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

Settling down the software architecture for embedded system is a complex and time consuming task. Specific concerns that are generally issued from implementation details must be captured in the software architecture and assessed to ensure system correctness. The matter is further complicated by the inherent complexity and heterogeneity of the targeted systems, platforms and concerns. In addition, tools capable of conjointly catering for the complete design-verification-deployment cycle, extra-functional properties and reuse are currently lacking. To address this, we have developed PRIDE, an integrated development environment for component-based development of embedded systems. PRIDE is based on an architecture relying on components with well-defined semantics that serve as the central development entity, and as means to support and aggregate various analysis and verification techniques throughout the development — from early specification to synthesis and deployment. PRIDE also provides generic support for integrating extra-functional properties into architectural definitions.

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, integrating 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. All these aspects must be accordingly reflected in the software architecture to allow proper evaluation of the system correctness. 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 (e.g., [1]).

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 [11] where reusability is a key concern. Other key benefits of PRIDE include its ability to *i)* capture and track design decisions related to EFPs and system constraints, such as resources usage or timing characteristics, at early stages; *ii)* consider distribution aspects; *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 ongoing and future works.

2. OVERALL APPROACH

The PRIDE approach aims to cover the whole development process, supporting the considerations of predictability and safety throughout the development starting from a vague specification of a system based on early requirements up to its final and precise specification and implementation ready to be deployed. This section describes how the tool suite addresses the particularities of embedded systems development, focusing on three important development aspects: software architecture, analysis and synthesis.

Software Architecture.

To address the increasing complexity of embedded systems and to accommodate demands of shorter time-to-market, we base our approach on the component-based software engineering (CBSE) paradigm. CBSE promotes building sys-

*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.

WICSA '11 Boulder, Colorado, USA

tems not from scratch, but from pre-developed software components. Management of complex systems should be facilitated by dividing them into smaller components that can be developed independently and reused in different contexts. Reusability is one of the key concepts in PRIDE, aiming to significantly shorten development time. The tool introduces the distinction between component type and component instance. Each usage (or reusage) of a component type creates a component instance of the given type, and by editing the component all its instances are affected. To foster reusability, components can be stored in (and imported from) a shared repository, making them available for reuse in different projects.

Our component-based approach is built around a two-layered component model called ProCom [11]. Owing to the embedded systems domain, we consider that the software architecture must be able to provide both a high-level view of loosely coupled subsystems and a low-level view of control functionality associated with a particular piece of hardware.

The upper layer of ProCom models a system as a collection of active, concurrent subsystems that communicate by asynchronous message passing, and are typically distributed. A subsystem can internally be realized as a hierarchical composition of other subsystems or built out of entities from the lower layer of ProCom.

The lower layer models the detailed structure of individual subsystems. Here, components are passive and represent smaller and simpler units of functionality with clearly defined interface to the environment. Internally, component functionality can be realised either by C code or as a composition of subcomponents. In this layer, ProCom provides an explicit separation of control and data flow. This separation concept together with different types of connectors and the simple structure of components makes it possible to explicitly specify, and then analyse, control flow, timing properties and system performance.

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 development artifacts, including requirements, various models (e.g. architectural, behavioural and timing), extra-functional properties (predicted or experimentally measured values), documentation, tests and source code, thus making a component a unifying concept throughout the development process. 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 co-exist within the same model and to be manipulated in a uniform way. Additionally, PRIDE provides an ability to leave component realization undecided and thus postponing component realization decision for later.

What particularly distinguishes PRIDE from other modelling tools is the support for managing extra-functional properties through the attribute framework [10]. Every component can include a collection of structured attributes defining simple or complex types of properties such as behavioural and resource models, certain dependability measures, different timing attributes and documentation. These attributes can be associated with any element of the software architecture such as a specific port, service or the component as a whole. New user-defined attribute types can also be added to the model.

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.

Traditionally, these analyses do not consider the software architecture and are performed in late stages of the development, when all detailed information is available. However, finding design flaws in late stages of development can result in need for costly changes or even complete change of the software architecture. PRIDE supports designing systems out of software components of different maturity, from components with existing deployable code to components with no realisation defined at all. This allows us to perform different analyses in early stages of development on the software architecture and provide early estimates on system behaviour and properties. In this way we can detect possible problems before the system is implemented and avoid 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. In some cases, this is due to the fact that they are produced in large quantities, and thus have to be cheap to produce. In other cases, resource limitations are the 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. PRIDE can be extended by adding new extra-functional properties (attribute definitions) together with their corresponding analysis support when needed. Figure 2 shows a screenshot from PRIDE, with

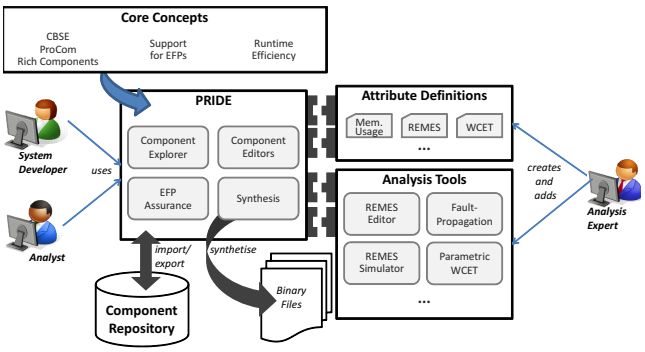

Figure 1: Architecture of PRIDE.

some of these parts highlighted.

Component Explorer.

The component explorer enables browsing the list of components available in the current development project. In it, a component owns a predefined and extensible information structure that corresponds to the aforementioned rich component concept. The component explorer also supports component versioning and importing and exporting of components from a project to a component repository, making them available for reuse in other projects.

Component Editors.

The component editors are used for developing an architectural model of components and a system as a whole. They are built around the ProCom component model and represent one of the central parts of PRIDE. Components from both ProCom layers are treated in a uniform way. The component editor provides two independent views on a component, *external* and *internal view*, thus alowing the separation of concerns. The *external view* handles the component specification, including information such as the component name, its interfaces and EFPs. The *internal view* focus on component internal structure implementing its functionality and it depends on the component realization type. 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.

Attribute Framework.

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 [10]. 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. This allows us to define early estimates for EFPs even before actual architectual element is implemented. Such values can be used for analysis in early stages of system development. Later, when the element is more mature, we can add more refined values for EFPs allowing us to conduct more accurate analysis.

Analysis Framework.

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, 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 [7], model checking of behavioural models [6], and fault-propagation [12].

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. RELATED WORKS

Contrasting many existing approaches for embedded system development [4, 8, 9], reusability is a key concern in PRIDE, covering not only code reuse but also reuse of parts of software architecture (models, EFPs and analysis artifacts together with the relations between them). PRIDE and TOP-CASED aim to be generic tools where many analysis can be integrated but TOP-CASED using SysML focuses on model checking while PRIDE integrates different kind of analysis including timing ones. BridgePoint[9] is a commercial tool based on xtUML which focuses on full code generation and strong optimization but lacks support of extra functional properties. Comparing to PRIDE, the Ocarina tool suite [5] using AADL integrates schedulability analysis. However PRIDE tries to support different aspects such as fault tolerant and consistency instead of focusing only on timing properties. While Ocarina is a tool chain, PRIDE is an integrated development environment which allow easy integration of new analysis. Although tools for East-ADL [3] include fault-propagation analysis and timing analysis , they are not integrated in a specific development environment. Focusing on the automotive domain only, Ar-

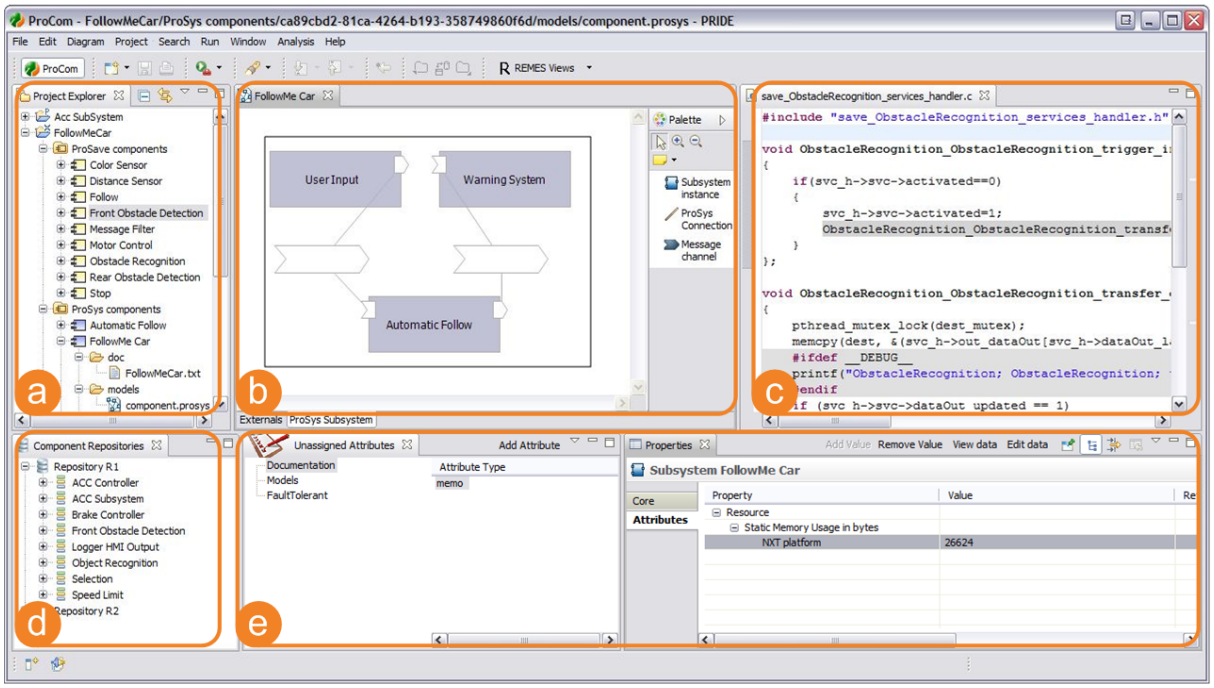

Figure 2: A screenshot of PRIDE showing *a)* the component explorer; *b)* a component editor; *c)* a code editor; *d)* the repository browser; and *e)* the attribute framework.

top (the AUTOSAR tool platform) [2] is intended to provide a unique framework to encompasses the heterogeneity of engineering needs. To do so, Artop provides code skeletons and testing tools without integration and management of extra-functional properties.

5. 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 and related publications can also be downloaded.

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.

6. REFERENCES

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

[2] S. Benz, M. Rudorfer, and C. Knuechel. Interoperable AUTOSAR tooling with Artop. *First workshop on hands-on platforms and tools for model-based engineering of embedded systems*, June 2010.

[3] P. Cuenot, D. J. Chen, S. Gérard, H. Lönn, M.-O. Reiser, D. Servat, C.-J. Sjöstedt, R. Kolagari, M. Törngren, and M. Weber. Managing complexity of automotive electronics using the EAST-ADL. In *12th IEEE International Conference on Engineering Complex Computer Systems*, pages 353–358, 2007.

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

[5] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. From the prototype to the final embedded system using the Ocarina AADL tool suite. *ACM Trans. on Embedded Computing Systems*, 7:42:1–42:25, August 2008.

[6] D. Ivanov, M. Orlic, C. Seceleanu, and A. Vulgarakis. REMES tool-chain – A set of integrated tools for behavioral modeling and analysis of embedded systems. In *25th IEEE/ACM International Conference on Automated Software Engineering*, 2010.

[7] T. Leveque, E. Borde, A. Marref, and J. Carlson. Hierarchical composition of parametric WCET in a component based approach. In *14th IEEE Int. Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, 2011.

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

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

[10] S. Sentilles, P. Štěpán, J. Carlson, and I. Crnković. Integration of extra-functional properties in component models. In *12th International Symposium on Component-Based Software Engineering*, pages 173–190. Springer-Verlag, 2009.

[11] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković. A component model for control-intensive distributed embedded systems. In *11th International Symposium on Component Based Software Engineering*. Springer Berlin, October 2008.

[12] M. Wallace. Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science*, 141(3):53–71, 2005.

APPENDIX

PRIDE Tool Status

PRIDE is a research tool which aims to integrate research results of PROGRESS project, to highlight their interconnection and complementarities and demonstrate the feasibility of an holistic and integrated approach for the development of distributed embedded systems. The main contribution of PRIDE is the integration of many research results in a same tool which covers all aspects of embedded system development while the other tools focus only on some specific points. For this reason, it is usable but is not as mature and reliable as a commercial tool. The tool can be downloaded from the PRIDE website `http://www.idt.mdh.se/pride`.

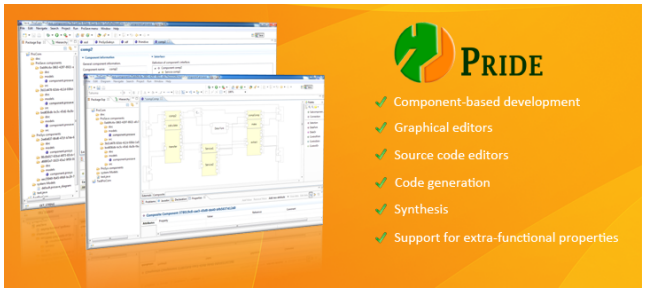

Figure 3: PRIDE.

The public release 0.4.1 includes the following features:

- ProCom graphical editors
- Attribute and analysis framework
- Parametric worst case execution time
- Fault propagation analysis
- Type Consistency analysis
- REMES graphical editors
- REMES analysis and simulations (in a dedicated PRIDE release)
- Deployment model editors
- ProCom synthesis

PRIDE Tool Demonstration

The aim of this demonstration is to show how to use PRIDE to develop an embedded real time system following a component-based software architecture and what the benefits of using the related process, tools and analysis are. Based on the example of an automatic car rental system, the demonstration will show the system design, analysis of system properties (worst case execution time and fault tolerant properties), implementation, deployment and the system in action.

A video describing ProCom component model, the PRIDE tool and some of the demonstration steps can be found at `http://www.youtube.com/watch?v=Y1dkUCuqzrA`

The FollowMe system example.

Its objective is to allow rental cars to be brought back to the rental station by following a specific leading car in an automated way. The automated car system implements 4 different modes:

1. **Detecting of the car to follow.** The car waits until there is a car in front with a predefined color.
2. **Following the car in front.** When a car is recognized as a car to follow, it follows the car in front forward and backward by trying to keep a constant distance between the two cars.
3. **Obstacle warning.** If a rear obstacle is detected and the car in front is too close, a warning is issued by blinking front lights and a warning sound.

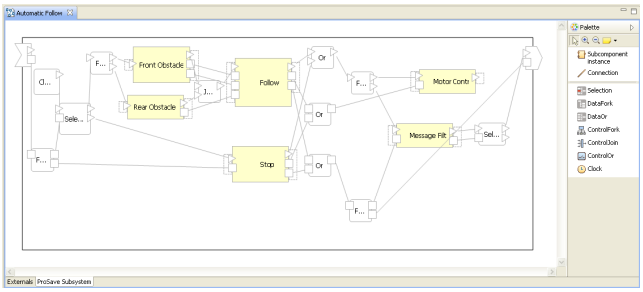

Figure 4: FollowMe System Partial Design.

Figure 5: Automatic Lego Car.

4. **Disable mode.** The car behaves as a manual car.

The automatic car contains the following physical elements:

- Front and rear obstacle distance sensors
- Front and rear lights
- Horn
- An engine which allow to change speed
- A front color sensor which is used to recognize if a car should be followed or not. This sensor works only on short distance and is not reliable.

We will demonstrate PRIDE using the FollowMe system as an example. This system will be generated for the Lego Mindstorm platform. The demonstration will end with demonstration of the FollowMe automatic car system following a red Lego car.

Demonstration Script.

1. Introducing the FollowMe application.
2. Partial design of FollowMe with ProCom is described. The system design is completed by importing components from the component repository and defining connections to the others.
3. The behavior of specific parts of the system is added using REMES language.
4. The parametric WCET analysis is used to check that detection of an obstacle in front takes no more than 40 ms.
5. The fault propagation analysis is used to compute if the automatic car is tolerant to unrecognized color.
6. The implementation is completed by adding some C code for a primitive component.
7. The synthesis generates glue code and compiles system as an executable file.
8. The system code is uploaded to the Lego Mindstorm car.
9. The system code is executed on the Lego Mindstorm car and it is demonstrated that it behaves as expected.

PRIDE research contributions

This section lists the major research contributions which have been integrated into PRIDE.

PRIDE

- Ivica Crnkovic, Séverine Sentilles, Thomas Leveque, Mario Zagar, Ana Petricic, Juraj Feljan, Luka Lednicki, and Josip Maras. PRIDE. In *DICES workshop @ SoftCOM 2010*, September 2010.

Software Architecture

ProCom Component Model

- Tomáš Bureš, Jan Carlson, Séverine Sentilles, and Aneta Vulgarakis. A component model family for vehicular embedded systems. In *The Third International Conference on Software Engineering Advances*. IEEE, October 2008.
- Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica 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.
- Aneta Vulgarakis, Jagadish Suryadevara, Jan Carlson, Cristina Seceleanu, and Paul Pettersson. Formal semantics of the ProCom real-time component model. In *35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, August 2009.

Attribute Framework

- Séverine Sentilles, Petr Stepan, Jan Carlson, and Ivica Crnkovic. Integration of extra-functional properties in component models. In *12th International Symposium on Component Based Software Engineering (CBSE 2009), LNCS 5582*. Springer Berlin, LNCS 5582, June 2009.
- Rikard Land, Jan Carlson, Stig Larsson, and Ivica Crnkovic. Project monitoring and control in model-driven and component-based development of embedded systems: The CARMA principle and preliminary results. In *5th International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress, July 2010.

Analysis

Worst Case Execution Time Analysis

- Thomas Leveque, Etienne Borde, Amine Marref, and Jan Carlson. Hierarchical composition of parametric wcet in a component based approach. In *14th IEEE International Symposium on Object/Component/ Service-oriented Real-time Distributed Computing (ISORC'11)*. IEEE, March 2011.
- Amine Marref. Compositional timing analysis. In *SAMOS X – International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. IEEE, June 2010.

REMES

- Aneta Vulgarakis, Séverine Sentilles, Jan Carlson, and Cristina Seceleanu. Integrating behavioral descriptions into a component model for embedded systems. In *36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 113–118. IEEE, September 2010.
- Dinko Ivanov, Marin Orlic, Cristina Seceleanu, and Aneta Vulgarakis. REMES tool-chain – A set of integrated tools for behavioral modeling and analysis of embedded systems. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010)*, September 2010.
- Cristina Seceleanu, Aneta Vulgarakis, and Paul Pettersson. REMES: A resource model for embedded systems. In *In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009)*. IEEE Computer Society, June 2009.

Synthesis

Code Generation

- Etienne Borde, and Jan Carlson. Automatic Synthesis and Integration of Gray-box Components for Critical Embedded Systems – Reuse vs. Optimizations. *Technical Report*, MDH-MRTC-252/2011-1-SE, February 2011.

Deployment modeling

- Jan Carlson, Juraj Feljan, Jukka Mäki-Turja, and Mikael Sjödin. Deployment modelling and synthesis in a component model for distributed embedded systems. In *36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, September 2010.