Towards Efficient Development of Embedded Real-Time Systems, the Component Based Approach

Jukka Mäki-Turja , Mikael Nolin, Kaj Hänninen
Mälardalen Real-Time research Centre, Västerås Sweden
E-mail: jukka.maki-turja@mdh.se

Abstract

We present our joint view for efficient development of efficient and dependable ERTS (Embedded Real-Time Systems). This view is based on the three main viewpoints of (1) the software designer, (2) the analysis tools, and (3) the code synthesis tools/run-time environment.

Our position is that any approach that is to achieve (i) decreased development effort, (ii) increased software quality, and (iii) efficient resource utilization, needs to take all three viewpoints into consideration.

We exemplify how our work with execution-model independent software components fits into this joint view and outline some research directions.

1. Introduction

Historically, the developers of embedded real-time systems (ERTS) have used low level programming languages to guarantee full control of the system behavior [25]. A common view on ERTS has been a, once developed, monolithic, platform dependent view, not constructed for evolution. Hence, many ERTS has become legacy systems that are hard to incorporate into functionality and/or technology shifts [9]. In fact, ERTS tends to have very long life-times, decades in some cases. The development effort placed in them can not be ignored.

Today’s embedded systems are typically characterized by having a mix of functionality with requirements ranging from hard real-time, soft or even non real-time. Many of these systems operate in resource constrained environments and need to satisfy requirements on dependability [6][12]. For example, reliability of the systems is paramount [10], software flaws can have disastrous impact and upgrading is often difficult.

In addition, there is a clear trend towards more diverging type of functionality in embedded system [10]. The wide range of functionality in future embedded systems (ranging from predictable safety critical functionality to more resource efficient and flexible soft or non-real-time functionality) requires a shift to a more heterogeneous development methodology with tools and techniques to support efficient development with respect to: (i) development effort, (ii) achieved software quality, and (iii) resource utilization.

In the office-/Internet-area Component-Based Software Engineering (CBSE) has had a tremendous impact. CBSE offers an opportunity to increase productivity by providing natural units of reuse (components and architectures), by raising the level of abstraction for system construction, and by separating services from their configuration to facilitate evolution [5]. Today, there exists several commercial component technologies for the desktop- and Internet-market, e.g., COM/DCOM [13], Corba [19][20], Java Beans/EJB [29][30], .NET [14] are readily available and used by developers on a day-to-day basis. However, these technologies are typically not suitable for embedded control systems [15].

In the embedded systems domain, CBSE is still only perceived as a promising future technology. Several, non-standard, proprietary component technologies for embedded systems have been proposed (e.g. Koala [32], PECOS [16], MetaH [31], VEST [28], the control server [6], ReFlex [33], etc.). EU Projects such a Space4U [27], its predecessor Robocop [22], and DECOS [8] are targeting CBSE for embedded systems. The EU Network of Excellence ARTIST [3] has a track "Component-based Design and Development". The SAVE [24] project, a collaboration between research groups and industrial partners, studied safety of component-based systems.

Ever still, there is an apprehension that current tools and methods for embedded CBSE are lacking one or more key-properties such as:

- Giving the software developer suitable level of expressiveness and/or abstraction
- Enabling development of flexible systems, supporting code and architecture reuse
- Enabling development of resource efficient systems
- Enabling development of predictable systems

2. Joint view of development of ERTS

In order to support efficient component-based development with respect to: (i) development effort, (ii) achieved software quality, and (iii) resource utilization, we believe that development of systems should be done taking three main viewpoints into consideration. These viewpoints need to be jointly approached, by any development methodology, to find a suitable trade-off between conflicting requirements. The viewpoints are:

- The designers/developers viewpoint
- The viewpoint of the analysis framework
- The viewpoint run-time system

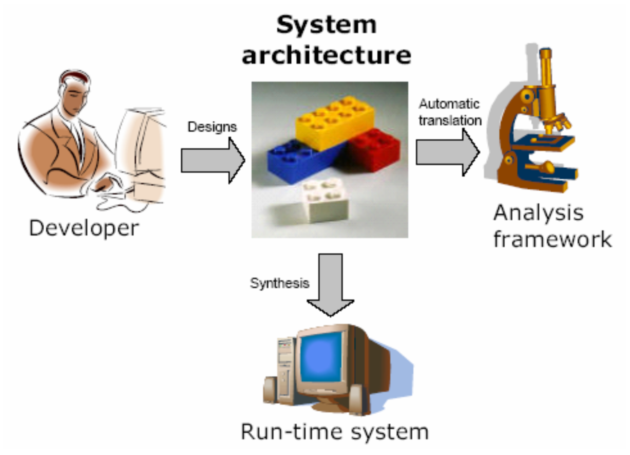

Figure 1: Three main views in ERTS development

These viewpoints typically emphasize different and sometimes conflicting requirements in the development. In reality however, the viewpoints are dependent and should not be considered in isolation (which is traditionally done in the software engineering and real-time communities).

For example, an abstraction mechanism that cannot be analyzed or resource efficiently mapped to a run-time system is of little use for a dependable or resource constrained systems. The other way around, analysis techniques that place a too heavy burden on the developer, such as manually specifying execution time distributions, will not help to bring down development costs or increase software quality since they will not be used.

2.1. Developers viewpoint

The developers view should comprise of sufficient tools to handle the increasing complexity in ERTS, e.g. providing appropriate level of expressiveness and abstraction mechanisms. The developer (or designer) of a system should have a component model, architectural rules and constraints at his disposal to develop a high level (abstract away from pure source code) architecture of the application (system).

The main goal during construction should be to relive the designers from the burden of low level details, so that they can focus on the problem at hand. This facilitates the construction of a component architecture that is understandable, maintainable and formal enough for automated analysis and synthesis.

2.2. Analysis viewpoint

From the analysis viewpoint, the design/architecture must be formal enough so that automated analysis techniques, such as response time and memory utilization analysis, can be performed.

The diverging type of functionality in today's ERTS requires flexible scheduling techniques or several execution models to be analyzed. Hence, a component-based architecture for an application should be analyzed whether it satisfies certain properties or not with an automated analysis framework. The objective is to deal with as intricate problems as possible (to hide complexity from developers) with automated tool support, i.e., the analysis framework must have knowledge of the component architecture (design) as well as the constraints and services provided by the component framework (run-time system).

An important task for an analysis framework is to provide information of assumptions (artifacts) it had to make in order make a property feasible. Such artifacts can typically consist of task model attributes that cannot directly be derived from a high-level design (for example task priorities). This information together with the component architecture that is both syntactically and semantically correct, should be output to a synthesis tool (analogous to a back-end generator of a compiler) when generating code for the run-time environment.

2.3. Synthesis viewpoint

A synthesis should take (as input) the architecture design and possibly some artifacts (such as priorities for a task model) produced by the analysis framework, and map it to the run-time system (generate component glue code etc.). The aim is to provide a run-time system that has a small footprint, but still providing sufficient run-time services to the components of the application. Note that this is a degree of freedom; if the application makes use of lot of run-time services these must be provided by the run-time system. However, at one extreme, if the application is purely static, all connections between components can be resolved off-line which result in a static schedule yielding little run-time overhead.

With this view, the entire component framework is provided at development time (as opposed to run-time for component technologies such as e.g. .NET), but only the part that are used are mapped down to the actual run-time system.

3. Execution-model independent SW

Using this joint view of ERTS development, we will focus on providing methods and tools to enable execution model independent software development.

3.1. State of practice

A typical, execution model *dependent*, development process today is illustrated in Figure 2.

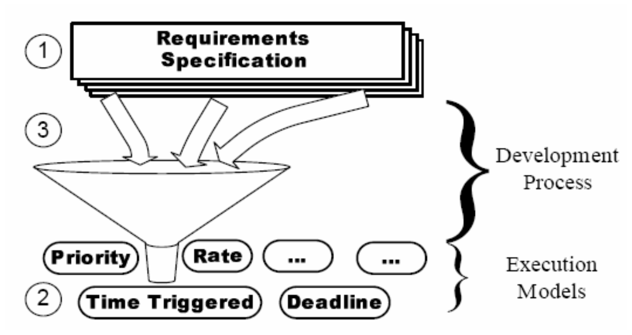

Figure 2: Current practice in development. Requirements are force fitted to a particular execution model.

In the first phase, (1), product requirements are established. Next, some crucial and strategic over-all decisions about the final system are made. One of these decisions include, (2), which execution model (EM) to use (in Figure 2 we choose the time triggered EM as an example). Finally, (3), development takes place, and each requirement is mapped onto the chosen EM.

This force-fitting of requirement to one particular EM has three major drawbacks:

- It increases software complexity, since functionality better implemented in other EMs still needs to be fitted to the chosen EM.
- It reduces software reuse, since the software is tailored to one specific EM.
- It hinders the designer from reaping benefits provided by other EMs [17].

Recent results in real-time scheduling theory [1][2][4][17][18][21][26] make it possible to combine several EMs in one system (and still achieve predictable timing). However, the cited techniques by themselves, do not remove any problems. Instead a change in the software development process, and tools supporting such a new process, is needed.

3.2. Our vision

Figure 3 illustrates an envisioned process where software development to a large extent is done in an EM-independent way (2).

During this phase general and reusable (EM independent) software components are developed or reused. At a later stage, once the bulk of the system has been developed, different parts of the system are mapped to different EMs (3). This mapping is done in such a way that the application specific system requirements are best fulfilled (optimizing e.g. predictability, throughput, reliability, memory footprint, or a mix of these).

Software development with multiple EMs, using a development process as depicted in Figure 3, will have the following main benefits:

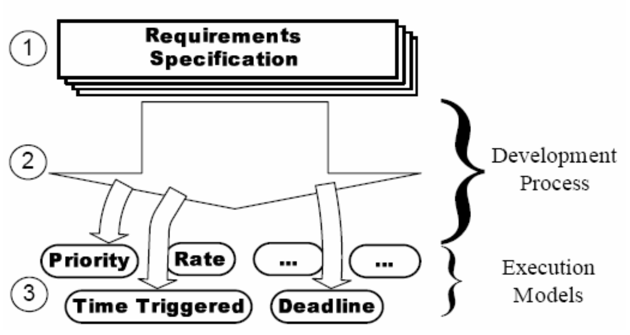

Figure 3: A Vision of a future development process. The choice of execution models is postponed to a later stage.

- Developers can postpone the choice, or rely on automated choice of EM, to later stages of the development process. This enables them to focus on the problem at hand instead of focusing on lower-level details early in the design process.
- Developers can choose different EMs for different subsystems. This will give the developer more expressive power as well as providing the right level of abstraction. With these two points, software complexity will be reduced. This, in turn, will result in lower software development costs and increased software quality.
- The possibility to use static scheduling for critical core functionality (which is often desired and sometimes even mandated by safety standards and certification agencies [11]) while allowing less critical functions to be executed using a less resource demanding and flexible model. This, in turn, will result in a cheaper validation/certification process for critical functions, and in lower production costs (more functions can be fitted onto cheaper hardware).
- Component reuse will be facilitated, since components can be developed independent of the EM. This, in turn, will decrease the software development costs. Also, for companies that sustain a product line, software reuse is crucial and is an important factor in decreasing the time-to-market for new products.

4. Open issues - research questions

A development strategy with execution independent software components, where the designer, the analysis framework, and synthesis tools are seen from a joint viewpoint collaborating to fulfill common requirements, oblige addressing of some concrete research issues as described in the following:

- What is considered an appropriate level of expressiveness and abstraction in development tools for the embedded domains? What type of execution models are actually needed in future embedded systems?
- Should a future component model be general enough to be used in different domains or should it be pin-pointed towards a single domain (such as automotive) or on one extreme, should the model be application specific?
- Since, the role of an analysis framework is to examine a system for feasibility of certain properties, relevant properties need to be identified in future systems. The analysis framework needs behavior-models for the architectural elements. However, manually specifying such models add burdens to the developer and increases both effort and complexity in the development process. Hence techniques for automatic model construction are needed with focus on techniques for model extraction from running systems.
- Techniques for resource efficient run-time monitoring of component-based systems should be derived. Also, based on run-time observations stochastic behavior-models need to be derived. Furthermore, since components can be expected to be reused in various contexts, and each context can result in quite different behavior, the models should allow for context dependent information to be represented.
- This new type of observation-based, stochastic, and context dependent models do not allow commodity analysis techniques to be used. Hence, new, or modified existing, techniques is needed to analyze these models.
- In addition, little is known about the relation between component models and the underlying EMs. Specifically, no component model has been explicitly designed to support multiple EMs, rather existing component models (implicitly or explicitly) assume some form of EM. What changes of existing methodologies are required to support multiple EMs? How does one develop software that is independent of, and can be mapped to, any EM? A methodology for software development using multiple EMs needs to give the engineers means to postpone decisions about what EM to use until late project phases. Also, guidelines and heuristics for how to map EM-independent software to the available EMs need to be derived.
- How can/should software engineering tools support a methodology for multiple EMs? Combining multiple EMs in a single system adds significant complexity to system integration, the integration should, to a large extent, be automatically performed by tools. The tools need to account for system requirements, such as end-to-end deadline, memory usage, communication protocol requirements (e.g. following standards like J1939 [23]).
- These system-integration and configuration tools should be able to synthesize optimized implementations where components are mapped to EMs and hardware resource in an efficient way.

5. Conclusions

Any development environment that claims reduce development effort, increase software quality and minimize resource utilization must consider the requirement from three, sometimes conflicting, viewpoints: (i) The developers'/designers' viewpoint, the viewpoint of the analysis framework, and the run-time framework. One has to be able to find a suitable balance between the requirements placed on the development environment from these viewpoints.

The trend on ERTS is an ever increasing diversity on functionality and complexity of SW. This diversity and complexity must be handled and our approach is a more heterogeneous development environment based on the component based approach with a run-time system that supports multiple execution models. In addition the development should be aided with extensive analysis and synthesis tool support.

In order to achieve our vision in a component based with multiple execution models context, we have identified some research directions:

- What is considered an appropriate level of expressiveness and abstraction?
- What type of execution models are needed in future embedded systems?
- Should a component model be general or specific?
- What analysis properties are of interest for the analysis framework?
- What is the relation between component and the underlying execution models?
- Extensive tool support is needed especially in analysis, synthesis and system integration, how should such tools interact?

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98), pages 4–13, Madrid, Spain, December 1998. IEEE Computer Society.

[2] L. Abeni. Server Mechanisms for Multimedia Applications. Technical Report RETIS TR98-01, Scuola Superiore S. Anna, Pisa, Italy, 1998.

[3] ARTIST (Advanced Real-Time Systems) Network of Excellence. http://www.systemes-critiques.org/-ARTIST/.

[4] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time, and Non-Real-Time Processes. In Proc. 24th IEEE Real-Time Systems Symposium (RTSS). IEEE Computer Society, December 2003.

[5] E. Brinksma, G. Coulson, I. Crnkovic, A. Evans, S. Gérard, S. Graf, H. Hermanns, B. Jonsson, A. Ravn, P. Schnoebelen, F. Terrier, A. Votintseva, and J.M Jézéquel. Component-based design and integration platforms. Roadmap, Advanced Real-Time Systems Information Society Technologies (ARTIST), May 2003.

[6] A. Cervin and J. Eker. The Control Server Model: A Computational Model for Real-Time Control Tasks. In Proc. of the 15th Euromicro Conference on Real-Time Systems, July 2003.

[7] I. Crnkovic and M. Larsson, editors. Building Reliable Component-Based Software Systems. Artech House publisher, 2002. ISBN 1-58053-327-2.

[8] DECOS - Dependable Embedded Components and Systems. https://www.decos.at.

[9] J. Fröberg, K. Sandström, C. Norström, H. Hansson, J. Axelsson, and B. Villing. Correlating business needs and network architectures in automotive applications - a comparative case study. In proceedings of the 5th IFAC International Conference on Fieldbus Systems and their Applications (FET), Aveiro, Portugal, July 2003.

[10] K. Hänninen, J. Mäki-Turja, M. Nolin, Present and Future Requirements in Developing Industrial Embedded Real-Time Systems - Interviews with Designers in the Vehicle Domain, To be published in the Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Potsdam, Germany, March 2006

[11] IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems.

[12] A. Möller, J. Fröberg, and M. Nolin. Industrial Requirements on Component Technologies for Embedded Systems. In 7th International Symposium on Component-based Software Engineering (CBSE7).

[13] Microsoft. Microsoft COM Technologies. http://www.microsoft.com/com/.

[14] Microsoft. .NET Home Page. http://www.microsoft.com/net/.

[15] A. Möller, M. Åkerholm, J. Fredriksson, and M. Nolin. Evaluation of Component Technologies with Respect to Industrial Requirements. In Euromicro Conference, Component-Based Software Engineering Track, August 2004.

[16] P. O. Müller, C. M. Stich, and C. Zeidler. Building Reliable Component-Based Software Systems, chapter Component Based Embedded Systems, pages 303–323. Artech House publisher, 2002. ISBN 1-58053-327-2.

[17] J. Mäki-Turja, K. Hänninen, and M. Nolin. Efficient Development of Real-Time Systems Using Hybrid Scheduling. In International conference on Embedded Systems and Applications (ESA), June 2005.

[18] J. Mäki-Turja and M. Nolin. Fast and Tight Response-Times for Tasks with Offsets. In 17th Euromicro Conference on Real-Time Systems. IEEE, July 2005.

[19] OMG. CORBA Home Page. http://www.omg.org/corba/.

[20] OMG. CORBA Component Model 3.0, June 2002. http://www.omg.org/technology/documents/formal/-components.htm.

[21] J.C. Palencia Gutierrez and M. González Harbour. Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities. In Proc. 24th IEEE Real-Time Systems Symposium (RTSS), December 2003.

[22] Robocop project home-page. http://www.extra.research.philips.com/euprojects/robocop/index.htm.

[23] SAE Standard. SAE J1939, Joint SAE/TMC Electronic Data Interchange Between Microcomputer Systems In Heavy-Duty Vehicle Applications. http://www.sae.org.

[24] SAVE. SAVE Project Page. http://www.artes.uu.se/++/SAVE/

[25] K. Sandström, J. Fredriksson, and M. Åkerholm. Introducing a component technology for safety critical embedded real-time systems. In International Symposium on Component-based Software Engineering (CBSE7), Edinburgh, Scotland, May 2004.

[26] M. Sjödin. Response-Time Analysis for Dynamically and Statically Scheduled Systems. Technical Report MRTC Report no. 55, Mälardalen Real-Time Research Centre (MRTC), April 2002. http://www.mrtc.mdh.se/showPublications.phtml.

[27] Space4u project home-page. http://www.extra.research.philips.com/euprojects/space4u.

[28] J. A. Stankovic. VEST — A toolset for constructing and analyzing component based embedded systems. Lecture Notes in Computer Science, 2211:390–??, 2001.

[29] SUN Microsystems. Enterprise Javabeans Technology. http://java.sun.com/products/ejb/.

[30] SUN Microsystems. Introducing Java Beans. http://developer.java.sun.com/developer/onlineTraining/-Beans/Beans1/index.html.

[31] S. Vestal. Support for Real-TimeMulti-Processor Avionics. In Proc. 18th IEEE Real-Time Systems Symposium (RTSS), pages 11–21, December 1997.

[32] R. van Ommering, F. van der Linden, J. Kramer, J. Magee. The Koala Component Model for Consumer Electronics Software. IEEE Computer, 33(3):78–85, March 2000.

[33] A. Wall. Architectural Modelling and Analysis of Complex Real-Time Systems. PhD thesis, Mälardalen University, Dept. of Computer Science and Engineering, September 2003.