STRUCTURING CRITERIA FOR THE DESIGN OF COMPONENT-BASED REAL-TIME SYSTEMS

Aleksandra Tešanović and Jörgen Hansson
Department of Computer Science, Linköping University, Sweden
{alete,jorha}@ida.liu.se

ABSTRACT

Using component-based software development in real-time system development offers significant benefits especially in enabling configurability, and rapid development and deployment of real-time software. Successful integration of component-based software development into real-time system development is greatly dependent on identifying issues that a design method for component-based real-time systems needs to address. The issues focused on by existing design approaches are decomposition of a real-time system into tasks and temporal analysis, while issues such as components, their relationship to tasks, and separation of concerns are not addressed. In this paper, we identify the criteria a design method for component-based real-time systems should fulfill to enable efficient, reuse-oriented, development of reliable and configurable real-time systems. The criteria include a real-time component model that supports mapping of components to tasks, separation of concerns in real-time systems through the notion of different types of aspects, and composition support, namely support for configuration and analysis of the composed real-time software. By introducing a new set of criteria for the design of component-based real-time systems, we help bridging the gap between the real-time and software engineering community and offer a good basis for evaluating existing, and developing new, design methods for building component-based real-time systems.

KEYWORDS

Design method, real-time systems, component model, aspects, reusability, configurability, interfaces, temporal analysis.

1. INTRODUCTION

Real-time and embedded systems are being used widely in modern society of today. However, successful deployment of embedded and real-time systems depends on low development costs, high degree of tailorability and quickness to market [18]. Thus, the introduction of the component-based software development (CBSD) [21] paradigm into real-time and embedded systems development offers significant benefits, namely: (i) configuration of real-time software for a specific application using components from the component library, thus, reducing the system complexity as components can be chosen to provide the functionality needed by the system; (ii) rapid development and deployment of real-time software as many software components, if properly designed and verified, can be reused in different embedded and real-time applications; and (iii) evolutionary design as components can be replaced or added to the system, which is appropriate for complex real-time systems that require continuous hardware and software upgrades.

To successfully assemble real-time systems out of components and fully exploit potentials of CBSD in real-time system development a number of research challenges need to be addressed. In this paper we focus on the following: (i) issues that should be addressed and the criteria that should be enforced by a design method to allow integration of CBSD into real-time systems; and (ii) the relationship between components and tasks in real-time systems, with a focus on questioning the traditional view of real-time systems as systems composed out of tasks only.

The problem of decomposing real-time systems into tasks and modules (manageable units) has been researched and addressed extensively over the years [8, 9, 10, 14, 13, 18, 20]. In most traditional approaches to real-time system design, the system was first decomposed into tasks, and then tasks where grouped into modules, i.e., traditional functions [8, 9, 14]. In this paper we show that component-based design of real-time systems requires decomposition of a real-time system first into software components, which are later mapped to tasks. We also show that, in contrast to design methods used in the real-time system design so far [8, 9, 14], in a component-based real-time system the relationship between tasks and components should not be fixed. We show that a real-time software component should support two views, namely the structural (static) view, which is the traditional view of a software component in software engineering, and the temporal view where components are mapped to tasks.

Existing design methods for real-time systems mainly focus on fulfilling temporal constraints and decomposing real-time systems into tasks [8, 9, 10, 14, 18, 20, 11]. Configurability issues such as a component model, interfaces, and separation of concerns are only addressed by software engineering methods for building general-purpose component-based systems [1, 23, 12]. Hence, there is a gap between the approaches from real-time and software engineering community, as real-time community has focused primarily on the real-time issues not exploiting modularity of software to the extent that software engineering community has done. By introducing a new set of criteria that a design method for component-based real-time systems should fulfil, we help bridging the gap between the communities and offer a good basis for evaluating existing, and developing new, design methods for building component-based real-time systems.

Our contribution is two-fold: (i) providing a set of criteria for evaluation of existing and new approaches to design of real-time systems emphasizing configurability, and (ii) determining the relationship between components and tasks in component-based real-time systems.

The paper is organized as follows. In section 2 we study design approaches in real-time and software engineering, and in section 3 we introduce a uniform set of criteria that a design method for component-based real-time systems should fulfil. The paper finishes with main conclusions and directions for our future research in section 4.

2. SOFTWARE ENGINEERING AND REAL-TIME DESIGN METHODS

Since the component-based paradigm is a product of the advancements in the software engineering research community, to efficiently apply these ideas to real-time system development it is essential to first identify the issues considered to be of prime importance when developing a general-purpose component-based software system (section 2.1), and contrast those with the most relevant issues addressed by design approaches in the real-time community (section 2.2). These help us form a common ground for the criteria that any design method for component-based real-time systems should fulfill.

2.1 Software Engineering Design Methods

The design methods for general-purpose software systems in the software engineering community are mostly targeted towards a *component model*. The component model typically enables full enforcement of the *information-hiding* criterion [16] as components are considered to be black boxes that communicate with each other, or the environment, through well-defined *interfaces* [21]. This is a component view taken by the first generation of component-based systems, e.g., COM and CORBA. Both COM and CORBA provide a standardized component model with interface definition languages, but lack support in composability of different system configurations [6, 15].

Some design approaches [23, 1, 17, 3] have taken one step further in software configurability by providing support for *aspects* and *aspect weaving*, thus adopting the main ideas of aspect-oriented software development [12]. A typical representative of programming languages that explicitly provide ways for specifying aspects is AspectJ [23]. It is accompanied by powerful configuration tools for development of software systems using aspects written in AspectJ and components written in Java. However, AspectJ is limited as it supports only the notion of white box components, i.e., components are not encapsulated and their behavior is visible to other parts of the system, thus not exploiting the powers of information hiding. Invasive software composition (ISC) [1] overcomes the drawbacks of pure aspect language approaches, and enforces information hiding by having a well-defined component model, called the box. The box is a general component model supporting two types of interfaces: explicit interfaces and implicit interfaces. Explicit interfaces are used for inter-component communication, while implicit interfaces are used for aspect weaving into the code of components. Here, components can be viewed as gray boxes in the sense that they are encapsulated, but still aspect weaving can change their behavior.

2.2 Real-Time Design Methods

There are several established design methods developed by the real-time community. We focus on a few representative approaches [8, 9, 10, 14, 13, 7, 2] to illustrate the types of requirements that a real-time domain places on a design method.

DARTS [8], a design approach for real-time systems, ADARTS [9] (DARTS extension for Ada-based systems), and COMET [10] (DARTS extension to UML) focus on the decomposition of real-time systems into tasks. In DARTS and its variants, a real-time system is first decomposed into tasks that are then grouped into software modules. DARTS, therefore, emphasizes the need for having *task structuring criteria* that could help the designer to make the transition from tasks to modules. Modules in DARTS typically represent traditional functions. Hence, the method offers two views on a real-time system: (i) a temporal view where the system is composed out of tasks, and (ii) a structural view where the system is composed out of modules performing a specific (usually task-oriented) function. Configuration of DARTS-based systems is done using *configuration guidelines* that are very clear and have been refined over the years, especially through their use in industry. With COMET, DARTS approach to design of real-time systems has been extended to use the UML tool environment and object-oriented technology, thereby improving information hiding as objects provide better information hiding than traditional modules. However, it has been recognized that DARTS does not provide mechanisms for checking and verifying temporal behavior of a system under development [14, 2]. Similarly, COMET, the most recent variant of DARTS, does not provide support for temporal analysis of the system under construction nor does it support configurability of the real-time software.

TRSD, a transactional real-time system design, is an approach to real-time system development introduced by Kopetz et al. [14]. Building blocks of a real-time system are transactions consisting of one or several tasks. A transaction, in this context a group of tasks, is associated with real-time attributes, e.g., deadline and criticality. TRSD, in addition to rules for decomposition of real-time systems into tasks, provides *temporal analysis* of the overall real-time system. TRSD goes one step further than DARTS and its variants by providing support for temporal analysis of a real-time system under development. The extension of TRSD discussed in [13] establishes the notion of a component in a distributed real-time system as a distributed node (with software and the underlying hardware). Components are equipped with temporal interfaces enabling temporal analysis of the composed distributed real-time system. However, TRSD and its recent extensions focus on system-level components that are large-grained as compared to the traditional software components as they include both software and the hardware.

HRT-HOOD [2], a hard real-time hierarchical object oriented design, is an extension of the well-defined HOOD design method to the real-time domain. As such, it utilizes the HOOD tools to support the real-time design process. Building blocks of a real-time system in HRT-HOOD are HRT-HOOD objects, supplied with two different types of interfaces, namely required interface and provided interface. Having been based on the object-oriented technology and supporting *different types of interfaces*, HRT-HOOD enforces information hiding in terms of objects as entities that hide the information. HRT-HOOD makes a distinction between the logical and physical architectural design. The logical design results in a collection of terminal objects (these do not require further decomposition) with a fully defined interaction. This design step assumes that the designer knows the relationship of an object to a task, since an object can represent one or more tasks. At the physical design stage, the logical architecture is mapped to physical resources on the target system. The physical design stage is primarily concerned with object allocation, network scheduling, processor scheduling, and dependability. Additionally, HRT-HOOD provides support for static priority analysis of the overall real-time system. Although the HRT-HOOD design process is well defined and supported by tools, it does not facilitate object reuse.

VEST [18, 19], a Virginia embedded systems toolkit, is a *configuration tool* for development of component-based real-time systems. VEST provides a graphical environment in which temporal behavior of the building blocks of a real-time system can be specified and analyzed, e.g., worst-case execution times (WCETs), deadlines, and periods. VEST supports two views of real-time components, temporal and structural, and assumes that components making the system configuration are later mapped to tasks. However, the actual process of mapping between a component and a task is not defined. VEST recognizes the need for having *separation of concerns* in a real-time system design. Hence, VEST provides support for analysis of the component memory consumption, which is a concern that crosscuts the structure of the overall component-based real-time system. In its recent edition [19], the tool has been extended to support design-level crosscutting concerns by providing a description language for design-level aspects of a real-time system. In the first version of VEST components were fine-granule, but VEST did not have an explicit component model, implying that components could be pieces of code, classes, and objects [18]. Currently VEST uses the well-defined CORBA component model [19].

RT-UML [7], a real-time unified modeling language, provides stereotypes for specifying real-time notions. Although RT-UML provides powerful notation for modeling real-time systems it essentially provides only syntax, not semantics, for the real-time system design. Thus, RT-UML cannot be considered a design method, rather it is an infrastructure for a design method as it provides a visual language as a basis for enforcing design methods, e.g., its powerful expressiveness could be used by a design method as means of specifying real-time software components [10, 4].[1] Therefore, in the remainder of the paper, we do not discuss properties of RT-UML with respect to component-based real-time system design criteria.

3. COMPONENT-BASED REAL-TIME SYSTEMS DESIGN CRITERIA

We have shown (in section 2) that there is a gap between the approaches from different communities as the real-time community has focused primarily on the real-time issues not exploiting modularity and composability of software to the extent that the software engineering community has done. In this section we present the first uniform set of criteria for the design of component-based real-time systems and discuss the extent to which design approaches investigated fulfil the criteria. The criteria include component model (section 3.1), aspect separation (section 3.2), and system composability (section 3.3).

Table 1. The criteria for the evaluation of design approaches

Design approaches / Criteria	COMET	TRSD	VEST	ISC	COM	AspectJ	HRT-HOOD
Component model							
CM1 Information hiding	◐	●	◐	●	●	◐	●
CM2 Interfaces	◐	●	◐	●	●	●	●
CM3 Component Views	◐	◐	●	◐	-	-	◐
CM4 Temporal attributes	-	◐	●	-	-	-	●
CM5 Task mapping	●	-	◐	-	-	-	-
Aspect separation							
AS1 Aspect support	-	-	◐	●	-	●	-
AS2 Aspect weaving	-	-	-	●	-	●	-
AS3 Multiple interfaces	-	◐	-	●	●	◐	◐
AS4 Multiple aspect types	-	-	-	-	-	-	-
System composability							
SC1 Configuration support	◐	◐	●	●	◐	●	◐
SC2 Temporal analysis	-	-	◐	-	-	-	●

LEGEND: ● supported ◐ partially supported - not supported

COMET: concurrent object modeling and architectural design with UML
TRSD: transactional real-time system design
VEST: Virginia embedded systems toolkit
ISC: invasive software composition
HRT-HOOD: a hard real-time hierarchical object-oriented desing

3.1 Component Model

One of the most fundamental reasons behind the need to divide software into modules is to guarantee exchange of parts. In mature industries, e.g., mechanical engineering, an engineer constructs a product by decomposition; the problem is decomposed into sub-problems until one arrives to the basic problem for which basic solution elements can be chosen. Software engineering is not different from other engineering

[1] RT-UML can be used, for example, as a tool for designing systems based on the COMET design method.

disciplines in the effort to mature, i.e., enable software decomposition into components and use of already developed components to build software systems.

To be able to build software systems out of reusable components, we need a way of specifying what a component should look like, i.e., we need a component model. A component model describes what a software component should look like, and it supports modularity of the software in the sense that it defines what should be hidden in a component such that the component can be exchanged and reused in several systems [1, 21, 5]. Further, a component model should enforce information hiding, implying that the deployers of a component cannot see when the manufacturer changes the component internals, and that newer versions of components can substitute older versions if the interfaces of components remain the same. Hence, the interfaces of components should be well defined by the component model to provide necessary information to a component user, e.g., reuse context and performance attributes. Most software engineering approaches enforce a good component model by providing information hiding and having well defined interfaces for accessing components, e.g., COM, CORBA, and ISC (see table 1).

The most important criteria for a software component with respect to component model (CM) can be summarized as follows [1].

CM1 *Information hiding*. A component has to maintain one or several secrets, e.g., design, implementation, and programming language, which are encapsulated to enable component exchangeability.

CM2 *Interfaces*. A component should have one or more well-defined interfaces to the environment. The component can be accessed only through these interfaces (thus complementing the information hiding criterion).

Real-time systems sharing the component vision are faced with additional requirements on the component model, as components in a real-time system should provide mechanisms for handling temporal constraints. Moreover, since the traditional view of real-time systems implies tasks as building elements of the system, the relationship between tasks and components needs to be clearly identified. We argue that this relationship (between a real-time software component and a task) should not be fixed for several reasons. First, we would like to reuse any software components applicable to the real-time system under construction, i.e., we do not want to limit reusability of the real-time software only to tasks. Second, perfect mapping of a component to a task for all applications is hard to determine. We illustrate this with a simple example as follows. Assume that we have a set of components, e.g., $c_1 \ldots c_6$, in a component library (see figure 1). If the components are developed to be reusable that implies that the same set or a subset of components $c_1 \ldots c_6$ can be used in different run-time environments (denoted RT_1-RT_3 in figure 1). Since different run-time environments typically have different resources (tasks) available, we can observe that the operations o_{ij}, offered by the component c_i to the environment via interfaces, could be allocated differently to tasks in different real-time environments, depending on the actual available resources of the underlying run-time environment.

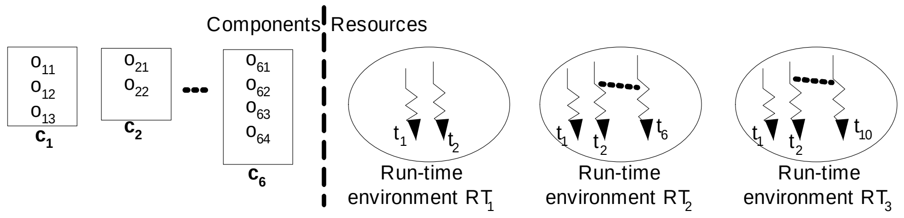

Figure 1. An example of the component-to-task relationship

The problem now is to distinguish between components and tasks in a component-based real-time system, and to determine their relationship. A similar problem is addressed in COMET (and DARTS) as COMET method is concerned with the relationship between tasks, modules, objects, and classes, and the way tasks can be efficiently structured into these. To solve the problem (objects vs. tasks) COMET proposes a set of task structuring criteria to guide the designer when grouping tasks into classes and objects. We already argued for the necessity of having components and tasks distinguished from each other and their relationship clearly defined. One of the ways to achieve this is to distinguish between a temporal and a structural dimension of a component. The structural dimension (or structural view) of a component represents a traditional software component as perceived by software engineering community, i.e., software artifact with well-defined interfaces that enforces information hiding. The temporal dimension should reflect real-time attributes needed to ensure mapping of components to tasks on the target platform and performing temporal analysis of the resulting real-time system. This is especially true for components built for use in hard real-time systems. In all approaches discussed, VEST is the only platform that can be considered to enforce the component views as it allows components (pieces of code) to be mapped to tasks for a particular platform (see table 1). However, VEST does not provide detailed information on how the actual mapping should be done. The graphical environment of VEST enables storing temporal attributes of components that are used for schedulability analysis. More traditional approaches to real-time system design, COMET and TRSD, do not provide temporal information for their components (see table 1). HRT-HOOD, in contrast, supports only a temporal dimension of objects as building parts of a real-time system. Hence, we conclude that in a real-time environment, a software component model should support the following.

CM3 *Component views*. A design method should support decomposition of real-time systems into components as basic building blocks, and, further, components should support both a structural and a temporal view. In the structural view real-time software is composed out of software components, and in the temporal view the real-time software is composed out of tasks.

CM4 *Temporal attributes*. A component model should provide means for specifying component temporal attributes, e.g., WCET, in order to support temporal and structural views of a real-time system, and enable static and dynamic temporal analysis of the overall real-time system.

CM5 *Task mapping*. A design method should provide clear guidelines (or possibly tools) to support mapping of components to tasks.

3.2 Aspect Separation

While modularity helps to functionally decompose a system, designers would like to have modular exchange in several dimensions so that different features of components can be exchanged separately [1]. Separation of concerns is the main idea of AOSD [12]. Aspects of software such as persistence and debugging can be described separately and exchanged independently of each other without disturbing the modular structure of the system. AOSD offers several important advantages [1]: (i) aspect specifications can be exchanged separately from the component[2] and independently from each other; (ii) aspect weaving creates system configurations that might have completely different functionality or non-functional features; (iii) separation of concerns into aspects eases configuration of the system as fewer parts of the overall system need to be modified for a particular system configuration; and (iv) a system becomes scalable as the same core components can be reused elsewhere. These are the reasons why some design approaches in software engineering, e.g., ISC and AspectJ, provide support for aspects and aspect weaving (see table 1). A restricted form of aspect separation is also realized in COM and CORBA through defining multiple interfaces as access points for the component, since each of the interfaces can be viewed as one aspect.

It is clear that software engineering design methods increasingly emphasize support for aspects and aspect weaving, giving us the following criteria with respect to aspect separation (AS) [12, 1].

AS1 *Aspect support*. A design method should support separation of concerns, namely it should provide support for identifying and specifying aspects in a real-time software system.

AS2 *Aspect weaving.* A design method should provide tools that weave aspect specifications into the final product.

AS3 *Multiple interfaces*. A component should have multiple interfaces under which it can be accessed.

The frontier where aspects and aspect weaving meet real-time has not been explored yet, although there is a strong motivation for using aspects in real-time system development. Namely, some of the core real-time concerns such as synchronization, memory optimization, power consumption, temporal attributes, etc., are crosscutting, in typical implementations of real-time systems, the overall system. Moreover, these concerns cannot be easily encapsulated in a component with well-defined interfaces. However, some real-time design approaches do not support aspects and aspect weaving at all, such as DARTS. HRT-HOOD and TRSD with its new extensions [13] support different types of interfaces, e.g., provided and required, and, hence, support only the AS3 criterion (see table 1). Some work on aspect separation in the design of real-time systems has been addressed in VEST. However, VEST supports aspects like memory consumption describing the run-time behavior of the component with respect to memory consumption but not changing the code of the VEST

[2] As already mentioned, a component in AOSD is typically a white box component, e.g., traditional program, function and method.

components. From this, it can be observed that real-time systems have another dimension of crosscutting compared to general-purpose software systems, e.g., the VEST memory consumption aspect describes the memory needs of components and crosscuts the entire real-time system composed out of components. Hence, in a real-time environment, not only is it desirable to support aspects that crosscut the code of the components, but also aspects that crosscut the structure of a real-time system and describe the behavior of components and the system. This implies that a design method of configurable real-time systems should support multiple aspect types. We express this in the following criterion.

AS4 *Multiple aspect types.* The notion of separation of concerns in real-time systems is influenced by the nature of a real-time system, e.g., temporal constraints and run-time environment. Thus, a real-time system design method should support aspects that crosscut code of core components, as well as additional aspect types to specify the behavior of a real-time component in the run-time environment.

3.3 System Composability

Software for real-time systems should be produced quickly, reliably, and should be optimized for a particular application or a product. Successful deployment of software components for building real-time systems depends on the availability of tools that would improve development, implementation, and evaluation of component-based real-time systems. An example of an effort to create a tool helping the designer to configure and analyze a component-based real-time system is VEST. Configuration support, although not automated by tools, has also been addressed by both COMET and TRSD as they provide detailed design guidelines for real-time system composition. Note that most approaches to design of general-purpose software systems, with the exception of COM and CORBA, have powerful tools for software configuration. However, analysis of the software behavior is of secondary (if any) interest, since these approaches do not provide means for analyzing behavior of the composed software system (see table 1).

Temporal analysis of a real-time system is one of the key issues in real-time system development, and there are a number of approaches that provide support in performing temporal analysis of the real-time system in their design method, namely TRSD, HRT-HOOD, and VEST (see table 1). We conclude that a design method for building reliable component-based real-time systems should provide support for exchanging components and configuring new systems out of existing, or newly developed, components. Hence, the following should be considered with respect to system composability (SC).

SC1 *Configuration support.* The design method should provide recipes for combining different components into system configurations.

SC2 *Temporal analysis.* The design method should provide support for temporal analysis of the composed real-time system. Tools to achieve predictable software behavior are preferable.

4. SUMMARY

We have identified necessary criteria that a design method for building configurable real-time systems needs to satisfy, namely: (i) a real-time component model with two views, temporal and structural, is required to facilitate easy system composition; (ii) aspects and aspect weaving as means of achieving efficient component and system tailoring; and (iii) tools for configuration support and temporal analysis of the real-time system composed out of components. These criteria represent a combination of requirements placed on a design method by the software engineering community, to achieve software reusability and configurability, and requirements placed on a design method by the real-time system community, to achieve system predictability. Hence, for the component-based development to be efficiently applied to the real-time domain, a design method targeted towards building component-based real-time systems should satisfy all the criteria identified. However, we showed that existing real-time design approaches fulfil only a small subset of the criteria as they primarily focus on task decomposition and temporal view of real-time systems (COMET, DARTS, HRT-HOOD), and temporal analysis (TRSD, HRT-HOOD). Some criteria, such as support for separation of concerns via aspects and a well-defined component model with different types of interfaces are only partially addressed, e.g., VEST provides only design-level aspects while HRT-HOOD and COMET use object-orientation to achieve information hiding. Clearly, there is a need for extending existing design approaches to support the identified issues, or specifically developing new design approaches that fulfil identified criteria. The criteria could further be ranked based on their importance given that some real-time applications could consider decomposition into components to be more important than support for aspects, while other could emphasize efficient handling of crosscutting concerns. We are currently developing a design method for configurable real-time systems that is a basis for aspectual component-based real-time systems development [22], and that aims at fulfilling the criteria identified in this paper.

REFERENCES

[1] U. Aßmann. *Invasive Software Composition*. Springer-Verlag, December 2002.

[2] A. Burns and A.Wellings. *HRT-HOOD: a Structured Design Method for Hard Real-Time Ada Systems*, volume 3 of *Real-Time Safety Critical Systems*. Elsevier, 1995.

[3] Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn. Using AspectC to improve the modularity of path-specific customization in operating system code. In *Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th International Symposium on the Foundations of Software Engineering (FSE-9)*, 2002.

[4] I. Crnkovic, B. Hnich, T. Jonsson, and Z. Kiziltan. Specification, implementation, and deployment of components. *Communications of the ACM*, 45(10):35–40, October 2002.

[5] I. Crnkovic and M. Larsson, editors. *Building Reliable Component-Based Real-Time Systems*. Artech House Publishers, July 2002.

[6] A. Dogac, C. Dengi, and M. T. Öszu. Distributed object computing platform. *Communications of the ACM*, 41(9):95–103, 1998.

[7] B. P. Douglass. *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Addison-Wesley, 2000.

[8] H. Gomaa. A software design method for real-time systems. *Communications of the ACM*, 27(9):938–949, September 1984.

[9] H. Gomaa. A software design method for Ada based real time systems. In *Proceedings of the 6th Washington Ada symposium on Ada*, pages 273–284, McLean, Virginia, United States, 1989. ACM Press.

[10] H. Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[11] D. Isovic and C. Norström. Components in real-time systems. In *Proceedings of the Eight International Conference on Real-Time Computing Systems and Applications (RTCSA'02)*, pages 135–139, Tokyo, Japan, March 2002.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the ECOOP*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.

[13] H. Kopetz and H. Obermaisser. Temporal composability. *Computing and Control Engineering Journal*, pages 156–162, August 2002.

[14] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schütz. The design of real-time systems: from specification to implementation and verification. *Software Engineering Journal*, 6(3):72–82, 1991.

[15] M. T. Özsu and B. Yao. *Component Database Systems*, chapter Building Component Database Systems Using CORBA. Data Management Systems. Morgan Kaufmann Publishers, 2000.

[16] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[17] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: an aspect-oriented extension to C++. In *Proceedings of the TOOLS Pacific 2002*, Sydney, Australia, February 2002. Australian Computer Society.

[18] J. Stankovic. VEST: a toolset for constructing and analyzing component based operating systems for embedded and real-time systems. In *Proceedings of the Embedded Software, First International Workshop (EMSOFT 2001)*, volume 2211 of *Lecture Notes in Computer Science*, pages 390–402, Tahoe City, CA, USA, October 2001. Springer-Verlag.

[19] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. VEST: an aspect-based composition tool for real-time systems. In *Proceedings of the 9th Real-Time Applications Symposium 2003*, Toronto, Canada, May 2003. IEEE Computer Society Press

[20] D. S. Stewart. Designing software components for real-time applications. In *Proceedings of Embedded System Conference*, San Jose, CA, September 2000. Class 408, 428.

[21] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 1999.

[22] A. Tesanovic, D. Nyström, J. Hansson, and C. Nörstrom. Towards Aspectual Component-Based Real-Time Systems Development. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'03)*, Taiwan, February 2003.

[23] Xerox Corporation. *The AspectJ Programming Guide*, September 2002.