Research Planning Course
Assignment 2

Design-time prediction of extra-functional properties of component-based embedded software systems

Juraj Feljan
juraj.feljan@mdh.se

2009-10-01
# Table of Contents

1 Introduction ......................................................................................................................................1  
1.1 Research area ............................................................................................................................1  
1.1.1 Component-based software engineering ..............................................................................1  
1.1.2 Model-based software engineering ......................................................................................1  
1.1.3 Embedded systems ................................................................................................................2  
1.1.4 CBSE vs. MBSE .....................................................................................................................2  
1.2 My research topic ......................................................................................................................2  
1.3 Research methodology .............................................................................................................2  
2 Research area overview ................................................................................................................3  
2.1 Related work .............................................................................................................................3  
2.1.1 Component models for embedded systems with support for EFPs .................................3  
2.1.2 Model-based approaches for embedded systems with support for EFPs ......................7  
2.2 Research groups ......................................................................................................................7  
2.3 Central literature .......................................................................................................................8  
2.4 Key conferences ......................................................................................................................8
1 Introduction

1.1 Research area

Today software has become inevitably intertwined with our daily routine, as it is used in industry, business, communication, traffic, health, research, education, entertainment, etc. Software systems are continuously growing in complexity and size, increasing the cost of developing and maintaining them. Also, there is an increasing demand on software to be robust, reliable, flexible, adoptable, etc. Thus, a systematic approach to steering the whole software system life cycle is necessary. Component-based software engineering and model-based software engineering are two, relatively new approaches for handling the complexity of software systems.

1.1.1 Component-based software engineering

Component-based software engineering (CBSE) promotes building software systems from pre-existing software components. The central idea behind CBSE is to separate the system development from the development of components, and, ideally, build systems by finding appropriate components and simply plugging them together. A component should be reused across different systems. Thus, employing CBSE should:

- shorten time to market;
- facilitate management of complex systems – by dividing them into smaller, less complex components;
- increase quality of the software – since a component is intended for use in multiple systems, it can be repeatedly tested in various contexts;
- simplify maintenance – improving existing or adding new functionality is done by replacing existing or adding new components, respectively.

A paramount concept in CBSE is the component model. A component model provides methods and rules for (i) component specification and (ii) component composition. A component is distinguished from other forms of packaged software by conformance to a component model.

1.1.2 Model-based software engineering

Model-based software engineering (MBSE, or MDSE (Model-driven software engineering)) promotes tackling with software complexity by raising the abstraction level of software systems\(^1\) to a higher level than code, closer to concepts from the application domain than to algorithmic concepts. This enables the developer to focus on the application logic, without worrying about implementation details. Ideally, models would be used for code generation, but also for generation of non-implementation artifacts such as documentation, tests, deployment scripts or other models.

MBSE encourages that a model of a system on the highest level of abstraction describes how the system is constructed, without saying anything about the technologies used for implementation. Such a model is called platform independent model (PIM). A PIM should be incrementally transformed (refined) into a platform specific model (PSM). Finally, a PSM should be transformed into code. This separation of platform independent and platform specific information facilitates generating different system implementations from a single specification.

---

\(^1\) Similarly, the abstraction level was raised when switching from assembler to procedural languages, and from procedural to object-oriented languages.
1.1.3 Embedded systems

Today most computer systems are embedded systems. The term embedded system is not exactly defined, but usually refers to a microprocessor-based system with a single dedicated function (or few dedicated functions, as opposed to a general purpose personal computer). An embedded system can be part of a larger system or product, for example the anti lock braking system in a car. Embedded systems span from small devices such as MP3 players to large systems such as factory controllers. The complexity of embedded systems follows accordingly, from a single microcontroller chip, to multiple nodes connected via a network.

Usually embedded systems have limited resources and often a need for dependable operation in safety-relevant scenarios. Thus, for embedded systems extra-functional properties (EFPs) have equal importance as the functional ones. EFPs encompass timing properties (e.g. worst-case execution time), resource-wise properties (e.g. memory consumption), dependability, portability etc.

1.1.4 CBSE vs. MBSE

CBSE and MBSE are two approaches, naturally having differences. In general, looking at the whole development cycle, they are complementary, but focusing on design-time it can be said that they overlap. At design-time, which is in the focus of my research, the boarder between CBSE and MBSE is fuzzy. Here the two approaches share the notion of models, as CBSE is striving to develop models of components and their interactions to support composability [1].

1.2 My research topic

My research topic is at the point where CBSE and MBSE approaches intersect and focus on EFPs of embedded systems. Software components can be specified in a way that the specification is independent from the deployment technology. This makes it possible for one component specification to be reused for different implementations, in the spirit of MBSE. Different implementations imply different EFPs. In this research I focus on early (design-time) prediction of EFPs of embedded systems, using both CBSE and MBSE principles. I plan to investigate how EFPs of component-based embedded software systems change with different implementations. I will propose an approach where MBSE concepts are used for specifying EFPs on both platform-independent and platform-specific level, to enable EFP prediction for different implementations of an embedded system. In this case, on the platform independent level, the EFP framework proposed by Sentilles et al. [2] may be used.

1.3 Research methodology

As preparation work, I have defined an implementation of a component model for embedded systems, in particular a JavaBeans [3] implementation of SaveCCM [4]. Through this work I had gotten initial understanding of the core CBSE and MBSE concepts. The work resulted with an accepted paper at the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA09) [5].

The first step of the research, gaining detailed knowledge of the research area, has partly been done, as I have studied CBSE in depth, resulting with a technical report [6]. I have also begun studying MBSE in detail. Further, I plan to look into platform modeling and prediction theories.

Parallel to the above, I need to define in more detail the research problem within the research area to focus on. The research is done under the DICES [7] and PROGRESS [8] projects. Both projects share a general goal – to improve development of predictable component-based embedded software systems. However, PROGRESS is mainly focused on the real-time domain (and thus C as the
implementation language), while DICES addresses the soft real-time domain (and Java as the implementation language). Therefore, for PROGRESS worst-case values are of interest, while DICES is interested in average values. While narrowing the research problem, I need to choose one of these two domains. In other words, I will start identifying a particular question (particular questions) that my research will address by deciding which EFPs and which platforms I should concentrate on.

Theories and/or methods developed as answers to the research questions will be validated on case-studies.

2 Research area overview

2.1 Related work

Work related to my research can be divided into two directions: (i) component models for embedded systems with support for EFPs, and (ii) model-based approaches for embedded systems with support for EFPs. Here I give an overview of both directions.

2.1.1 Component models for embedded systems with support for EFPs

CBSE has proven to be quite successful in the desktop-, Web- and enterprise application domains (e.g. JavaBeans [3], .NET [9], Enterprise Java Beans [10]). Component models used in these domains usually focus on enabling design phase simplicity, relying on powerful hardware to handle the model overhead. However, most embedded systems have very limited memory and processing power at their disposal, and they are often subject to real-time constraints or even have a safety-critical role. These features are not considered by the aforementioned component models, thus emphasizing the necessity to develop special component models used in the embedded systems domain. Component models for embedded systems (e.g. Koala [11], Rubus [12]) are currently a hot topic in the scope of CBSE. In particular, handling EFPs is in the focus of ongoing research. Here I briefly overview several component models for embedded systems that have support for EFPs.

**BlueArX**

BlueArX [13] is a component model developed and used by Bosch for real-time embedded automotive applications, for example in engine control systems or chassis systems. These are closed control loop systems, meaning that they receive physical values from sensors, perform computations and then control actuators with new physical values.

The component model supports two types of components: atomic and structural. An atomic component is a unit of specification that has an implementation (in C), while a structural component is a unit of specification that has a decomposition (i.e. it is composed from several atomic and/or structural components). Each component consists of the specification, documentation and implementation/decomposition. Components communicate using messages (global variables) or service calls (function calls).

BlueArX interfaces are divided into two types: import and export. An import interface specifies variables, messages, services calibration parameters, etc. required by a component to execute, while an export interface specifies the same types of elements that a component provides.

BlueArX focuses on design-time, through the concept of signal flows. The idea is to use signal flow visualization to provide crucial behavior information on the component level, and get an explicit functional view from implicit component specifications.

EFPs in BlueArX are handled through Analytic Interfaces. An Analytic Interface is used to store a
component's EFPs. EFP values are specified in XML. Since EPF values have dependencies on the hardware platform, compiler, software context etc., the context has to be specified. Analytic Interfaces in BlueArX systems are connected to Reasoning Frameworks, which are used for various EFP consistency checks.

**ProCom**

ProCom [14] is a component model developed at Mälardalen University, intended for control-intensive distributed embedded systems, and designed to cover the whole development process of systems in the vehicular-, automation- and telecommunication domains. Typically, these systems have different characteristics at different levels of granularity. ProCom addresses this problem by introducing two layers: ProSys and ProSave. Both layers are hierarchical.

ProSys is the upper layer, which models a system as a number of active and concurrent subsystems which communicate by asynchronous message passing. ProSave is the lower layer, which models the internal design of a single ProSys subsystem. ProSave components are passive, reusable units of functionality that can either be realized by code (C functions), or by interconnected sub-components. They use the pipe-and-filter communication paradigm and are typically not distinguishable as individual units in the final executing system. The functionality of a ProSave component is offered by a set of services. ProSys and ProSave are connected at the lowest level of ProSys, where a ProSys subsystem is internally modeled by ProSave components.

In [2] Sentilles et. al. propose a generic framework (not tied to a particular component model) for handling EFPs in component models. They exemplify their approach by integrating it with ProCom and ProCom's integrated development environment. The framework is expandable in the sense that it allows specification of new EFP types. It can handle several values for a single EFP (for instance in early stages the value can be an estimation, while later it can be a measure), and also specify the context under which a certain EFP value is valid.

Sentilles et. al. define an EFP (or attribute as they refer to it) as:

\[
\text{Attribute} = \langle \text{TypelDentifier}, \text{Value}+ \rangle
\]

\[
\text{Value} = \langle \text{Data, Metadata, ValidityCondition*} \rangle
\]

TypelDentifier defines a class of attributes\(^2\). An attribute belongs to a single attribute type only. Available attribute types are kept in a registry of attribute types. These can, for example, be power consumption, worst-case execution time or static memory usage.

Data contains a concrete value for an attribute. Both primitive and complex types of data are supported. Metadata distinguishes between several values for the same attribute and describes how a value was obtained. ValidityCondition describes the conditions under which a certain value is valid.

This attribute framework is closely connected to my work as I may use it for specifying EFPs on the platform independent level. However, this has to be given more consideration.

**Robocop**

Robocop [15] is a component model for high volume consumer electronics developed at the Eindhoven University of Technology. It defines components as units of trade. A Robocop component is a set of different models that are related to each other. These models address different aspects of a component that can be of interest to different stakeholders, for example interface definition, behavioral models, resource consumption models, etc. A special type of model is the executable component. An executable component represents an implementation and can be executed. A Robocop component can contain multiple executable components that are targeted for execution.

---

\(^2\) The relation between attribute type and attribute is equivalent to the relation between class and object in OOP.
different platforms or operating systems. The functionality of executable Robocop components is encapsulated in services. To expose that functionality, services define interfaces they provide. Interfaces are groups of semantically related named operations. Robocop does not support hierarchical composition.

EFPs of Robocop components can be given in models that the component consists of. These extra functional models can include timeliness, reliability, safety, security and resource consumption. Robocop implements resource management through the Resource management framework. The aim of this framework is to prevent resource overloads on embedded devices that support dynamic updates or upgrades. It introduces a notion of resource-aware consumers, which are application entities that have information about resources needed for its operation. A special type of such entities are the quality-aware consumers, which consume different amount of resources depending on the level of quality they provide in a given moment. The consumers can register their resource needs to the framework, which can then guarantee them requested resources or deny their request. The framework can also optimize system quality depending on the available resources.

**Rubus**

The Rubus Component Model [16] is developed by Arcticus Systems in cooperation with Mälardalen University. It is intended for development of distributed, resource-constrained, embedded control systems, with a mix of hard-, soft- and non real-time requirements.

Architectural elements of Rubus are collectively named software items (SWIs). Basic Rubus components are called software circuits (SWCs). Each SWC is defined by its behavior, internal state data and interface. An SWC can have multiple behaviors, each one represented by a specific C entry function. Internal state data is used to preserve data across multiple executions. Assemblies (ASMs) and composites (CMPs) provide the means for hierarchical composition.

Rubus interfaces are port-based. Rubus distinguishes between data- and trigger ports, which capture data- and control flow, respectively.

A system is a top level hierarchical entity that describes the software logic and the software architecture (not the hardware architecture showing which SWIs execute on which hardware nodes) of a complete distributed system. At run-time a system executes in one out of a set of predefined modes, and can make transitions (mode shifts) from one mode to another. Modes are means to distinguish different states or conditions of a system – for instance a system executes a certain type of functionality during start-up, another type during normal operation, and a third type during errors.

Timing properties of SWCs and real-time requirements on the execution can be specified. Regarding the former, to enable timing analysis at design time, each SWC is associated with a run-time profile describing its run-time properties on different platforms. The latter are specified within the context of an ASM/CMP as bounds on time from the generation of a trigger signal to the generation of another trigger signal.

**SaveCCM**

SaveCCM [4] is a component model developed at Mälardalen University, intended for embedded control applications in vehicular systems. It is a simple component model that limits the flexibility of modeling to enable analyzability with respect to timing.

The main architectural elements of SaveCCM are components, switches and assemblies. The interface of an architectural element is defined by a set of input- and output ports. SaveCCM is based on the control flow (pipe-and-filter) paradigm, but data transfer and control flow are separated. The former is captured by typed data ports, and the latter by trigger ports. This is similar to Rubus described above.
Components represent basic units of encapsulated behavior. For basic components the functionality is typically defined by an entry function written in C. There are also composite components, for which the functionality is defined by an internal composition of subcomponents. Switches enable dynamic modification of the connections between components by providing means for conditional transfer of data and/or triggering. Assemblies are encapsulated subsystems, a mechanism for naming a collection of components and hiding the internal structure.

Regarding EFPs, SaveCCM focuses on timing properties. It supports specification and analysis of timing properties. They can be analyzed at design time using the UPPAAAL Port model checker [17]. There is also support for generic specification of EFPs. An EFP is represented as a triple $<\text{Attribute}, \text{Value}, \text{Credibility}>$ where Attribute is the property name, Value holds the property data, and Credibility gives the confidence measure with which the property represents the real value.

**Palladio**

Palladio [18] is not a component model for embedded systems, but is described here since it is designed to enable early performance predictions for component-based software architectures of business information systems, and early performance predictions are in the focus of my research. Development of the model started in 2003 at the University of Oldenburg and is since 2006 continued at the University of Karlsruhe.

Two key features of PCM are:

- parameterized component quality-of-service (QoS) specification, and
- developer role concept.

The former is a special QoS specification for software components, which is parametrized over environmental influences that are unknown to component developers during component design and implementation. This specification is called resource demanding service effect specifications (RDSEFF). Regarding the latter, PCM is aligned to different roles involved in component-based development and as such distinguishes between the following roles:

1. component developer,
2. system architect,
3. system deployer, and
4. domain expert.

A component developer specifies the functional- and extra-functional properties of components, which results in a repository model. A system architect assembles component specifications to form an assembly model. A system deployer specifies the resource environment to which the system will be deployed, providing a resource environment model. The system deployer also models the allocation of components from the assembly model to different resources of the resource environment, resulting in an allocation model. A domain expert is familiar with users of the system, and provides a usage model describing usage scenarios. From these partial model, a model of the complete system can be derived and then analyzed in terms of performance using multiple analysis methods, such as queuing networks, stochastic regular expressions or stochastic process algebra.

RDSEFFs abstractly model the externally observable behavior of a component. They specify: how a provided service calls the required services of a component, resource usage, transition probabilities, loop iteration numbers and parameter dependencies, all this to allow accurate performance predictions. RDSEFFs can be considered as a domain-specific modeling language which the component developer uses to specify performance related information for components. They represent the gray-box view of components.
2.1.2 Model-based approaches for embedded systems with support for EFPs

The second direction of related work, namely model-based approaches for embedded systems with support for EFPs, has not yet been studied in depth as the CBSE direction. Therefore I give only a short overview of several approaches here, as studying this direction is still ongoing work.

**MARTE**

MARTE (Modeling and Analysis of Real Time and Embedded systems) [19] is a UML profile defined by OMG that enables the use of UML for model-based development of embedded systems. It provides facilities to annotate models with information required to perform various analyses. It focuses on performance- and schedulability analysis, but enables any kind of quantitative analysis.

**DECOS**

DECOS (Dependable Embedded Components and Systems) [20] is a project whose goal is to enable development of dependable components that will be applied in various control systems in the automotive and aerospace/avionics industries. The DECOS tool-chain is based on a model-driven approach. The PIM to PSM mapping process is used for allocation of software jobs to hardware nodes, in such a way that the jobs are mapped to a minimal set of nodes while fulfilling all the allocation constraints. This definition of EFPs on the platform independent and platform specific levels in order to facilitate prediction is the connection to my research.

2.2 Research groups

CBSE for embedded systems is the subject of research both in industry and academia. Researchers in this field include:

- Michel Chaudron at Leiden University
  - ROBOCOP component model
- Ivica Crnković at Mälardalen University
  - SaveCCM and ProCom component models
- Kung–Kiu Lau The University of Manchester
- Rob van Ommering at Philips Research Laboratories
  - Koala component model
- Heinz Schmidt at RMIT University
- Judith Stafford at Tufts University

Most relevant researchers in the field of MBSE for embedded systems are the lecturers of the International School on Model-Driven Development for Distributed Realtime Embedded Systems (MDD4DRES). Here I list some of them:

- Kim G. Larsen at Aalborg University
- Huascar Espinoza at CEA (French Atomic Energy Commission)
- Stefan Kuntz at Continental Automotive GmbH in Regensburg
- Lothar Thiele at the Swiss Federal Institute of Technology, Zurich
2.3 Central literature

The idea behind building software from prefabricated components is not new – it originates from a paper published by Douglas McIlroy at the NATO conference on software engineering in Garmisch, Germany in 1968. However, CBSE is a relatively new discipline. One of the seminal CBSE papers is The Current State of CBSE by Allan Brown and Kurt Wallnau published in IEEE Software in 1998.

Two fundamental books in the CBSE area are:

- Clemens Szyperski, Component Software, Beyond Object-Oriented Programming, Addison-Wesley, 2002
- Ivica Crnković, Magnus Larsson, Building reliable component-based software systems, Artech House, 2002

A collection of papers that cover CBSE for embedded systems is given in:


A general book on MBSE is:


A collection of papers providing a state of the art overview of MBSE is given in:


An overview of MBSE for embedded systems is given in the following book:

- Sebastien Gerard, Jean-Philippe Babau, Joel Champeau: Model Driven Engineering for Distributed Real-Time Embedded Systems, Wiley, 2005

2.4 Key conferences

Since my research is under the scope of CBSE and MBSE, accordingly conferences in these areas and general software engineering conferences are of interest. Given the organization of conferences changes over the years, the information about tracks presented here generally applies to current instances (last and next few) of a particular conference.

**CBSE (International Symposium on Component-Based Software Engineering)**

The main conference in the CBSE field. Is it federated with the QoSa (Quality of Software Architecture) conference under a common event called COMPARCH, in order to encourage collaboration with the software architecture community.

**ESEC/FSE (The joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE))**

A joint venture of two general software engineering conferences. The most interesting part for me is the CBSE track.

**ICSE (International Conference on Software Engineering)**

The premier general software engineering conference. It has a large number of tracks, the Software components and reuse track being of most interest to me.
MODELS (ACM/IEEE International Conference on Model Driven Engineering Languages and Systems)

The main conference devoted to the MBSE field.

SEAA (EUROMICRO Conference on Software Engineering and Advanced Applications)

A general conference on software engineering, having a special track devoted to CBSE (Service and Component Based Software Engineering track (SCBSE)). Also, as part of the SCBSE track, there is the Model Driven Engineering track.

References


[5] Juraj Feljan, Jan Carlson, Mario Žagar, Using JavaBeans to Realize a Domain-Specific Component Model, 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2009


International Symposium on Component Based Software Engineering, 2008


