PRIDE

Ivica Crnković, Séverine Sentilles, Thomas Leveque
Mälardalen research and technology centre
PO Box 883, SE-721 23, Västerås, Sweden
{ivica.crnkovic, severine.sentilles, thomas.leveque}@mdh.se

Mario Žagar, Ana Petričić, Juraj Feljan, Luka Lednicki
Faculty of Electrical Engineering and Computing
University of Zagreb, Croatia
{mario.zagar, ana.petricic, juraj.feljan, luka.lednicki}@fer.hr

Josip Maras
Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture
University of Split, Croatia
josip.maras@fesb.hr

Abstract: **This paper describes PRIDE, an integrated development environment for efficient component-based software development of embedded systems. PRIDE uses reusable software components as the central development units, and as a means to support and aggregate various analysis and verification techniques throughout the whole lifecycle - from early specification to deployment and synthesis. This paper focuses on support provided by PRIDE for the modeling and analysis aspects of the development of embedded systems based on reusable software components.**

1 INTRODUCTION

In the recent years, embedded system (ES) development has changed significantly due to the rapid increase of software in these systems. This has resulted in software becoming as complex as in conventional systems. In non-embedded domains, new approaches such as model-based, component-based, and service-oriented development have been proposed to manage software complexity and there is a trend to apply these approaches also in ES development. However, due to the specific requirements of ES, these approaches are insufficient. In particular, ES correctness is strongly correlated to specific extra-functional properties (EFPs) such as timing (e.g. execution and response time) or dependability (e.g reliability and safety) under constrained resources such as memory, energy, or computation speed. This calls for additional domain-specific technologies that provide support not only for functional development but also for analysis and verification of EFPs.

As a possible solution, we have been developing a new component-based approach [1] built around a two-layer component model called ProCom [2], which addresses the particularity of ES development from big complex functionalities, to small, close to control loop functionalities. This approach requires specific tool support that should enable:

- efficient system design by using existing components,
- seamless integration of different tools to provide the analysis and verification required for system correctness, and
- efficient EFP management of components and systems.

In this paper, we introduce the ProCom Integrated Development Environment, PRIDE[1], a tool-suite supporting this approach. We focus on the support PRIDE provides concerning two main aspects of developing ES with reusable software components - modeling and analysis. In difference to similar approaches ([3], [4], [5]), PRIDE puts emphasis on EFPs during the entire lifecycle.

In Section 2 we describe the design strategies that drove the development of PRIDE. We describe PRIDE's modeling support in Section 3 and analysis support in Section 4. The paper is concluded in Section 5. In Appendix A we give a description of the tool demonstration.

2 PRIDE DESIGN STRATEGIES

PRIDE has been designed to support four design strategies that are especially important to consider for having an efficient component-based development of embedded systems.

Levels of abstraction. Using components throughout the whole development process implies that the component concept spans a wide range of abstractions, from a vague and incomplete early specification, to very "concrete" with a fixed specification, a corresponding implementation and information about their EFPs. This means that components at different levels of abstraction must be able to co-exist within the same model.

Component granularity. In distributed ES, components span a large variety in size and complexity; the larger components are typically active (i.e. with their own thread of execution) with an asynchronous message passing

1 PRIDE Web page: www.idt.mdh.se/pride

communication style, whereas the smaller components are responsible for a part of control functionality with a strong synchronization. For an efficient development, a support for handling different types of components must be provided.

Component vs. system development. The common distinction between component development and system development brings issues in ES development, where the coupling between the hardware platform and the software is particularly tight. As a consequence, component development needs some knowledge of where the components are to be deployed. This requires support to handle the coupling between components, system and target platform, while still allowing separate development of components and systems.

Extra-functional properties. The correctness of ES is ascertained based on both the functional and extra-functional aspects. However, many EFPs typically encountered in ES are assessed through different methods during the development lifecycle (from early estimation to precise measurements) and different values may be obtained according to the characteristics of the resources of the platform on which the components are to be deployed. For this reason, an efficient ES development should provide a means for specifying, managing and verifying these properties with respect to the context in which their values are provided.

To comply with the design strategies, the following requirements have been identified as principles that guided the design and development of PRIDE:

- allowing to move freely between any development stages,
- displaying the consequences of a change in the system or within a component,
- supporting the coupling with the hardware platform, and
- enabling and enforcing the analysis, validation and verification steps.

In addition, a central requirement relates to the notion of component. Components are the main units of development and seen as rich-design artifacts that exist throughout the whole development lifecycle, from early design stage, in which little information about them exists, to deployment and synthesis stages, in which they are fully implemented. PRIDE views a component as a collection of all the development artifacts (requirements, models, EFPs, documentation, tests, source code, etc.), and enables their manipulation in a uniform way.

Driven by the aforementioned principles, several tools have been developed and tightly integrated into PRIDE. PRIDE is built as an Eclipse RCP application that can be easily extended with addition of new plugins.

3 MODELING WITH PRIDE

PRIDE's modeling part currently consists of a component explorer and component editors.

Component Explorer. It enables browsing the list of the components available in the current development project. In it a component owns a predefined information structure consisting of a source folder for source code, a model folder to store the architectural model, and other models such as resource usage models, behavioral models, etc., a documentation folder, and a metadata file, which contains specific properties of the component such as its creation times, its version number, etc. This structure is extendable.

Component Editors. PRIDE is built around ProCom, a hierarchical component model that additionally distinguishes between two types of components: ProSys components, i.e high granularity level components to develop complex functionalities possibly distributed, and ProSave components, i.e. non-distributed, smaller and simpler components. However, in the component editors, all these components are treated in an uniform way. Each component editor partitions the components in two views. The "external view" provides all the pieces of information about component functionality such as the component name, its interface and EFPs. The "internal view" depends on the component realization. For primitive components, the internal view is linked to the component implementation and the source code is displayed. For composite components, the internal view corresponds to an interconnection of subcomponent instances and a graphical form is made available allowing to make modifications in this inner structure (addition/deletion of component instances, connectors, change in the connections, etc.).

4 ANALYSIS WITH PRIDE

The analysis support in PRIDE is based on two main parts, an attribute framework and an analysis framework.

Attribute Framework. The purpose of the attribute framework [6] is to provide a uniform and user-friendly structure to seamlessly manage EFPs in a systematic way. The attribute framework enables attaching extra-functional properties to any architectural element. Attributes are defined by attribute types, and include attribute values with metadata and the specification of the conditions under which the attribute value is valid. One key feature is that the attribute framework allows an attribute to be given additional values during the development without replacing old values.

The analysis framework provides a common platform for integrating in a consistent way various analysis techniques, ranging from simple constraint checking and attribute derivation (e.g., propagating port type information over connections), to complex external analysis tools. Analysis results can either be presented to the user directly, or stored as component attributes. They are also added to a common analysis result log, categorized as Ok, Error or Warning, allowing the user easy access to earlier analysis results.

4.1 Analysis example: Parametric component-level WCET analysis

Worst case execution time (WCET) is a crucial property in real-time systems, since it serves as the basis for schedulability analysis. In practice, it is often determined by extensive testing and measuring, but there are also methods to derive safe approximations by means of static code analysis. In either case, the information is only available late in development, once all parts of the system are fully implemented.

In early development stages, WCET analysis can be performed at component-level, based on WCET information for individual components (estimates in the case of components under construction, or the result of code analysis or measurements in the case of reused components). The attribute representing the WCET of an individual component is expressed in parametric form with respect to component input, in order to facilitate reuse of this attribute when the component is reused. It is possible to specify, for example, that the WCET is 150 if $x<10$ and $100+5x$ otherwise, where x denotes an input port value. In a similar way, component outputs can be specified in terms of their inputs. Based on these attributes, and the component interconnections, the analysis tool derives a WCET value for the composite structure. The result is stored in an attribute of the composite, and presented to the user via the analysis result log.

5 CONCLUSION

The key benefits of PRIDE lay in its domain-orientation: (i) it facilitates bringing design decisions related to EFPs and system constraints such as resources usage or timing characteristics. Designers can in an early phase of development investigate different choices before component implementation by estimating component properties. PRIDE also enables (ii) system design that consists of already existing components and components that still do not exist; (iii) a separation of development of software from system development, and yet allowing reasoning about the system properties; (iv) a separate development of software components from software systems and reusing not only the code, but also of their EFPs and other development artifacts such as models; (v) design of local and distributed embedded systems that might have different concerns on different levels; and (vi) an iterative development process.

The PRIDE toolset, through the attribute framework and the analysis framework, provides a consistent interface to the various analysis tools used to increase predictability during the development process.

The WCET analysis exemplifies the use of parametric attributes for capturing information that can be reused together with the component in different systems, while still allowing for sufficiently detailed analysis in a particular context. It also shows how early analysis on component level can be based on a combination of detailed information about reused components and estimates or budgets for components under development.

Additional tools will be integrated into PRIDE as part of our future work, including analysis of failure propagation, code-level WCET analysis, and model checking of behavioral models.

REFERENCES

[1] T. Bureš, J. Carlson, S. Sentilles, A. Vulgarakis. A Component Model Family for Vehicular Embedded Systems, The Third International Conference on Software Engineering Advances, IEEE, 2008

[2] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, I. Crnković, A Component Model for Control-Intensive Distributed Embedded Systems, Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008), Springer Berlin, 2008

[3] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, M. Tivoli, The SAVE Approach to Component-Based Development of Vehicular Systems, Journal of Systems and Software, 2007

[4] Arcticus Systems, Rubus Software Components, http://www.arcticus-systems.com

[5] R. van Ommering, F. van der Linden, J. Kramer, J. Magee. The Koala Component Model for Consumer Electronics Software, Computer, 2000

[6] S. Sentilles, P. Stepan, J. Carlson, and I. Crnkovic, Integration of extra-functional properties in component models, The 12th International Symposium on Component Based Software Engineering, Springer Berlin, 2009

[7] Séverine Sentilles, Anders Pettersson, Dag Nyström, Thomas Nolte, Paul Pettersson, Ivica Crnković, Save-IDE - A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems, Proceedings of the Research Demo Track of the 31st International Conference on Software Engineering, 2009

6 APPENDIX - TOOL DEMONSTRATION DESCRIPTION

We will demonstrate the core support of PRIDE: (i) design of distributed embedded systems using the ProCom component model, (ii) assigning quality attributes to components and other architectural elements. The demonstration will be implemented by an example: an autonomous truck navigation system (Figure 1). A similar example was demonstrated by another component model, namely SaveCCM [7]. However, in PRIDE we also show advantages of the ProCom component model (e.g. use of reusable components), and the use of the attribute framework built in PRIDE.

The truck is intended to follow a straight black line with two filled black circles on each end. The truck has to follow the line until it reaches its end. Then it turns around, signals the turning with blinking lights, and starts searching for the line again. The truck executes the navigation running in three different operational modes, namely:

- *Follow* mode in which the truck aligns itself with the line once it reaches it, and then follows the line using its line sensors. When the truck detects the end of the line, it changes mode to the Turn mode.
- *Turn* mode in which the truck turns for a fixed distance and signals the turning until it reaches a state where it is able to go straight to the line again. Upon completion, the truck changes mode to the Find mode.
- *Find* mode in which the truck goes straight to the line. When the line is reached, the truck returns to the Follow mode where it aligns itself with the line and keeps following it.

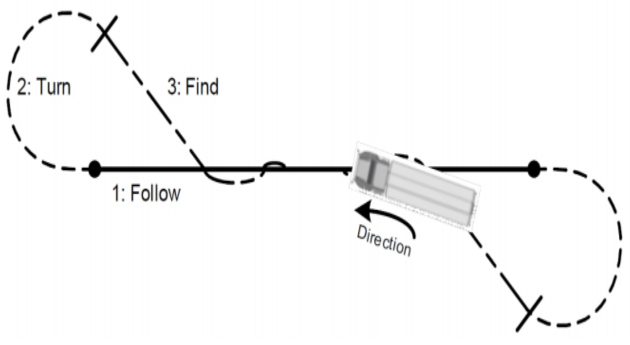


Figure 1: The truck

The truck has four sensors - a speed sensor and three line sensors (left, right and middle one). Line sensors are able to detect the line on the surface, and are used to determine if the truck has a correct position with respect to the line. The truck also has two moving actuators, one for steering and one for speed; and two light actuators that can turn on and off the blinking lights.

In the demonstration we will show how to build the matching system: first by decomposing it into two high level functionalities: The “Movement” and the “Lights“ components which will be modeled as ProSys components (Figure 2). Next, we will show how to model and implement their internals (Figure 3) by composing a system from a combination of pre-existing and newly developed components. Each component can have a number of quality attributes. In this example we will use worse-case execution time (WCET) and static memory that can either be calculated (using for example static source code analysis), derived by analysis, or estimated. We will show how to derive the mentioned quality attributes of the whole system from attributes of individual components.

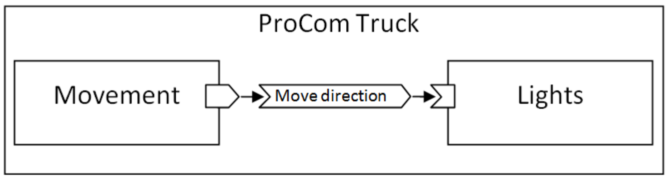
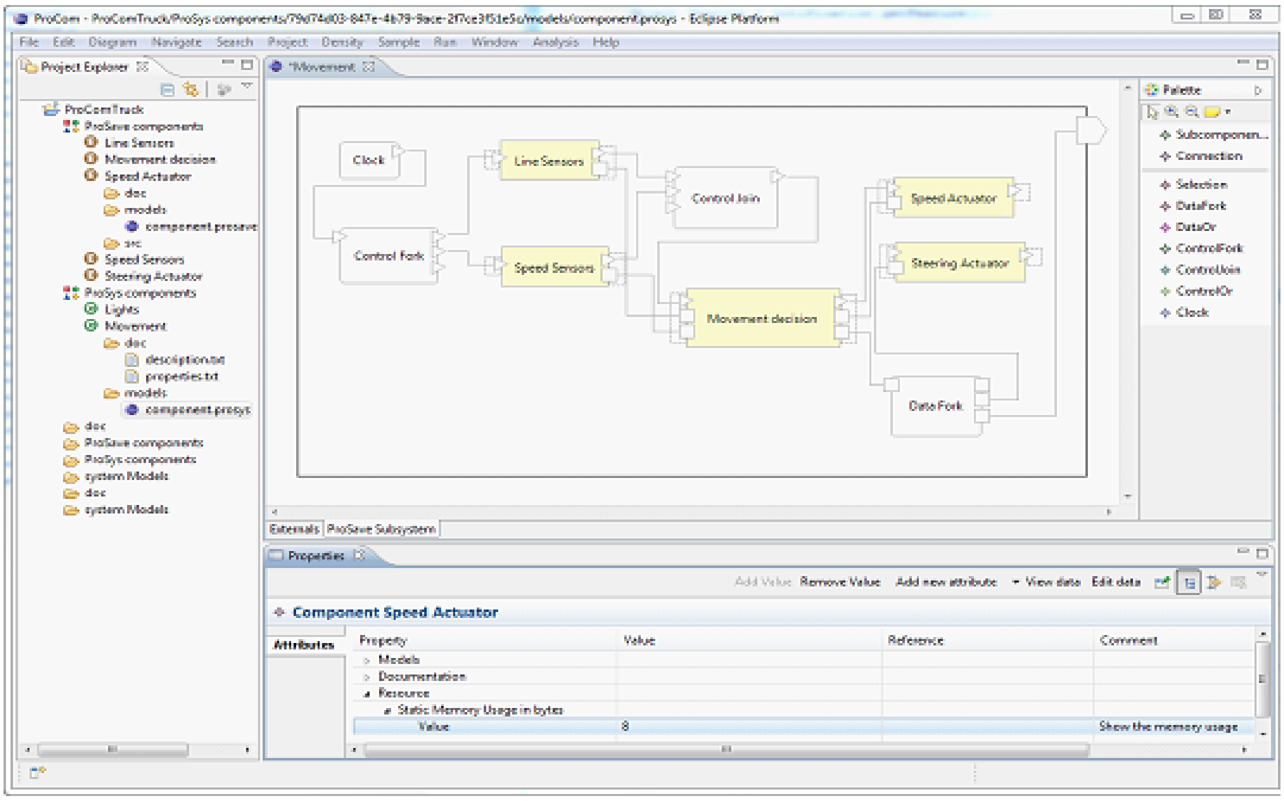


Figure 2: The ProSys components of the truck system



Property	Value	Reference	Comment
▹ Models			
▹ Documentation			
▴ Resource			
▴ Static Memory Usage in bytes			
Value	8		Show the memory usage

Figure 3: The internals of the Movement ProSys component