Model-Based Development of Integrated Computer Systems: Modeling the Execution Platform

Bernhard Huber[1] and Roman Obermaisser[1]

[1]Institute of Computer Engineering,
Vienna University of Technology, Vienna, Austria
{huberb, romano}@vmars.tuwien.ac.at

Abstract — *The DECOS architecture provides a framework for integrating multiple application systems within a single distributed computer system. Since the DECOS architecture aims at applications in the automotive, avionic, and industrial control domain, including applications up to the highest criticality level, the design and development process of DECOS-based integrated computer systems is of utmost importance. Within the DECOS project a model-based development process is devised which aims at enabling a reduced time-to-market in spite of increasing the system's functionality, the reuse of application software on different instantiations of the DECOS platform, and performing validation activities earlier in the development phase of integrated computer systems. In this paper we outline the overall model-based development process of integrated computer systems based on the DECOS architecture with a strong focus on the modeling of the DECOS execution platform. Additionally, we present a novel graphical model editor based on GME for capturing the execution platform in the model-based development process.*

1 Introduction

The key challenge faced by embedded system developers has become the ever increasing complexity of distributed real-time systems. Driven by competitive pressure and market forces, complexity is primarily resulting from growing application functionality and increasing numbers of interacting components. For example, in the automotive industry electronic systems have been introduced in vehicles in the 90s in order to improve the performance of mechanical systems (e.g., Antilock Braking System) [1]. According to [2], the number of micro controllers per vehicle has risen from an average of 20 to 40–60 between 2000 and 2003.

The management of this increasing 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* [3, p. 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 in [3] to raise the level of abstraction of program specification to a level that is closer to the problem domain.

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 [4] or AUTOSAR [5], contain a multitude of application subsystems (e.g., powertrain, comfort, multimedia, safety in a car) which share a common distributed real-time system. Evidently, these integrated computer systems belong to those of today's embedded systems that exhibit the highest inherent application complexity.

The presented model-based development process is part of the integrated system architecture that is developed within the Dependable Embedded Components and Systems (DECOS) project within the EU Framework Programme 6. The 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 [6], 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).

While the platform-independent model has been discussed in [7], this work focuses on the execution platform. We introduce a data-capturing tool based on the Generic Modeling Environment (GME), which provides an intuitive and convenient front-end for creating the model of the execution platform. The data-capturing tool 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 provides more information about the model-based development process in the DECOS architecture. The platform model, which formally captures the communication and computational resources of a particular instance of the DECOS architecture, is the focus of Section 3. Section 4 describes the data-capturing tool that has been realized with GME. The platform model and the data-capturing tool are exemplified in Section 5. The paper concludes with a discussion in Section 6.

2 Model-Based Design in DECOS

The DECOS integrated architecture [4] 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. As a baseline for the development of applications, the DECOS integrated architecture offers to system designers generic architectural services (e.g., virtual networks [8], gateways [9], diagnostic services [10]).

The development methodology of DECOS adapts the distinction between *platform-independent* and *platform-specific* viewpoints as introduced by the Model Driven Architecture (MDA) [11]. For the description of the structure of distributed computer systems, the MDA introduces *models* with various levels of detail and focus. A model is a formal specification of a system and provides an abstraction, i.e., the model includes certain classes of information while suppressing other ones. The selection of which classes of

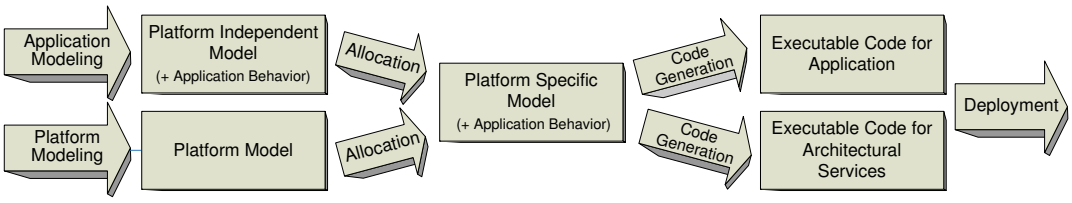


Figure 1: Development Methodology in DECOS

information to include or suppress depends on the purpose and the focus of the model. A particular selection of such information classes is denoted as a *viewpoint*. Widely used viewpoints in the design of distributed computer systems are platform-independent and platform-specific viewpoints. MDA is a standard that proposes such viewpoints – denoted as Platform Independent Model (PIM) and Platform Specific Model (PSM) – and defines their role in the design of a system.

By adhering to this distinction between PIM and PSM, we separate modeling of the application from the modeling of the execution platform. The development as depicted in Figure 1 starts with the modeling of the PIM by structuring the overall application into smaller logic elements denoted as application subsystems and jobs, but abstracts yet from the underlying execution platform. In parallel to the construction of the PIM, a resource specification model can be constructed, which captures the execution platform by describing the node computers with their computational resources (e.g., memory, processor performance) and local interfaces (e.g., specific sensors and actuators), as well as the network interconnecting the node computers.

After having formally defined the PIM and the execution platform, a tool-supported transformation towards the PSM follows. This transformation includes the mapping of the logic elements of the PIM to the physical elements of the execution platform.

In addition to devising the logical structure of the overall application in the PIM, the behavior has to be formally captured. Although it is possible to directly capture the behavior of the application by using a specific programming language, it is recommended to elevate the level of abstraction of the behavior specification. This can be achieved for instance by using formalisms provided by Matlab–Simulink[1] or SCADE [12]. A code generation tool transforms this behavior specification to executable code.

Finally, 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.

Platform Independent Model. The PIM decomposes the overall system (e.g., electronics of the complete car) into a set of nearly-independent application subsystems (e.g., see Figure 2). Each application subsystem provides an application services that is meaningful in the application context. In the automotive industry, candidates for application subsystems would be the different domains (e.g., comfort, powertrain, infotainment, safety) of the in-vehicle electronic system.

The PIM of an application subsystem consists of a set of *jobs*, which exchange messages via so-called *virtual networks* [8]. The PIM provides for each virtual network an

[1] `http://www.mathworks.com`

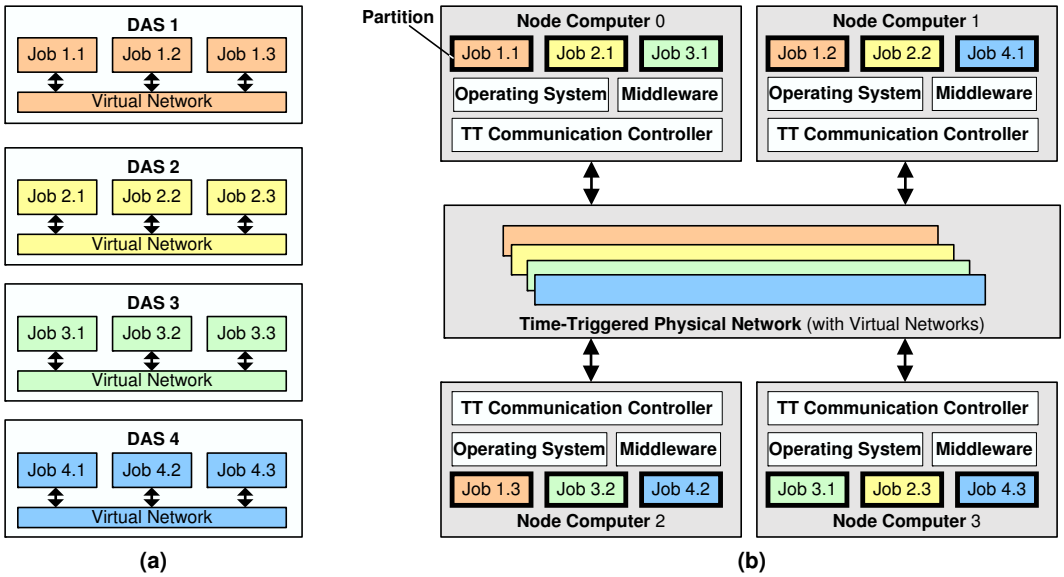


Figure 2: Structuring in Platform Independent Model (a) and Platform Specific Model (b)

operational specification that denotes the syntax of exchanged messages, the timing of the messages, and error conditions (e.g., input and output assertions). Furthermore, the PIM contains a formal specification of the dependability and performance characteristics of the entire application subsystem. A meta-model for expressing this information in UML has been defined in [7]. In addition, a formal definition of the interaction relationships and the behavior of jobs is associated with the PIM. This definition uses an appropriate formalism, such as SCADE [12].

Platform Model. The *platform model* specifies the physical building blocks of the execution platform with the available communication and computational resources. These building blocks include clusters, node computers, and (physical) networks. 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. A node contains one or more partitions. A *partition* is an encapsulated execution space within a node computer with a priori assigned computational (e.g., processor, memory, I/O) and network resources (e.g., network bandwidth) that can host a job. Partitions are the target of job allocation and each job is always assigned in its entirety onto a partition, i.e., a job is never fragmented onto multiple partitions.

Platform Specific Model. The PSM is a refinement of the PIM, which extends the PIM with information concerning the mapping of jobs to a specific execution platform as specified by the platform model (see right hand side in Figure 2). The PSM of an application subsystem is a model where the jobs have been assigned to partitions of nodes and virtual networks required by the individual application subsystems are reserved on the

time-triggered physical network. The transformation of the PIM to the PSM is constrained by the following requirements:

- **Dependability requirements.** Replicated jobs must be assigned to partitions of independent fault-containment regions.
- **Resource constraints of nodes.** The required computational resources of the jobs allocated to a node computer must not exceed the available resources of the partitions located in this node computer.
- **Resource constraints of the physical network.** The resources of the physical network constrain the number of virtual networks that can be supported simultaneously and the temporal performance of these virtual networks.

Executable Code. In the DECOS architecture, code generation tools use the PSM as an input and automatically produce executable code for the application and the platform. For example, in order to enable the use of SCADE for modeling the behavior of jobs, a DECOS PIM to SCADE gateway has been developed which directly imports the building blocks of the PIM into SCADE [6]. This way, a code generator like SCADE's qualified code generator KCG can be applied for generating the executable code for the application.

In analogy, code generators have been developed for producing executable code for the architectural services to be deployed on the DECOS computers (e.g., middleware for virtual networks [8] and virtual gateways [13]).

3 Execution Platform Modeling

It is the purpose of the platform model to capture the relevant characteristics of the platform for allocating jobs and virtual networks to the available resources of the platform. We first give an overview of the meta-model of the platform model as defined in [14] and then describe the XML-based interface to the DECOS tool-chain [6].

3.1 Platform Meta-Model

Specifying the characteristics of the execution platform like computational resources or communication resources is a time-intensive and error-prone engineering task. It is therefore an objective of the DECOS development methodology to simplify and reduce the effort for this modeling task by the development of a *Platform Model Editor (PME)*. The formal foundation of this editor is established by the platform meta-model, which aims at facilitating re-use of and hierarchical composition of (parts of) platform models [14]. For this purpose, the modeling process is separated into two phases: the resource capturing phase and the platform composition phase.

3.1.1 Resource Capturing Phase

The specification of reusable hardware entities of an instantiation of the DECOS architecture, so-called resource primitives, is addressed in the resource capturing phase. Resource primitives form the lowest level of the platform description, since they are the atomic hardware units of an execution platform which 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 are *Processor*, *Memory*, *Communication Interface*, *Communication Controller*, and *Connector*. For each of those resource primitives, a set of *hardware properties* is predefined (like clock frequency for a processor resource primitive).

However, for each resource primitive further hardware properties can be defined in a concrete model for an execution platform. Furthermore, 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.

3.1.2 Platform Composition Phase

The second phase is concerned with the composition of an entire platform model out of the previously modeled resource primitives. That is, the description of the internal structure of a DECOS node and the network infrastructure (e.g., the time-triggered core network between nodes, external networks like fieldbuses etc). According to the DECOS model of a node computer [4], a node computer is vertically structured into two subsystems. 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 to three different steps:

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, this hardware units are denoted as *hardware elements* [14]. A collection of these hardware elements form a *resource library*, which provides the building blocks for the subsequent steps in the platform composition phase.

Node Composition: The platform meta-model, which is derived from the DECOS model of node computers, 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.

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

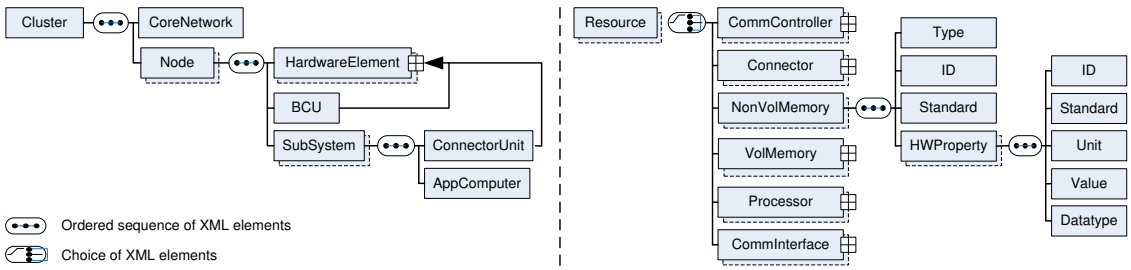


Figure 3: XML Schema Specifying PME Interface

3.2 Interface to DECOS Tool-Chain

In the course of the DECOS project a tool-chain has been defined and is currently under development that supports the model-based development of mixed criticality systems based on the DECOS architecture. The PME for modeling the execution platform is part of this tool-chain. In the following a short outline of the DECOS tool-chain is given and the interface of the PME to the *VIATRA DECOS Model Store* – the core of the tool-chain is described.

3.2.1 Outline of DECOS Tool-Chain

The DECOS tool-chain covers all of the required aspects for following a model-based development methodology for the design and development of integrated computer systems based on the DECOS architecture. The core of the tool-chain forms the *VIATRA DECOS Model Store* [6], which is concerned with the interconnection of the individual tools which are applied in the different phases of the development process. The Model Store is based on VIATRA (VIsual Automated model TRAnsformations) [16], which is a framework for transformation-based verification and validation of systems design using the Unified Modeling Language. In addition, the DECOS tool-chain encompasses a variety of tools, each specialized for a particular subtask of the development process. For instance, for modeling the PIM and the allocation of jobs to the available resources the VIATRA framework is used, the modeling of the execution platform is realized by the GME-based PME (see Section 4), and behavior modeling is performed with SCADE.

3.2.2 XML Interface

The exchange of data between the PME and the VIATRA model store is performed by the use of XML. Figure 3 depicts a graphical representation of the hierarchical structure of the XML schema specifying the PME interface (each rectangular object represents an XML element). There are several reasons in favor of using XML for the data exchange:

Interface Specification using an XML schema: The XML schema definition language can be seen as the interface definition language between the platform model editor and the model store. An XML schema specifies the structure of well-defined and valid XML documents. This way, before importing a platform model into the model store, the validity of the model can be easily verified by the use of (freely) available XML validation tools.

Hierarchical Data Representation: One of the inherent properties of XML documents is that the contained information is represented in a highly structured and hierarchical manner by using XML tags. Due to the hierarchical setup of DECOS nodes, this is a natural representation for DECOS nodes, thus it eases the specification of the interface.

Open and Standardized Specification: Using an XML schema to specify the interface for data exchange creates an open and standardized specification that can be fulfilled by a variety of tools. Thus, it facilitates the substitution and evolvement of the tools on both sides of the interface.

4 Platform Model Editor

The main motivation for creating a graphical modeling environment is to keep the modeling process of the execution platform 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, fastening the modeling process.

The implementation of the PME is based on GME, which is a configurable, graphical framework for creating modeling environments [17]. For this purpose, GME has to be configured by a modeling paradigm, which formally specifies the modeling language, i.e., describes the entities, attributes, relationships, and constraints of the resulting modeling environment. The modeling paradigm defines the family of models that can be created using the resultant modeling environment and thus has to contain all the syntactic, semantic, and presentation information regarding the targeted domain [18].

4.1 The DECOS Platform Modeling Paradigm

In GME the PME is specified by a DECOS-specific modeling paradigm, which is derived from the platform meta-model. According to the phases of the platform modeling process, the modeling paradigm comprises three different viewpoints: the *hardware element viewpoint* which is related to the resource capturing phase and the composition of hardware elements as well as the *node viewpoint*, and the *cluster viewpoint*, which are concerned with the latter two steps in the platform composition phase as explained in Section 3.1.2.

Hardware Element Viewpoint The hardware element viewpoint specifies the modeling entities, which are concerned with the specification of the physical building blocks of a DECOS execution platform. It therefore specifies separate model entities for the entire set of resource primitives (cf. Section 3.1.1) as well as for their composition, denoted as hardware element. Further on, containment relationships between the hardware element and the resource primitives, as well as between resource primitives are defined, which determine the valid interconnections of model entities in the PME.

Node Viewpoint The safety-critical and the non safety-critical subsystem of a DECOS node are represented in the node viewpoint by a containment relationship between the model elements *Node* and *ConnectorUnit* (which represents the execution environment of the DECOS architectural services) and *ApplicationComputer* (which

```
[1]  Initialize_XML_file();                  // create and initialize the XML file
[2]  cluster = Get_Root_Cluster_Object();    // get reference to root cluster object
[3]  coreNet = cluster.GetCoreNetwork();     // get reference to core network object
[4]  WriteAttributesToXML(cluster, coreNet); // write attributes to XML file.
[5]  forall nodes in cluster do {
[6]    bcu = node.GetBCU();                  // get reference to BCU object
[7]    subsys = node.GetSubSystem();         // get reference to subsystem object
[8]    WriteAttributesToXML(node, bcu, subsys);
[9]    forall hwElements in node do {
[10]     WriteAttributesToXML(hwElement);
[11]     forall resouces in hwElement do WriteAttributesToXML(resource);
[12]     forall features in hwElement do WriteAttributesToXML(feature);
[13]   }
[14] }
[15] Close_XML_file();                       // finalize and close XML file.
```

Figure 4: Pseudo-Code of the PME Model Interpreter

represent the execution environment of jobs), respectively. For assigning in the PME hardware elements to those execution environments, respective association relationships are specified in the node viewpoint.

Cluster Viewpoint The purpose of the cluster viewpoint is to specify the required entities for modeling the cluster setup including the association relationships 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 [19] is a formal language for describing expressions on models. OCL can be used to specify invariants that must hold for the modeled system during the whole lifetime or in particular 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 zero and less than or equal to two. Further constraints are deployed to specialize model entities depending on their actual role in the model. For instance a connector unit used as basic connector unit in the model, requires a mandatory additional interface to the core network.

4.2 Model Interpreter for XML Transformation

The PME provides a one-click mechanism for exporting the platform model into an XML-file according to the XML schema as introduced in Section 3.2. This transformation is realized via a so-called model interpreter. In GME all model entities defined by the modeling paradigm can be accessed and the therein contained information can be further processed by model interpreters.

The MetaGME modeling environment itself, which is the model editor for creating GME models, provides a model interpreter that automatically generates a C++ programming framework. This C++ framework comprises a class hierarchy reflecting exactly the

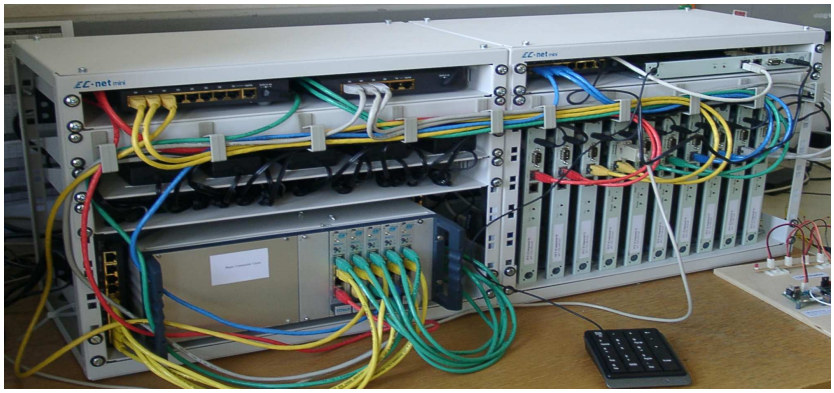


Figure 5: DECOS Prototype Cluster

meta-model for which the interpreter has been executed, i.e., in our case for each entity of the modeling paradigm of the DECOS execution platform the appropriate classes, attributes, and methods (e.g., setter- and getter-functions) are created. This framework has been exploited for realizing the interpreter functionality.

For the access to model entities, GME offers various APIs. The here introduced model interpreter utilizes the so-called *Builder Object Network 2* [18]. This API provides the encapsulation of the raw COM-objects within a C++ class library, which disburdens the programmer from manually managing COM objects.

A high-level pseudo-code representation of the interpreter functionality is shown in Figure 4. First, an empty XML file is created and the appropriate header tags are produced (cf. Line 1). Afterwards, the model interpreter hierarchically traverses all elements of the platform model and produces the XML contents. The lines 2–4 describe the generation of the XML contents related to cluster and core network, while the generation of the contents related to the individual DECOS nodes is represented at Lines 5–14. Finally, the closing XML tags required for a well-formed XML file are added (cf. Line 15).

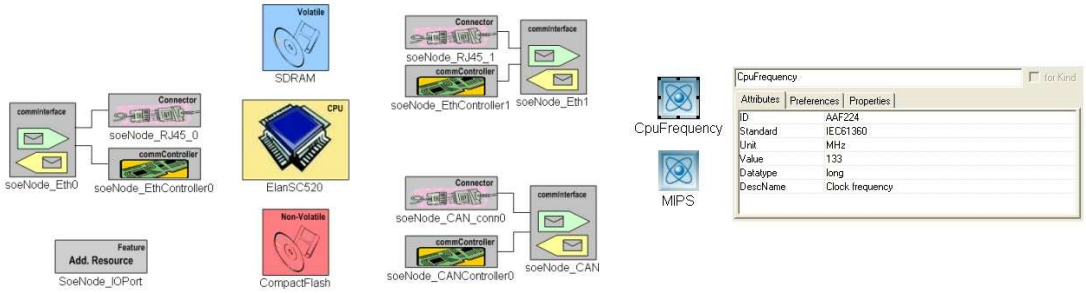
5 DECOS Execution Platform

We have applied the PME introduced in the previous section to model a real instance of the integrated DECOS architecture – a prototypical execution platform that has been developed in the course of the DECOS project [20]. The prototype (cf. Figure 5) consists of a cluster of five nodes using TTP/C [21] 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 single board computers, namely the TTTech monitoring node[2] for realizing the basic connector units and Soekris Engineering net4521 boards[3] for implementing the safety-critical and non safety-critical subsystems.

5.1 Hardware Element Viewpoint

The hardware element viewpoint of the PME is used to specify the characteristics of the hardware elements – the building blocks for the composition of the execution platform.

[2] `http://www.tttech.com`
[3] `http://www.soekris.com`



(a) Hardware Element Viewpoint (b) Detailed View of a Resource Primitive

Figure 6: Screenshots of PME Models of the Internal Configuration of the Soekris Engineering net4521 Single Board Computer

In our prototype cluster two such hardware elements exist: the TTTech monitoring node and the Soekris Engineering net4521 board, which is depicted in Figure 6(a).

The viewpoint specifies only those characteristics of the single board computer that are important for the DECOS development process. Thus, it describes the processor (a 133Mhz 468 class ElanSC520 processor from AMD), volatile memory (64 MBytes of RAM) and non volatile memory (512 MBytes CompactFlash) for program and data storage as well as communication interfaces (two 100 Mbit Ethernet interfaces and a CAN interface) and I/O ports. All of these entities represent resource primitives of the platform meta-model.

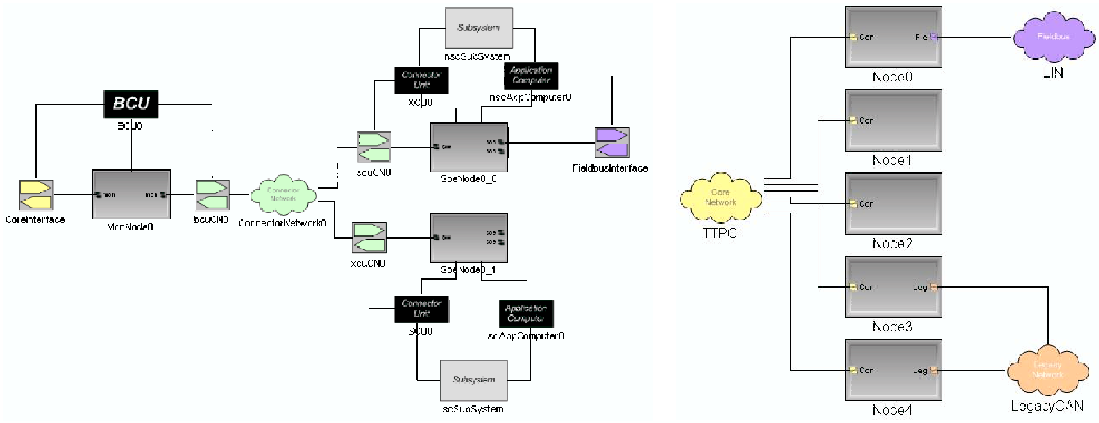
The specification of detailed information on resource primitives is depicted in Figure 6(b). Two detailed properties of the ElanSC520 CPU are exemplarily outlined here: the clock frequency and the instructions per second. Figure 6(b) further shows the usage of technical dictionaries as explained in Section 3.1.1: In the IEC 61360 standard[4] the property *AAF224* is defined as the clock frequency for micro processors enabling an unambiguous interpretation of this property of the ElanSC520 CPU.

5.2 Node Level Viewpoint

The node level viewpoint specifies the internal configuration of a DECOS node. For the nodes of the prototype cluster we employ distinct hardware elements as specified in the previous viewpoint for the basic connector unit (cf. *MonNode0* in Figure 7(a)), 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 Figure 7(a)).

In the prototype implementation the connector units of the DECOS node are interconnected by a time-triggered Ethernet network (*ConnectorNetwork0*), a standard Ethernet connection, where the arbitration of the communication medium is performed by Time Division Multiple Access (TDMA). 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 by the node.

[4]Standard data element types with associated classification scheme for electric components" - available at `http://dom2.iec.ch/iec61360/iec61360.nsf/`



(a) Node Level Viewpoint (b) Cluster Level Viewpoint

Figure 7: PME Screenshots of Different Viewpoints of the Platform Model

In the node level viewpoint it is further 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. Figure 7(a) depicts a configuration in which the architectural services and the applications of one subsystem share a single hardware element. Furthermore, for the non safety-critical subsystem an additional communication interface to a fieldbus network is specified.

5.3 Cluster Level Viewpoint

The cluster level viewpoint represents the highest level of abstraction of the platform model. 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 of possible additional physical networks like field buses. The specifiable properties of the networks include performance and temporal characteristics (e.g., maximum bandwidth, guaranteed latency) as well as physical characteristics like redundancy, topology, or the used physical layer. In Figure 7(b) the cluster level viewpoint of our prototypical DECOS execution platform is depicted. The cluster consists of five nodes, interconnected by a TTP/C network in star topology. In addition, two nodes are locally connected to a CAN [22] network while a third one accesses a LIN [23] field bus.

Across all separate viewpoints, the PME provides the user valuable assistance in modeling the execution platform. For instance, it prevents the user from creating 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, an interface in between is mandatory). Further on, multiplicities of associations are monitored. This way it is not possible, e.g., to connect communication interface resource primitives to more than one network. Finally, violations of the meta-model like 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 the DECOS architecture offers a model-based design process for distributed embedded real-time system. Due to comprehensive tool support,

embedded system engineers can rapidly capture the essential properties of the application and platform in formal models and perform successive model transformations down to the physical target system.

An essential part of the tool chain is the resource modeling editor, which is described in this paper. 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 well-known and accepted entities (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 has to interpret it.

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. The decision is based on the specified model or atom inter-relationships in the 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 is a powerful method to 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 (entire or only parts of) the platform model. Therefore, re-use of resource descriptions in different models can be simply performed by *drag and drop* operations. It thereby checks if the insertion of the model element at the specified position complies to 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] F.J. Winters. Design process changes enabling rapid development. In *Convergence International Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, October 2004. Delphi Delco Electronics Systems.

[2] C.J. Murray. Auto group seeks universal software. *EE Times*, 2003.

[3] 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.

[4] R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006. Available at http://www.springerlink.com.

[5] AUTOSAR GbR. *AUTOSAR – Technical Overview V2.0.1*, June 2006.

[6] 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 at SAFECOMP 2006, Gdansk, Poland*, Sep. 2006.

[7] DECOS. Dependable Embedded Components and Systems. project deliverable D1.1.1 – report about decision on meta model and tools for PIM specification. Technical report, December 2004.

[8] R. Obermaisser and B. Huber. Model-based design of the communication system in an integrated architecture. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*, pages 96–107, Dallas, USA, November 2006.

[9] R. Obermaisser, P. Peti, and H. Kopetz. Virtual gateways in the DECOS integrated architecture. In *Proc. of the Workshop on Parallel and Distributed Real-Time Systems 2005 (WPDRTS)*. IEEE, April 2005.

[10] P. Peti and R. Obermaisser. A diagnostic framework for integrated time-triggered architectures. In *Proc. of the 9th IEEE Int. Symposium on Object-oriented Real-time distributed Computing*, April 2006.

[11] OMG. Model Driven Architecture (MDA). Technical Report document number ormsc/2001-07-01, Object Management Group, July 2001. Available at `http://www.omg.org`.

[12] Esterel Technologies. *SCADE Suite Technical and User Manuals, Version 5.0.1*, 2005.

[13] DECOS. Dependable Embedded Components and Systems. project deliverable D2.2.3 – virtual communication links and gateways – implementation of design tools and middleware services. Technical report, December 2005.

[14] B. Huber, R. Obermaisser, and P. Peti. MDA-Based Development in the DECOS Integrated Architecture - Modeling the Hardware Platform. *Proceedings of the 9th IEEE International Symposium on Object and component-oriented Real-time distributed Computing (ISORC'06)*, Apr. 2006.

[15] M. Sundaram and S.S.Y. Shim. Infrastructure for B2B exchanges with RosettaNet. In *Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS 2001, Third Int. Workshop on.*, pages 110–119, 2001.

[16] G. Csertan, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D Varro. VIATRA - Visual Automated Transformations for Formal Verification and Validation of UML Models. *17th IEEE International Conference on Automated Software Engineering*, pages 267–270, 2002.

[17] 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*, Budapest, Hungary, May 2001.

[18] Institute for Software Integrated Systems, Vanderbilt University. *GME 5 – User's Manual, Version 5.0*, 2005. `http://www.isis.vanderbilt.edu/projects/gme/GMEUMan.pdf`.

[19] OMG. Uml 2.0 ocl specification, omg final adopted specification. Technical Report OMG Document No. ptc/03-10-14, Object Management Group, October 2003.

[20] 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.

[21] H. Kopetz and G. Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994. Vienna University of Technology, Real-Time Systems Group.

[22] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[23] LIN Consortium. *LIN Specification Package Revision 2.0*, September 2003.