Chapter #

PLATFORM MODELING IN SAFETY-CRITICAL EMBEDDED SYSTEMS

Bernhard Huber and Roman Obermaisser
Institute of Computer Engineering, Vienna University of Technology, Austria

Abstract: This paper describes a model-based development process for safety-critical embedded real-time systems that are based on the DECOS integrated architecture. The DECOS architecture guides system engineers in the development of complex embedded real-time systems by providing a framework for integrating multiple application systems within a single distributed computer system. This integration is supported by a model-based development process which enables the reuse of application software on different instantiations of the DECOS platform, performing validation activities earlier in the development phase, and a reduced time-to-market in spite of increasing system functionality. For this purpose, model-based development in DECOS distinguishes between the capturing of the application functionality in a platform-independent model and the specification of the characteristics of the execution platform in the platform model. In this paper, we focus on the modeling of the execution platform and present a novel graphical model editor based on GME for specifying the DECOS execution platform. A platform meta-model expressed using UML and OCL constrains developers in such a way that the ensuing system becomes more dependable, maintainable and supports composability.

Key words: Model-based design; Integrated architectures; Embedded real-time systems

1. INTRODUCTION

Over the past two decades, the world of embedded systems has substantially changed. Mainly driven by competitive pressure and market forces of designers to introduce new products that set themselves apart from competitors, the functionality of embedded systems as well as the number of interacting components has steadily increased. For instance in the automotive industry, the replacement of many hydraulic control systems has risen the number of micro controllers per vehicle from an average of 20 to 40-60 between the year 2000 and 2003[1]. Thus, the key challenge faced by embedded system developers has become to manage the ever increasing complexity of such distributed real-time systems. The management of this complexity is aggravated by the inadequate abstraction level of today's programming languages. Many present-day embedded systems are *defect intolerant*, in the sense that even the smallest defects can cause major and expensive failures[2]. This observation is based on the insight that a semantic gap exists between the means of expression in programming languages and real-world problems. Therefore, it is thus suggested to raise the level of abstraction of program specification to a level that is closer to the problem domain[2].

Following this line of reasoning, the present paper describes a contribution towards a model-based development process for integrated systems. Integrated computer systems such as DECOS[3] contain a multitude of application subsystems (e.g., power-train, comfort, multimedia, safety in a car) which share a common distributed real-time system. The presented model-based development process starts with two models at a high level of abstraction, namely a platform-independent model that specifies the application services and a model of the execution platform. Using these two models, embedded system developers can specify the relevant properties at the application and platform-level in a form that is close to their problem domain. The models are used as an input for a tool-chain, which creates a platform-specific model in which the application services have been mapped to the available resources of the execution platform (e.g., computational resources of node computers, communication resources of networks).

The present paper contributes to an essential part of the model-based design process of DECOS—the modeling of the execution platform. It argues for the separation of resource modeling into two phases, which mainly facilitates re-use and hierarchical composition of platform models. Furthermore, the paper shows a prototypical implementation of a graphical editor that fits to the DECOS tool-chain. This tool is based on the Generic Modeling Environment (GME) and provides an intuitive and convenient front-end for modeling the execution platform. It expedites the modeling activities via generic templates for all constituting parts of the execution platform. Furthermore, by enforcing consistency checks and structural constraints, the majority of design faults are ruled out right from the beginning.

The paper is structured as follows. Section 2 gives a short overview on related work. The model-based development process is outlined in Section 3.

Section 4 describes the devised platform model editor based on GME, which is employed for modeling an exemplary DECOS platform. The results of this case study are described in Section 5. The paper concludes with a discussion in Section 6.

2. RELATED WORK

This section exemplarily describes important related work on model-based design for embedded systems and elaborates on the contributions of the proposed framework compared to those existing solutions.

2.1 MARTE

MARTE[4] (Modeling and Analysis of Real-Time and Embedded Systems) is a UML profile that offers a unified modeling language for real-time embedded systems. MARTE was developed to enable tool interoperability and facilitate training of system and software engineers through a common language for real-time embedded systems. UML does not provide concepts for fully capturing real-time systems, such as time, resource, scheduling. Therefore, MARTE introduced time and resource models, support for modeling non functional properties, and platform modeling. In analogy to the work presented in this paper, MARTE distinguishes between logical and physical views and is compatible to the Model Driven Architecture (MDA)[5]. MARTE also addresses the allocation of applications to platforms.

The major difference to the presented work is that MARTE does not constraint an implementation concerning a specific architecture. As stated by Rioux (p.17)[6], MARTE is simply a language and "*MARTE does not tell you how to design your real-time and embedded systems*".

In contrast, the framework presented in this paper constrains developers in such a way that the ensuing system becomes more dependable, maintainable, and supports composability. Technically, this is achieved using the UML meta-model in conjunction with Object Constraint Language (OCL) constrains. For example, constraints regarding the allocation of computational resources ensure that safety-critical and non safety-critical application subsystems are assigned dedicated hardware resources. This is a key for the cost effective realization of mixed criticality systems with fault isolation and support for modular certification. Likewise, constraints for the allocation of the communication resources facilitate the correct use of the network interfaces (e.g., limitations concerning topologies).

2.2 SysML

SysML (Systems Modeling Language)[7] is a graphical modeling language for systems engineers based on a subset of UML2.0 with extensions such as new diagrams (e.g., parametric diagram) and modified diagrams (e.g., activity diagram). SysML supports the specification, analysis, design and validation of systems. It has been motivated by the missing standard notation and semantic of UML.

The newly introduced *requirements diagram* supports the specification of relationships between requirements using stereotypes (e.g., satisfy, derive, verify). The *use case diagrams* elaborate the interactions between external users and the system. They are expressed either from the point of view of the users or from the point of view of the system. *Block definition diagrams* describe the structure of a system in a hierarchical, tree-like fashion. In order to describe behavior, *sequence diagrams*, *state machines*, and *activity diagrams* can be used. The *parametrics diagram* is a new diagram that explains relationships between parameters (e.g., dependencies between variables).

The differences of SysML to the solution presented in this paper are similar to the ones for MARTE. SysML provides a language and does not constraint an implementation concerning a specific architecture.

2.3 Architecture Analysis & Design Language

The Architecture Analysis & Design Language (AADL) is an approved industry standard that has been developed under the guidance of the Society for Automotive Engineers (SAE)[8]. Its core focus is modeling and model-based analysis of real-time embedded systems. Systems are modeled in terms of components and their interactions, for which AADL distinguishes two classes of components: software components and execution platform components. Software components describe the software structure including the sequence of execution in the final system using threads, processes, subprograms, and data. The hardware of embedded systems is expressed in terms of execution platform components such as processors (execution of threads), memories (storage of code and data), busses (access among components), and devices (interaction with the environment). For specifying interactions between components AADL provides ports (data, event, and event data port), which enable the directional exchange of data or events. Moreover, specialized connectors describing the access to a common shared resource such as a bus as well as for the interaction between subprograms are defined in the AADL standard.

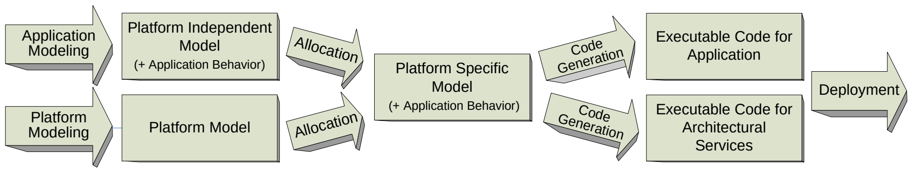


Figure 1. Development Methodology in DECOS

In contrast to other modeling languages, e.g. such as UML, AADL specifies semantics for the standardized types, components and their interactions. This way, different tools have a common interpretation of AADL models, which eases the comparability of analysis results of different tools. AADL supports model interchange and tool chaining based on a standard XML/XMI definition.

However, as explained for the MARTE UML profile, AADL provides the means for modeling and analysis of embedded systems but does not guide the system engineer in way to design embedded systems. To our knowledge, AADL provides no mechanism to define meta-models for AADL that define what a valid model of an execution platform for a particular type of systems – such as DECOS – should look like.

3. MODEL-BASED DESIGN IN DECOS

The DECOS integrated architecture[3] offers a framework for the development of distributed embedded real-time systems integrating multiple application subsystems with different levels of criticality and different requirements concerning the underlying platform. The DECOS development methodology adapts the distinction between *platform-independent* and *platform-specific* viewpoints as introduced by the Model Driven Architecture (MDA)[5]. For the description of the structure of distributed computer systems, the MDA introduces *models* with various levels of detail and focus such as the Platform Independent Model (PIM) and the Platform Specific Model (PSM) – and defines their role in the design of a system.

3.1 Adapted MDA for DECOS

By adhering to the distinction between PIM and PSM as introduced in the MDA, we separate modeling of the application from modeling of the execution platform. As depicted in Fig. 1 we start with the modeling of the PIM and the construction of the platform model.

The *DECOS PIM*[9] decomposes the overall system (e.g., the electronics of an entire car) into a set of nearly-independent application subsystems. Each application subsystem provides an application services that is meaningful in the given application context (e.g. in the context of an in-vehicle electronic system such subsystems are the comfort, power-train, or infotainment application subsystem). Moreover, application subsystems are further subdivided into a set of *jobs*, which interact exclusively by the exchange of messages via *virtual networks*[10]. Additionally, the DECOS PIM enables the specification of dependability and performance characteristics for each application subsystem and the establishment of links to specifications of the behavior of the individual jobs, which are expressed in DECOS using SCADE[11].

The *platform model* specifies the physical building blocks of the execution platform, the node computers, with the available communication resources (e.g. network interfaces) and computational resources (e.g. memory, processors). The formal foundation for modeling the execution platform is established by the platform meta-model[12], which is outlined in the following subsection.

Using the specifications of the PIM and the execution platform, a tool-supported transformation towards the *DECOS PSM* is initiated, which performs the mapping of the logic system structure described in the PIM to the physical system structure captured in the platform model. The PSM of an application subsystem is a refinement of the PIM where the jobs have been assigned to node computers and communication resources for virtual networks are reserved on the physical network. Besides resource constraints of node computers (e.g. the memory requirements of a job must not exceed the available memory on a particular node computer) and the physical network (e.g. the available bandwidth of the physical network determines the number of simultaneous virtual networks), the transformation of the PIM to the PSM is constrained by dependability requirements. For instance, in order to increase the system reliability by means of Triple Modular Redundancy (TMR) replicated jobs have to be assigned to independent fault-containment regions (e.g. on different node computers).

In the DECOS architecture the PSM forms the input for code generation tools, which automatically produce *executable code* for the application and the platform. For this purpose, a DECOS PIM to SCADE gateway[13] has been developed which directly imports the building blocks of the PIM into SCADE, which enables the use of the qualified code generator KCG of SCADE for generating executable code. Also, automatic code generators for the architectural services (e.g., middleware for virtual networks[10] and virtual gateways[14]) have been devised. The generated source code together with the PSM forms the input to a deployment step in which the final executables for a specific instance of the DECOS execution platform are created.

3.2 Execution Platform Modeling

Modeling the characteristics of the execution platform is a time-intensive and error-prone engineering task. It is therefore our objective to simplify and reduce the effort for this modeling task by developing the *Platform Model Editor (PME)*. The formal foundation of this tool is established by the platform meta-model, which primarily aims at facilitating re-use and hierarchical composition of platform models[12]. For this purpose, the modeling process is separated into two phases: the resource capturing phase and the platform composition phase.

3.2.1 Resource Capturing Phase

This phase addresses the specification of reusable hardware entities of an instantiation of the DECOS architecture, so-called resource primitives. Resource primitives form the lowest level of the platform description. They are the atomic hardware units of an execution platform that are identified in the modeling process. The platform meta-model defines a set of common resource primitive types, which can be used across different instantiations of the DECOS architecture. These primitive types are *Processor*, *Memory*, *Communication Interface*, *Communication Controller*, and *Connector*. For each of them, a set of *hardware properties* is predefined (e.g. clock frequency for a processor resource primitive).

Moreover, for each resource primitive further hardware properties can be specified in concrete instantiations of an execution platform. In order to support the modeling of evolving types of resource primitives, the platform meta-model provides generic model entities, whose semantics and hardware properties can be freely defined. For establishing a common interpretation of newly defined resource primitives, the concept of *technical dictionaries*[15] is applied. A technical dictionary is a reference base, containing all relevant products/parts of a particular application field, where for each product/part a detailed description and a list of its properties is given. By assigning to each generic resource primitive a unique identifier of a technical dictionary, a common interpretation of the resource primitive among different developers of the model is ensured.

3.2.2 Platform Composition Phase

The second phase is concerned with the composition of the entire platform model out of the previously modeled resource primitives. The output of the composition is the description of a DECOS cluster with its internal structure and the network infrastructure (e.g., the time-triggered core network between nodes, external networks like field buses, etc). A *cluster* is a distributed computer system that consists of a set of node computers interconnected by a network. A *node computer* is a self-contained computational element with its own hardware (processor, memory, communication interface, and interface to the controlled object) and software (application programs, operating system), which interacts with its environment by exchanging messages. According to the DECOS model[3], a node computer is vertically structured into two subsystem. The *safety-critical subsystem* is an encapsulated execution environment for ultra-dependable applications, while the *non safety-critical subsystem* offers an environment for those applications having less stringent dependability requirements. Within the subsystems, *connector units* control the access of jobs to the shared time-triggered core network. A third connector unit, denoted as *basic connector unit* performs the primary allocation of physical network resources, as required for the separation of safety-critical and non safety-critical subsystems of a node.

The platform composition phase is structured into three different steps:

a) **Hardware Element Composition.** As a first step in the composition phase, individual resource primitives are composed to a larger physical hardware unit (e.g., a single board computer) that is capable of realizing (parts of) a DECOS node. In the platform meta-model, these hardware units are denoted as *hardware elements*[12]. A collection of these hardware elements form a *resource library*, which provides the building blocks for the subsequent steps in the platform composition phase.

b) **Node Composition.** The platform meta-model constrains the composition of possibly heterogeneous hardware elements to DECOS nodes. In this *node composition step*, the available resources for the safety-critical and the non safety-critical subsystems (including the resources for the execution of jobs as well as for the execution of the architectural services realizing the connector units) are specified.

c) **Cluster Composition.** As a final step in the platform composition phase – the *cluster composition* – the interconnection of nodes forming a cluster, which represents an instance of the DECOS architecture, is specified.

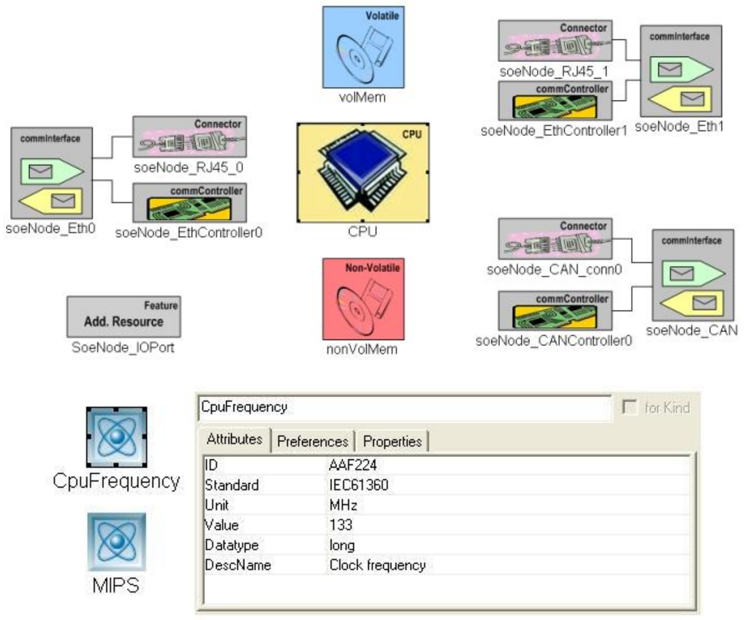
4. PLATFORM MODEL EDITOR

The main motivation for a tool-based modeling environment is to keep the modeling process focused on its essential challenge – the description of the available hardware resources. The user of the tool, who usually has knowledge of the setup and internals of the execution platform, should be able to describe the essential characteristics of the platform without being forced to extensively study meta-models and tools; thus, speeding up the modeling process.

The implementation of the platform model editor (PME) is based on GME, which is a configurable, graphical framework for creating modeling environments[16]. GME has been configured with a formal modeling paradigm, which contains all the syntactic, semantic, and presentation information regarding the targeted domain. The DECOS-specific modeling paradigm is derived from the platform meta-model. According to the previously described phases of the platform modeling process, the modeling paradigm comprises three different viewpoints:

a) **Hardware Element Viewpoint.** The hardware element viewpoint is related to the resource capturing phase and determines the modeling entities for specifying the physical building blocks of the platform. It contains separate model entities for the entire set of resource primitives as well as for hardware elements. Further on, valid compositions of those model entities are defined by using containment relationships between the hardware element and the resource primitives.
b) **Node Viewpoint.** The node viewpoint describes the structuring of a DECOS node in safety-critical and non safety-critical subsystems. It is expressed by a containment relationship between the model elements *Node* and *ConnectorUnit* (which represents the execution environment of the DECOS architectural services) and *ApplicationComputer* (which represent the execution environment of jobs), respectively, which are associated to H*ardwareElement* model elements.
c) **Cluster Viewpoint.** The purpose of this viewpoint is to specify the required entities for modeling the cluster setup including the association of nodes to the time-triggered core network as well as to additional physical networks (e.g., field buses).

In addition to the specification of the model entities and the associations between them, each viewpoint of the modeling paradigm comprises a set of Object Constraint Language (OCL) constraints. OCL[17] is a formal language for describing constraints on models. OCL can be used to specify invariants that must hold for the modeled system during the whole lifetime or in particular system states. Consider for instance the following simple constraint expressed in natural language: *A node consists of a safety-critical and/or a non safety-critical subsystem, but at least of one of them.* Just specifying an association with a multiplicity constraint (e.g., “0..1”) from node to both types of subsystems does not correctly represent this constraint, because it is still possible to model a node without any subsystem at all. This lack of information is easily added by an OCL constraint that specifies an invariant stating that the total number of subsystems must be greater than



ID	AAF224
Standard	IEC61360
Unit	MHz
Value	133
Datatype	long
DescName	Clock frequency

Figure 2. Screenshot of Models of the Internal Setup of Soekris net4521 Boards.

zero and less than or equal to two. Further constraints are deployed to restrict entities depending on their actual role in the model. For instance, a connector unit instantiated as basic connector unit, requires a mandatory additional interface to the core network.

5. DECOS EXECUTION PLATFORM

We have applied the PME to describe a real instance of the integrated DECOS architecture – a prototypical execution platform[18] that has been developed in the course of the DECOS project. The prototype consists of a cluster of five nodes using TTP/C[19] as time-triggered core communication network. Each node hosts a safety-critical and a non safety-critical subsystem with multiple jobs of different application subsystems. Each node of the cluster comprises three distinct single board computers for realizing the basic connector unit and the safety-critical and non safety-critical subsystems.

5.1 Hardware Element Viewpoint

The hardware element viewpoint is used to specify the characteristics of the hardware elements, i.e. the building blocks for the composition of the execution platform. In our prototype exist two such hardware elements: the

TTTech monitoring node[1] and the Soekris Engineering net4521 board[2]. The model of the Soekris board is depicted in Fig. 2. It specifies only those characteristics of the single board computer that are important for the DECOS development process. Thus, it describes the processor, volatile and non volatile memory for program and data storage, as well as, communication interfaces and I/O ports. All of these entities represent resource primitives of the platform meta-model.

Fig. 2 also exemplary shows the specification of detailed information on resource primitives. Two detailed properties of the ElanSC520 CPU are exemplified here: the clock frequency and the instructions per second. It further shows the usage of technical dictionaries. In the IEC 61360 standard[3] the property *AAF224* is defined as the clock frequency for micro processors enabling an unambiguous interpretation of this property for the ElanSC520 CPU.

5.2 Node Level Viewpoint

The node level viewpoint describes the internal configuration of a DECOS node. For the nodes of the prototype cluster we employ distinct hardware elements for the basic connector unit (cf. *MonNode0* in the left model of Fig. 3), the safety-critical subsystem, and the non safety-critical subsystem (i.e., one node computer for each connector unit and respective applications; cf. *SoeNode0_0* and *SoeNode0_1* in Fig. 3).

Those connector units are interconnected by a time-triggered Ethernet network (*ConnectorNetwork0*), which is a standard Ethernet connection using Time Division Multiple Access (TDMA) for the arbitration of the communication medium. The physical interfaces to this network are modeled by *bcuCN0*, *scuCN0*, and *xcuSC0* respectively. The basic connector unit possesses an additional interface – *CoreInterface* – with which the time-triggered core network is accessed. For the non safety-critical subsystem, an additional communication interface to a field bus network is specified.

Furthermore, it is specified whether the node provides separate encapsulated execution environments for safety-critical and for non safety-critical applications and by which hardware elements they are realized. The model on the left in Fig. 3 depicts a configuration in which the architectural services and the applications of one subsystem share a single hardware element.

[1] http://www.tttech.com
[2] http://www.soekris.com
[3] Standard data element types with associated classification scheme for electric components – available at http://dom2.iec.ch/iec61360/iec/61360.nsf/

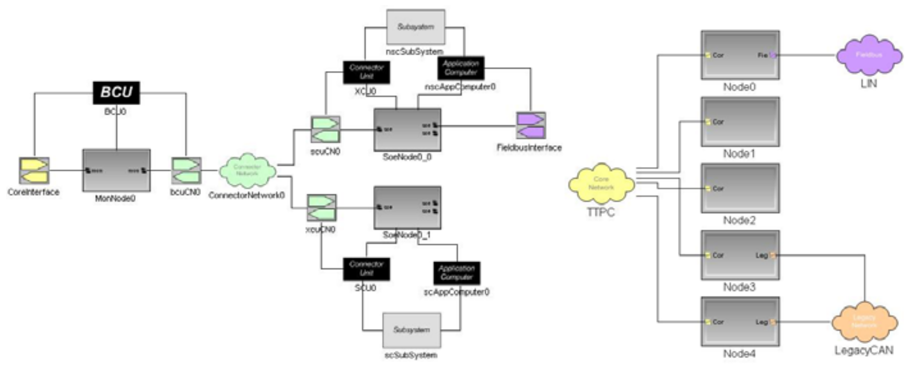


Figure 3. Screenshots of Node Level Viewpoint (left) and Cluster Level Viewpoint (right)

5.3 Cluster Level Viewpoint

It is the purpose of this viewpoint to describe the nodes connected to the core network as well as properties of the core network and additional networks like field buses. The specification includes performance properties and temporal characteristics (e.g., maximum bandwidth, guaranteed latency) as well as physical characteristics like redundancy, topology, or the used physical layer. In Fig. 3 the cluster level viewpoint of our prototypical DECOS execution platform is depicted. It consists of five nodes, interconnected by a TTP/C network in star topology. In addition, two nodes are locally connected to a CAN[20] network while a third one accesses a LIN[21] field bus.

Across all viewpoints the PME provides assistance to the user in modeling the execution platform, since the tool ensures that only models are created that comply with the DECOS modeling paradigm. Therefore, the tool prevents the creation of associations which are not specified in the modeling paradigm (e.g., in the node level viewpoint a hardware element cannot be directly connected to the network, since an interface in between is mandatory) and monitors the multiplicities of associations (e.g. it is not possible to connect communication interface resource primitives to more than one network). Finally, violations of the meta-model such as missing mandatory parts of the model (e.g., at least one subsystem has to be specified in the node) are reported to the user.

6. CONCLUSION

The development methodology of DECOS offers a model-based design process for distributed embedded real-time systems. Due to comprehensive tool support, embedded system engineers can rapidly capture the essential properties of applications and execution platforms and can perform successive model transformations down to the physical target system.

An essential part of the tool chain is the resource modeling editor. This editor enables a precise formal specification of the available communication and computational resources, while providing a user-friendly and intuitive interface to the application designer. Using a set of entities well-known in the domain of embedded systems (e.g., networks, processors, connector units, field buses), the modeling process is close to the problem domain of typical embedded system engineers. This reduces the mental effort for both, the person who creates the model and the person who is in charge of its interpretation.

Additionally, the resource modeling editor performs a pre-selection of the visible model entities to those parts that are appropriate in a given context. For instance the detailed representation of resource primitives is only visible when creating new elements of a resource library, but masked in the node or cluster viewpoint.

A further strength of the proposed modeling editor is the reduction of design faults. It is automatically checked, whether a particular relationship between two entities is permitted or not. These decisions are based on the specified DECOS modeling paradigm. Thus, violations of the underlying meta-model are directly reported to the user. Furthermore, the modeling paradigm of the execution platform model comprises various constraints specified using OCL, which further restrict entities, attributes and relationships of the modeling environment.

Finally, the presented framework supports the reuse of resource specifications through the import and export of (parts of) the platform model. Therefore, re-use of resource descriptions in different models can be simply performed by *drag and drop* operations. Moreover, the PME checks if the insertion of the model element at the specified position complies with the meta-model.

ACKNOWLEDGMENTS

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

REFERENCES

1. C.J. Murray. Auto group seeks universal software. *EE Times*, 2003.
2. B. Selic. Model-driven development: its essence and opportunities. In *Proc. of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, page 7pp, April 2006.
3. R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal*, 3:83.95, March 2006.
4. OMG. A UML Profile for MARTE, Beta 1. OMG adopted specification. August 2007.
5. OMG. Model Driven Architecture (MDA). Technical Report document number ormsc/2001-07-01, Object Management Group, July 2001.
6. L. Rioux. MARTE: A new OMG standard for Modeling and Analysis of Real-Time Embedded Systems. Thales Research & Technology, France. September 2007.
7. OMG. Systems Modeling Language (OMG SysML), V1.0 Specification. September 2007.
8. SAE. Architecture Analysis & Design Language (AADL). AS5506. November 2004.
9. DECOS. Dependable Embedded Components and Systems. Project deliverable D1.1.1. Report about decision on meta-model and tools for PIM specification. December 2004.
10. R. Obermaisser and B. Huber. Model-based design of the communication system in an integrated architecture. In *Proc. of the 18th Intern. Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*, pages 96-107, November 2006.
11. Esterel Technologies. SCADE Suite Technical and User Manuals, Version 5.0.1, 2005.
12. B. Huber, R. Obermaisser, and P. Peti. MDA-Based Development in the DECOS Integrated Architecture - Modeling the Hardware Platform. *Proc.of the 9th IEEE International Symposium on Object and component-oriented Real-time distributed Computing (ISORC'06)*, April 2006.
13. W. Herzner, B. Huber, A. Balogh, and P. Csertan. The DECOS Tool-Chain: Model-Based Development of Distributed Embedded Safety-Critical Real-time Systems. *DECOS/ERCIM Workshop on Dependable Embedded Systems*, September 2006.
14. DECOS. Dependable Embedded Components and Systems. Project deliverable D2.2.3. Virtual communication links and gateways – Implementation of design tools and middleware services. December 2005.
15. M. Sundaram and S.S.Y. Shim. Infrastructure for B2B exchanges with RosettaNet. In *Third Int. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS 2001*, pages 110.119, 2001.
16. A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garret, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Proc. of Workshop on Intelligent Signal Processing*, May 2001.
17. OMG. UML 2.0 OCL specification, OMG final adopted specification. Technical Report OMG Document No. ptc/03-10-14, Object Management Group, October 2003.
18. B. Huber, P. Peti, R. Obermaisser, and C. El Salloum. Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems*, May 2005.
19. H. Kopetz and G. Grünsteidl. TTP – A protocol for fault-tolerant real-time systems. *Computer*, 27(1):14.23, January 1994.
20. Robert Bosch Gmbh, Stuttgart, Germany. CAN Specification, Version 2.0, 1991.
21. LIN Consortium. LIN Specification Package Revision 2.0, September 2003.