PRIDE – an Environment for Component-based Development of Distributed Real-time Embedded Systems*

Etienne Borde, Jan Carlson, Juraj Feljan, Luka Lednicki, Thomas Lévêque,
Josip Maras, Ana Petričić and Séverine Sentilles
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
firstname.lastname@mdh.se

ABSTRACT

Embedded system development is currently hampered by the lack of tools capable of conjointly catering for the complete design-verification-deployment cycle, extra-functional properties and reuse. To address these concerns, we have developed PRIDE, an integrated development environment for component-based development of embedded systems.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments — *Integrated environments*

Keywords

Component-based development, extra-functional properties, integrated development environment, embedded systems.

1. INTRODUCTION

Embedded systems have changed radically. Nowadays, they integrate more and more software functionality while still having to comply with severe resource constraints (e.g., memory, energy or computation speed) and dependability and real-time concerns. As a result, their development should simultaneously handle and ensure various aspects such as extra-functional properties (EFPs), distribution, reuse, and hardware and software dependencies. This makes embedded system development a very complex and time-consuming task. Especially since there is currently no tool that supports the complete set of needs for embedded system development, catering for the complete functional development cycle with consideration for EFPs and reuse. In particular, EFPs are often disregarded in industrial tools [1, 2].

Taking this into account, we have built the ProCom Integrated Development Environment (PRIDE) that addresses the particularities of embedded system development. PRIDE has been developed to support a new component-based approach together with its underlying component model called ProCom [7].

Contrasting many existing approaches for embedded system development [3, 4, 5, 6], reusability is a key concern in PRIDE, covering not only code reuse but also reuse of models, EFPs and analysis artifacts. Other key benefits of PRIDE include its ability to *i)* bring design decisions related to EFPs and system constraints such as resources usage or timing characteristics, allowing developers to investigate different design choices in an early phase of development by estimating component properties; *ii)* enable the design of distributed embedded systems; *iii)* enable reuse of not only the code from the components, but also their EFPs and other development artifacts such as models; and *iv)* enable mixing already existing components with components that are still not implemented.

Section 2 describes the basic underlying approach guiding the development of PRIDE, followed by a presentation of the tool and some of its key parts in Section 3. Section 4 concludes the paper by presenting some ongoing and future work.

2. OVERALL APPROACH

This section describes how the tool suite addresses the particularities of embedded systems development, focusing on three important development aspects: design, analysis and synthesis.

Design.

The trend in embedded system development is to implement more and more functionality in software, resulting in a continuously increasing complexity. To address the increasing complexity and to accommodate demands of shorter time-to-market, we base our approach on the component-based software engineering (CBSE) paradigm. CBSE promotes building systems not from scratch, but from predeveloped reusable software components, which should significantly shorten development time. Furthermore, management of complex systems should be facilitated by dividing them into smaller components that can be developed independently.

Our component-based approach is built around a two-layered component model called ProCom [7]. The upper layer models a system as a collection of active, concurrent and typically distributed subsystems that communicate by asynchronous message passing. The lower layer models the

*This work was supported by the Swedish Foundation for Strategic Research via the PROGRESS research centre, and by the Unity Through Knowledge Fund via the DICES project.

ICSE '11 Hawaii, Waikiki, Honolulu, USA

detailed structure of individual subsystems as a collection of interconnected passive components.

To benefit from the component-based approach throughout the whole development process, ProCom adopts a particular component notion. Components are rich design entities encapsulating a collection of many development artifacts, including requirements, models, extra-functional properties, documentation, tests and source code).

The tool suite supports this view of a component as a collection of development artifacts, and allows components of different maturity, from early specifications to fully implemented components with more detailed information, to be manipulated in a uniform way.

Analysis.

Many embedded systems are found in applications with high dependability requirements, and they are often subject to real-time constraints. Consequently, the development activities should be complemented by different analysis techniques to derive extra-functional properties of individual components and the system as a whole, to ensure the correctness of the system. These analyses are traditionally performed in late stages of the development, when detailed information is available, but analysis should also be used in early stages to guide the development process and to avoid costly late changes.

As a result of the rich design-time component concept of ProCom, component reuse also implies reuse of component properties and previous analysis results. In those cases where analysis of a component depends also on factors outside the component, special care must be taken to identify to what extent the reused information is still applicable in the new environment.

Synthesis.

Embedded systems typically have resource limitations, for example in terms of memory and processing power. This can be due to the fact that they are made in large quantities, and thus have to be cheap to produce. In other cases, resource limitations are a result of limits in physical size or battery lifetime.

Contrasting component models for desktop applications, these limitations imply that a component model targeting the embedded systems domain should not come with a high run-time overhead. To satisfy this requirement, our approach does not provide full-scale component support at run-time. Instead, the development process includes a synthesis phase, where the component-based design is transformed into a system realization based on tasks executed by standard real-time operating system. During the synthesis, various optimizations can be applied to adjust the code of a component to its context in this particular system.

3. AN OVERVIEW OF PRIDE

Based around ProCom and the described overall approach, we have developed several tools, tightly integrated into Pride. Pride is built as an Eclipse RCP application that can be easily extended with addition of new plugins. As shown in Figure 1, the core part currently consists of a component explorer, component editors, attribute- and analysis frameworks, and a synthesis tool. Figure 2 shows a screenshot from Pride, with some of these parts highlighted.

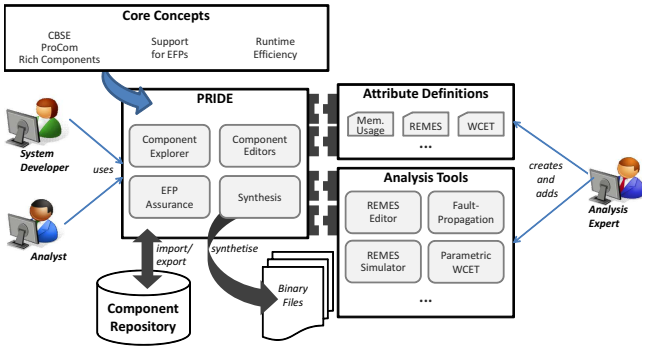

Figure 1: Architecture of Pride.

Component Explorer.

The component explorer enables browsing the list of components available in the current development project. In it, a component owns a predefined information structure that corresponds to the aforementioned rich component concept. This structure is extendable. The component representation also supports component versioning. Components can be exported from a project to a repository, making them available for reuse in other projects.

Component Editors.

The component editors are built around ProCom. Components from both layers are treated in a uniform way. Each component editor partitions the components in two views. The *external view* handles the component specification, including information such as the component name, its interfaces and EFPs. The *internal view* depends on the component realization. For composite components, the internal view corresponds to a collection of interconnected subcomponent instances, and a graphical editor is available allowing modifications to this inner structure (e.g., addition/deletion of component instances, connectors and connections). For primitive components, the internal view is linked to the component implementation in form of source code. Editing the component code is facilitated by features such as syntax highlighting and auto-completion, provided through the integration of the Eclipse C/C++ Development Tooling (CDT) plugins.

EFP Assurance.

Extra-functional properties are created, managed and ensured through two frameworks. The *Attribute Framework* provides a uniform and user-friendly structure to seamlessly define and manage EFPs in a systematic way, and to support the packaging of the development artifacts in the components. The attribute framework enables attachment of EFPs to any architectural element of the component model. Attributes are defined by an attribute type, 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 deriva-

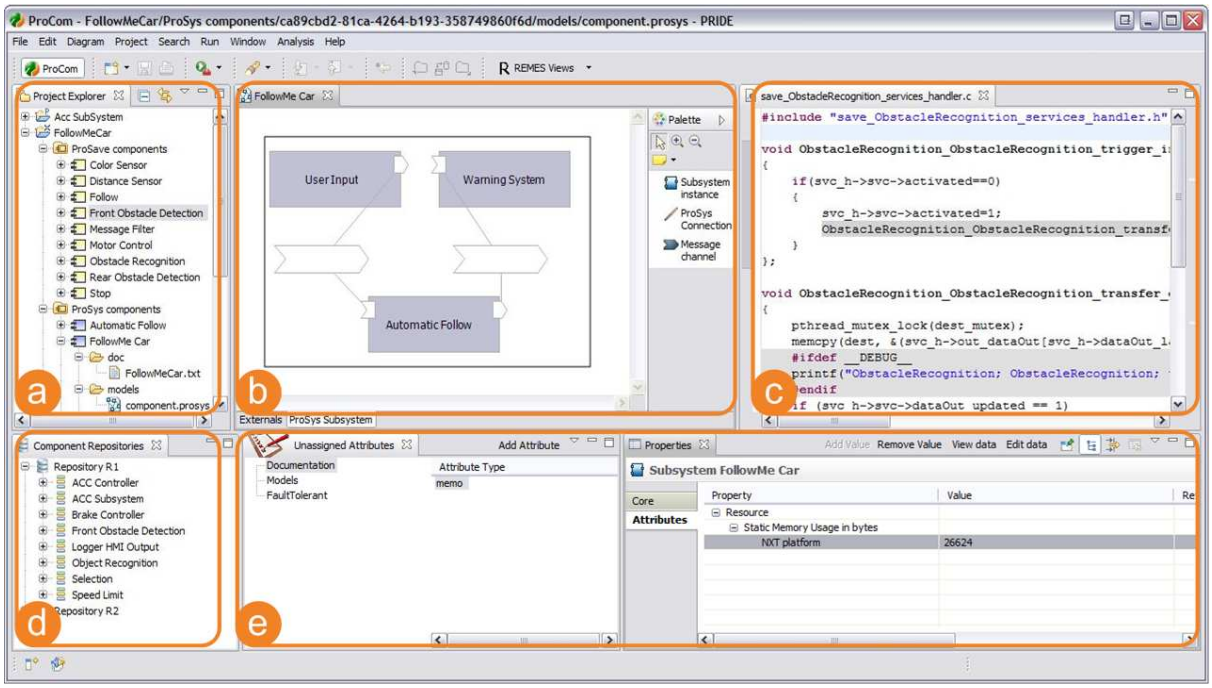

Figure 2: A screenshot of PRIDE showing *a)* the component explorer; *b)* a component editor; *c)* a code editor; *d)* the repository browser; and *e)* the attribute framework.

tion (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, allowing the user easy access to earlier analysis results.

Through the use of extension points in the analysis and attribute frameworks, PRIDE provides support to easily integrate new analysis techniques together with their associated extra-functional properties. The analysis techniques already integrated in PRIDE include parametric component-level worst-case execution time analysis, model checking of behavioural models, and fault-propagation.

Synthesis.

The synthesis part of PRIDE automates the generation of interfaces for primitive components in the lower layer, and generation of code for composite components in both layers. It also produces build configurations (in debug and release mode) for each level of composition.

Based on models of the physical platform and the allocation of components to physical nodes, the synthesis also produces the binary executable files of each node in the system. The synthesised code relies on a middleware that has been ported to different platforms, including POSIX-compliant operating systems, FreeRTOS and JSP.

4. CONCLUSIONS

We have presented PRIDE, a tool suite for developing embedded systems providing support for design, analysis and synthesis. A demonstration video is available from the PRIDE website (www.idt.mdh.se/pride), from where the tool suite can also be downloaded. A collection of publications presenting the research behind various parts of PRIDE is provided at the site as well.

Our ongoing work on PRIDE includes improving the support for deployment- and allocation modeling. We also plan to provide additional analysis techniques, including refactoring impact analysis, value domain propagation and response time analysis.

5. REFERENCES

[1] ArcCore. Arctic Studio. `http://arccore.com`.

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

[3] ESTEREL Technologies. SCADE Suite. `http://www.esterel-technologies.com/products/scade-suite/`.

[4] MathWorks. Simulink, MatLab and Real-Time Workshop. `http://www.mathworks.com`.

[5] Mentor Graphics. BridgePoint. `http://www.mentor.com/products/sm/model_development/bridgepoint/`.

[6] Object Management Group. A UML Profile for MARTE, Beta 2, June 2008. Document number: ptc/2008-06-09.

[7] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković. A Component Model for Control-Intensive Distributed Embedded Systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE'08)*. Springer Berlin, October 2008.