rather more dedicated and simpler propriety models have been developed. An example of such a component model is the Koala component model used at Philips [13,14]. Koala is a component model and an architectural description language to build a large diversity of products from a repository of components. Koala is designed to build consumer products such as televisions, video recorders, CD and DVD players and recorders, and combinations of them.

3.1.4 Other domains

There are many domains in which embedded systems are used extensively. Some of them are: Telecommunication, avionics and aerospace, transportation, computer games, home electronics, navigation systems, etc. While there is many similarities between these domains there are also very different requirements for their functional and extra-functional properties. The consequences are that the requirements for component-based technologies are different, and consequently we cannot expect to have one component model. The expectations are that many component models will coexist, sharing to less or larger

extent some common characteristics, such as basic principles of component specifications through interfaces, basic composition and run-time services, certain patterns, and similar.

4 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 software engineering, there are many different suggestions for precise definitions of components in component based software development. The best accepted understanding of component in the software industry world is based on Szyperski's definition [11]. 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, in particular 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. This more liberal view is partly motivated by the embedded systems context, as will be discussed in below.

Many important properties of components in embedded systems, such as timing and performance, depend on characteristics of the underlying hardware platform. Kopetz and Suri [3] propose to distinguish between *software components* and *system components*. Extra-functional properties, such as performance, cannot be specified for a software component in isolation. Such properties must either be specified with respect to a given hardware platform, or be parameterized on (characteristics of) the underlying platform. A system component, on the other hand, is defined as a self-contained hardware and software subsystem, and can satisfy both functional and extra-functional properties.

4.1 Component-based approach for small embedded systems

The specific characteristics of embedded systems lead to specific requirements of component technologies. In particular the approaches in development process and component specifications using interfaces are different form those implemented in the component technologies widely used in other domains.

The component interface summarizes the properties of the component that are externally visible to the other parts of the system. As for embedded systems non-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. Therefore the dynamic component deployment is not feasible for small embedded systems.

Taking into account all the constraints for real-time and embedded systems, we can conclude that there are several reasons to perform component deployment and composition at design time rather than run-time [4]: 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: e.g., in a static component composition known at design time, connections between components could be translated into direct function calls instead of using dynamic event notifications. Finally, verification and prediction of system requirements can be done statically from the given component properties.

This implies that the component-based characteristic is mostly visible at design time. To achieve an efficient development process tools should exist which will provide support for component composition, component adaptation and static verification and prediction of system requirements and properties from the given component properties.

There may also be a need for a run-time environment, which supports the component framework by a set of services. The framework enables component intercommunication (those aspects which are not performed at design time), and (where relevant) control of the behavior of the components.

Figure 2 shows different environments in a component life cycle. The figure is adopted from [4].

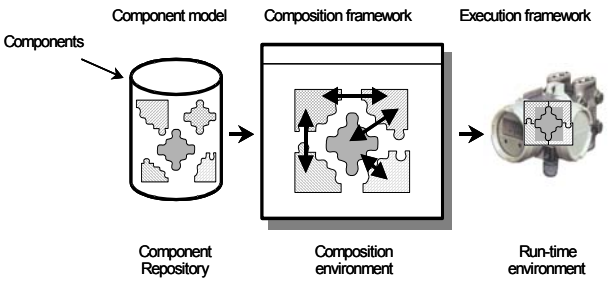

Figure 2. Component technology for small embedded systems

4.2 Component-based approach for large embedded systems

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 of more interesting than in a case for small systems.

The technology mostly used in large systems is Microsoft COM and recently .NET, and to smaller extent different implementations of CORBA, although no one of these technologies provide support for real-time. 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 standards or proprietary solutions. The main reason for wide use of component-based technology is the possibility of reusing solutions in different ranges of products, efficient development tools, standardized specifications and interoperation, and integration between different products.

One successful example of adoption of a component-based technology is the initiative OPC Foundation (OLE process control Foundation, www.opcfoundation.org), an organization that consists of more than 300 member companies worldwide, is responsible for a specification that defines a set of standard interfaces based upon Microsoft's OLE/COM and recently .NET technology. OPC consists of a standard set of interfaces, properties, and methods for use in process-control and manufacturing-automation applications. OPC provides a common interface for communicating with diverse process-control devices, regardless of the controlling software or devices in the process. The application of the OPC standard interface makes possible interoperability between automation/control applications, field systems/devices and business/office applications.

Another example of a component-based approach is development and use of the standard IEC 61131 [5]. IEC 61131 defines a family of languages that includes instruction lists, assembly languages, structured text, a high level language similar to Pascal, ladder diagrams, or function block diagrams (FBD). Function blocks can be viewed as components and interfaces between blocks are released by connecting in-ports and out-ports. Function block execution may be periodic or event-driven. IEC 61131 is successfully used in development of industrial process automation systems for many years.

Large embedded systems that must fulfill real-time requirements usually do not use general-purpose component-based technologies. However in some cases, such as for ABB controllers, a reduced version of COM has been used on a top of a real-time operating system [6]. The reused version includes facilities for component specification using the interface description language (IDL), and some basic services at run-time such as component deployment has been used. These services have been implemented internally. Different communication protocols and I/O drivers have been identified as components (see Figure 3).

5 The needs and priorities in research

The component-based approach on system level, where hardware components are designed with embedded software, has been successfully used for many years. Also large-grain generic components like protocol stacks, RTOSs, etc. have been used for a long time. In addition to this, technology supporting a component-based approach has been developed; either in the form of proprietary component models, or by using reduced versions of some widely used component models.

Still, there are needs for a number of improvements. Some of them are the following (differently important for different domains):

- 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 speed and memory consumption, which is still missing in most of the component technologies available today.
- There is also a need for interoperability between different component technologies. Specification technology is not sufficiently developed to guarantee a priori component interoperability.
- Most current component technologies do not support important extra-functional properties.
- There is a need for generic platform services, for, e.g., security and availability.
- Tools that support component based development are still lacking.
- There is a lack of efficient implementations of component frameworks (i.e., middleware), which have low requirements on memory and processing power.

Major needs for the further development of component technology for embedded systems are the following [16].

- Need for adopted component models and frameworks for embedded systems. A problem is that many application domains have application-dependent requirements on such a technology.
- Need for light-weight implementations of component frameworks. In order to support more advanced features in component-based systems, the run-time platform must provide certain services, which however must use only limited resources.
- Obtaining extra-functional properties of components: Timing and performance properties are usually

obtained from components by measurement, usually by means of simulation. Problems with this approach are that the results depend crucially on the environment (model) used for the measurements may not be valid in other environments, and that the results may depend on factors which cannot easily be controlled. Techniques should be developed for overcoming these problems, thereby obtaining more reliable specifications of component properties.

- Platform and vendor independence: Many current component technologies are rather tightly bound to a particular platform (either run-time platform or design platform). This means that components only make sense in the context of a particular platform.
- Efforts to predict system properties: The analysis of many global properties from component properties is hindered by inherent complexity issues. Efforts should be directed to finding techniques for coping with this complexity.
- Component certification: In order to transfer components across organizations, techniques and procedures should be developed for ensuring the trustworthiness of components.
- Component noninterference: Particularly in safety-critical applications, there is a need to ensure separation and protection between component implementations, in terms of memory protection, resource usage, etc.
- Tool support: The adoption of component technology depends on the development of tool support.

The clearly identified priorities of CBSE for embedded systems are:

- Predicting system properties. A research challenge today is to predict system properties from the component properties. This is interesting for system integration, to achieve predictability, etc.
- Development of widely adopted component models for real-time systems. Such a model should be supported by technology for generating necessary runtime infrastructure (which must be light-weight), generation of monitors to check conformance with contracts, etc.

6 References

1. Avižienis A., Laprie J-C., Randell B., Fundamental Concepts of Computer System Dependability, IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable, Robots in Human Environments, 2001.
2. F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau. Technical Concepts of Component-Based Software Engineering, Volume II. Technical Report CMU/SEI-2000-TR-008, Software Engineering Institute, Carnegie-Mellon University, May 2000.
3. H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In Proc. 6th IEEE International Symposium on Object-oriented Real-Time Distributed Computing (ISORC), Hokkaido, Japan, May 2003.
4. I. Crnkovic and M. Larsson. Building Reliable Component-Based Software Systems. ArtechHouse, 2002.
5. IEC. Application and implementation of IEC 61131-3. Technical report, IEC, Geneva, 1995.
6. F. Lüders, I. Crnkovic, and A. Sjögren. Case study: Componentization of an industrial control system. In Proc. 26th Annual International Computer Software and Applications Conference - COMPSAC 2002, Oxford, UK, Aug. 2002. IEEE Computer Society Press.
7. OMG. Response to the OMG RFP for schedulability, performance, and time (revised submission), June 2001. OMG, RFP ad/2001-06-14.
8. OMG. A uml profile for enterprise distributed object computing, June 18 2001. ptc/2001-12-04.
9. Stankovic J. and Ramamritham K., "Tutorial on Hard Real-Time Systems", IEEE Computer Society Press, 1998.
10. J. Siegel and the O.S.S. Group. Developing in OMG's model-driven architecture, Nov. 2001. OMG, White paper, Revision 2.6.
11. C. Szyperski. Component Software: Beyond Object-Oriented Programming. ACM, Press and Addison-Wesley, New York, N.Y., 1998.
12. C. Szyperski. Component Software: Beyond Object-Oriented Programming. Second edition, ACM, Press and Addison-Wesley, New York, N.Y., 2002.
13. 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.
14. R. van Ommering. Building product populations with software components. In Proceedings of the 24th international conference on Software engineering,. ACM Press, 2002.
15. N. Wang, D. Schmidt, and C. O'Ryan. Overview of the CORBA Component Model, Sept. 2000. White paper.
16. E. Brinksma et al., ROADMAP - Component-based Design and Integration Platforms, W1.A2.N1.Y1, Project IST-2001-34820, ARTIST - Advanced Real-Time Systems