AutoFOCUS and the MoDe Tool

Jan Romberg, Jan Jürjens, Guido Wimmel
Systems & Software Engineering
TU München
85748 Garching, Germany
romberg@in.tum.de

Oscar Slotosch, Gabor Hahn
Validas AG
Lichtenbergstr. 8
85748 Garching, Germany
{slotosch, hahn}@validas.de

1 Introduction

Software engineering for distributed automotive applications is shifting from a subsystem-level perspcective, where the focus is on optimization of a single electronic control unit, towards a system-level view. However, optimization of distributed systems with respect to non-functional properties remains a challenging task.

The goal of the MoDe (*Mo*del Based *De*ployment) approach is to give early guidance for design decisions using architectural-level models of the system. In its current version, MoDe supports those architecture-level decisions that require a performance model of the overall system.

The MoDe approach is based on a formal design notation, AutoFOCUS, which is used for specifying *system models*, functional models enriched with abstractions for communication and scheduling. The MoDe tool offers automated support for compiling platform abstractions into the system model, so MoDe allows a highly flexible evaluation of different architectural choices.

2 The tool AutoFOCUS/Quest

The tool AutoFOCUS supports the integral model-based development of critical systems using a notation conceptually similar to a restricted subset of UML. The core of this platform is the editor AutoFOCUS with graphical description techniques that allow to model embedded systems under different view-points. AutoFOCUS/Quest offers simulation, code generation, consistency checking, (bounded) model checking, verification, abstractions, and a test environment. AutoFOCUS/Quest has been applied in a large number of case studies, in particular in the domain of safety-critical and security-critical systems. See [1] for more information.

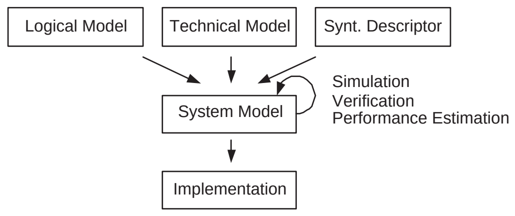

Figure 1. Design Flow

3 Design Flow

Figure 1 shows the design flow for the MoDe approach. The designer starts out with a functional view on the application, the *logical model*, a view on the technical architecture of the system, the *technical model*, and an additional *synthesis descriptor*. The MoDe tool reads the input representations and generates an AutoFOCUS *system model* used for simulation, verification, and performance analysis.

Logical model

The logical model captures the software architecture in terms of *software components*, component *interfaces*, and *communication dependencies* between components. The logical model abstracts from the actual deployment, i.e. communication protocols and scheduling are not part of the logical specification.

Figure 2 shows an example System Structure Diagram (SSD) for a logical model. Note that a component may be refined by a network of sub-components defined in another SSD (component `Controller` in the example.)

Leaf components in AutoFOCUS component hierarchies are defined by State Transition Diagrams (*STDs*), an FSM dialect extended by local variables and message send/receive statements. The MoDe tool also

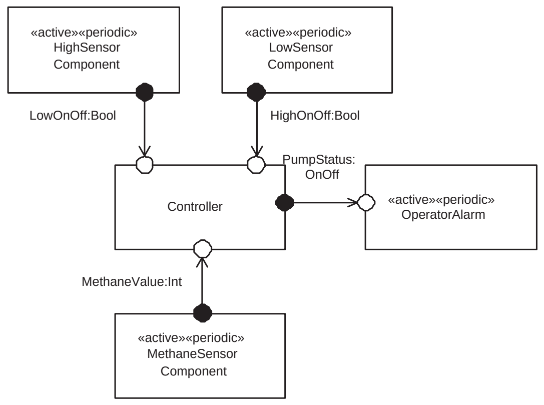

Figure 2. SSD for Logical Model

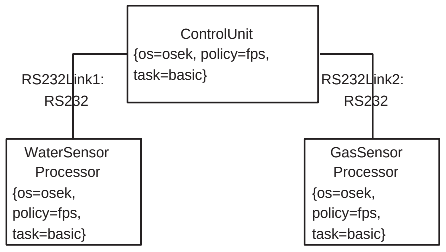

Figure 3. Technical model

supports an extended *timed STD* [3] notation for modeling execution times.

Leaf components in logical models are further differentiated along their mode of activation as *active* or *passive* components. Each active component corresponds to a lightweight task (thread) on the implementation level, while passive components correspond to code executed within the caller's thread.

Technical model

The characteristics of the platform are specified within the *technical model*. The technical model captures the structure of the underlying hardware such as interprocessor communication, operating system characteristics, and processor performance. The concepts of the technical model are supported by *component libraries* which define the abstractions required to generate the system model.

Technical models consist of networks of *nodes*, *links* between nodes, and *connectors* on nodes. Fig. 3 shows an example technical model with three nodes.

Synthesis descriptor

In addition to the logical and technical models, some additional mapping information has to be provided by the developer in the form of a *synthesis descriptor*. It is required to ensure an unambiguous translation to system models. The synthesis descriptor captures information that depends both on the logical and technical models; currently, there is a `mapping` part, and a `schedule` part.

System Model

The MoDe tool compiles the logical and technical models into an AutoFOCUS system model. The synthesis step uses the mapping from logical to technical components in the synthesis descriptor to synthesize the system model as a complete representation of the system behavior relevant to analysis. The MoDe tool uses predefined components for scheduling and communication defined in an application-specific library. All analysis is based on the system model; if several different deployments are evaluated, a corresponding number of analysis cycles are required.

4 MoDe Tool

The MoDe tool is an extension of the publicly available AutoFOCUS tool. The tool is implemented in about 12,000 lines of Java code, and supports graphical editing of logical and technical models through AutoFOCUS's standard editors. The automatically generated system model may be subjected to simulation or quantitative real-time analysis [2].

5 Conclusion

We have presented the tool AutoFOCUS/Quest and a tool-supported approach for architecture-level design and analysis of real-time systems. Logical and technical views of a system can be modeled separately and compiled into a system-level description. We have implemented a tool for modeling and compiling MoDe models.

References

[1] AutoFocus Project Homepage. URL: `http://autofocus.in.tum.de/index-e.html`.

[2] S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, 1994.

[3] J. Romberg, O. Slotosch, and G. Hahn. MoDe: A method for system-level architecture evaluation. In *Proceedings of the First Workshop on Formal Methods and Models for Codesign (MEMOCODE)*, 2003. to appear.