MDA-Based Development in the DECOS Integrated Architecture – Modeling the Hardware Platform

Bernhard Huber, Roman Obermaisser, and Philipp Peti
Vienna University of Technology
Real-Time Systems Group
Treitlstr. 3/182-1, 1040 Vienna, Austria
{huberb,romano,philipp}@vmars.tuwien.ac.at

Abstract

Reduced time–to–market in spite of increasing the system's functionality, reuse of software on different hardware platforms, and the demand for performing validation activities earlier in the development phase raise the need for revising the state–of–the–art development methodologies for distributed embedded systems.

The Model Driven Architecture is a design methodology addressing these emerging requirements. Developing embedded systems according to this model-based paradigm requires a platform-independent representation of the functionality of the application as well as a precise model of the targeted hardware platform.

In this paper we introduce a meta-model for capturing the resources of hardware platforms realizing the DECOS architecture, which is an integrated time-triggered architecture aimed at the development of distributed embedded systems. Furthermore, we present a tool chain based on this meta-model that speeds up the modeling process and reduces the likelihood of human errors by facilitating the reuse of hardware building blocks from libraries.

1. Introduction

In order to satisfy the industrial demands on performance, dependability and cost with respect to a large variety of different platforms (e.g., car platforms), the current state–of–the–art system development methodologies for distributed embedded systems are heavily imposed to be reviewed, because of the strong competition between manufacturers and the requirement to continuously improve functionality with stringent time–to–market constraints [3]. In the automotive domain, e.g., a main requirement for reducing development costs is to enable the reuse of software on different platforms, i.e. reuse between car series and car types. Therefore, it is required to separate the hardware of an Electronic Control Unit (ECU) from the software on it [6].

Furthermore, in today's development cycles of embedded systems the validation of the system can only occur after the integration of the software onto the chosen hardware platform. However, in this late phase of the development process, any changes to the system would result in significant costs. By applying a virtual integration [5] based on models of the software as well as the hardware of the embedded system, design faults could be identified earlier in the development process.

These emerging demands are addressed by the *platform-based design* methodology [21] and the Model Driven Architecture (MDA) [17] where application development is decoupled from the implementation of the architecture. This requires a separation of the application logic from the underlying platform technology by providing models for the application and the hardware platform. The main challenge in designing a model for the hardware platform is to precisely capture the resources in a sufficient level of detail for integrating the application functionality onto the target platform, but preserving flexibility and extensibility in order to adapt the model to changing technologies with reasonable effort.

In this paper we provide a solution for capturing the resources of the hardware platform and the integration of this information into a development process based on the MDA. Our approach is tailored to the DECOS integrated architecture, which is a system architecture designed to combine the benefits of federated and integrated system design [11]. The DECOS architecture offers a framework for the development of distributed embedded real-time systems integrating multiple applications with different levels of criticality and different requirements concerning the underlying platform.

In the scope of this paper we present a meta-model that enables the designer to model hardware platforms according to the DECOS component model. The meta-model defines

resource building blocks, which allow the composition of hardware platforms out of already specified resources stored in a library. This also facilitates the modeling of constituting parts of a hardware platform in parallel by different experts, which is ideal for handling domains with a high level of heterogeneity (e.g., the automotive domain). For reducing the likelihood of human errors in the modeling process, we provide tool support for automatically validating the conformance of the modeled hardware platform to the meta-model.

The paper is structured as follows. Section 2 gives an overview on related approaches, which separate hardware platform specification from application development. In Section 3 the DECOS architecture is introduced. Section 4 covers the proposed system design methodology. While Section 5 elaborates on the meta-model for the resource specification, a concrete realization based on UML and a tool chain supporting the modeling process are presented in Section 6. The results are discussed in Section 7.

2. Related Work

A design approach for avionic systems that strictly separates between hardware and software development was introduced by Marchetto in [13]. He introduces the concept of *Blueprints*, which tackle the design of distributed avionics systems by organizing the required information for system specification in separate units, the so-called Blueprints, so that changes of system integration decisions can be transferred in a controlled way to the target system by simply altering the appropriate Blueprints. Marchetto distinguishes three types of Blueprints, namely *Application Blueprints* containing run-time requirements of the application such as processing and memory requirements and communication requirements, *Resource Blueprints* describing the physical topology of the architecture, and *System Blueprints*, which specify the mapping of the system description contained in the Application Blueprints to the physical system description given by the Resource Blueprints.

A similar approach for Integrated Modular Avionics (IMA) [1] systems is presented by Fraboul and Martin in [4]. The modeling of application functionality is based on the APplication EXecutive (APEX) [2] interface definition. For describing the available resources of the hardware platform, they introduce the *Architecture Model* in which *Cabinets* and *Intercabinets Communication Busses* (global data busses) are defined. Cabinets are further subdivided into *Modules* consisting of *Application Processing Units* and *Bus Interface Units*, which communicate over *Intermodules Communication Busses* (back-plane busses). The mapping of APEX partitions to particular Modules is established by the *Allocation Model*.

For managing the complexity of automotive systems there is an ongoing initiative driven by the AUTOSAR (AUTomotive Open System ARchitecture) development partnership. In [7] the aimed system design methodology for automotive electric/electronic (E/E) systems of AUTOSAR is outlined. Software components are described by their logical functionality irrespective of the actual physical hardware they are executed on later. Additionally, ECU resource descriptions exists, which represent physical and electronic attributes of that ECU. Based on this information and an additional system constraint description [7], configuration information for ECUs and software components is generated, which is exploited for the final deployment of the generated software executables on the actual ECUs.

The basis of the resource specification meta-model and the development process introduced in this paper is formed by the component model of the DECOS architecture. DECOS is an integrated time-triggered architecture providing certain services at the architectural level facilitating the development of integrated mixed-criticality systems for the automotive and the avionics domain by preserving the certifyability of the safety-critical application subsystems as available in federated systems. The consequential requirements on the architecture (e.g., strict separation of application subsystems by means of connector units) have to be captured by the meta-model presented in this paper.

3. The DECOS Integrated Architecture

The DECOS architecture [11] offers a framework for the development of distributed embedded real-time systems integrating multiple Distributed Application Subsystems (DASs) with different levels of criticality and different requirements concerning the underlying platform. A DAS is a *nearly distributed subsystem of a large distributed real-time system that provides a well-specified application service* [11]. Examples of DASs in present day automotive applications are body electronics, the power-train system, and the multimedia system. Structuring rules guide the designer in the decomposition of the overall system at a functional level and for the transformation to the physical level. In addition, the DECOS integrated architecture aims at offering to system designers generic architectural services, which provide a validated stable baseline for the development of applications.

3.1. Functional System Structuring

For the provision of application services at the controlled object interface, the overall system is divided into a set of nearly-independent DASs. Each DAS is further decomposed into smaller units called *jobs*. A job is the basic unit of work and exploits a *virtual network* [15] in order to exchange messages with other jobs and works towards a com-

mon goal. A *virtual network* is the encapsulated communication system of a DAS. All communication activities of a virtual network are private to the DAS, i.e. transmissions and receptions of messages can only occur by jobs of the DAS unless a message is explicitly exported or imported by a gateway. Furthermore, a virtual network exhibits predefined temporal properties that are independent from other virtual networks.

A *port* is the access point between a job and the virtual network of the DAS the job belongs to. Depending on the data direction, one can distinguish input ports and output ports. In addition, we classify ports into state ports and event ports depending on the information semantics of send or received message.

3.2. Physical System Structuring

During the development of an integrated system the functional elements must be mapped onto the physical building blocks of the platform. These building blocks are the *time-triggered physical core network*, *components* and *partitions*. The components are part of a distributed computer system interconnected by the time-triggered physical core network. A component 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) [12]. Components are the target of job allocation and provide encapsulated execution environments denoted as partitions for jobs. Each partition prevents temporal interference (e.g., stealing processor time) and spatial interference [9,20] (e.g., overwriting data structures) between jobs. In the DECOS architecture, a component can host multiple partitions and jobs that belong to different DASs.

3.3. Architectural Services

Generic architectural services separate the application functionality from the underlying platform technology in order to facilitate reuse and reduce design complexity. This strategy corresponds to the concept of platform-based design [21], which proposes the introduction of abstraction layers, which facilitate refinements into subsequent abstraction layers in the design flow.

The architectural services of the DECOS architecture are such an abstraction layer. The specification of the architectural services hides the details of the underlying platform, while providing all information required for ensuring the functional and meta-functional (dependability, timeliness) requirements in the design of a safety-critical real-time application. The architectural services serve as a validated stable baseline that reduces application development efforts and facilitates reuse, because applications build on an architectural service interface that can be established on top of numerous platform technologies.

In order to maximize the number of platforms and applications that can be covered, the DECOS architectural service interface distinguishes a minimal set of *core services* and an open-ended number of *high-level services* that build on top of the core services. The core services include predictable time-triggered message transport, fault tolerant clock synchronization, strong fault isolation, and consistent diagnosis of failing components through a membership service. An example of a suitable base architecture providing those core services is the Time-Triggered Architecture (TTA) [10].

Based on the core services, the DECOS integrated architecture realizes high-level architectural services, which are DAS-specific and constitute the interface for the jobs to the underlying platform. Among the high-level services are gateway services, virtual network services, and encapsulation services. On top of the time-triggered physical network, different kinds of virtual networks are established and each type of virtual network can exhibit multiple instantiations. Gateway services selectively redirect messages between virtual networks and resolve differences with respect to operational properties and naming. The encapsulation services control the visibility of exchanged messages and ensure spatial and temporal partitioning for virtual networks in order to obtain error containment.

4. System Design in DECOS

The design flow of distributed embedded systems can be decomposed into three phases, the requirement analysis, the subsystem design, and the system integration phase [5]. In order to exploit the inherent benefits of integrated systems [11] , the integration phase for designing systems based on the DECOS architecture has to cope with increased requirements like mapping of jobs to appropriate hardware partitions, configuring the communication system and the parametrization of the architectural services.

System design in DECOS is guided by the ideas of the MDA introduced by the Object Management Group (OMG). The primary goals of the MDA are *portability*, *interoperability*, and *reusability* of applications achieved by architectural separation of concerns [17]. For the description of the structure of distributed computer systems, *models* with various levels of detail and focus are used. 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 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

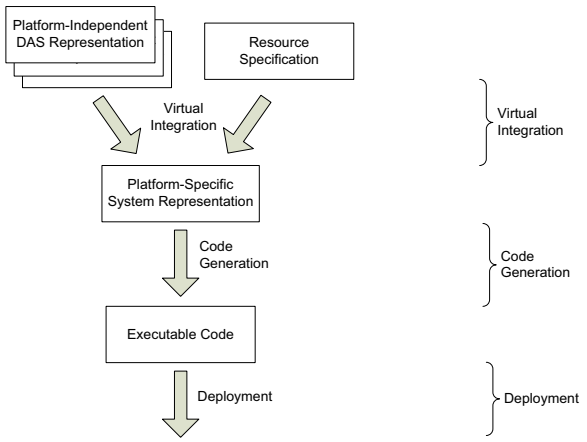

Figure 1. DECOS System Design Flow

computer systems are platform-independent and platform-specific viewpoints, which separate the application logic from the underlying platform technology. The MDA proposes such viewpoints – denoted as Platform Independent Model (PIM) and Platform Specific Model (PSM) – and defines their role in the design of a system [17].

This paper adapts this distinction between platform-independent and platform-specific viewpoints introduced by the MDA.

4.1. Platform-Independent DAS Representation

A PIM is a formal specification of the structure and function of a system that abstracts away technical details [16]. In the DECOS integrated architecture, the PIM structures the overall application functionality into DASs and jobs.

The identification of DASs is guided by functional coherence and common criticality of subsystems. A DAS should provide a meaningful application service (e.g., brake-by-wire service of a car) to its users at the controlled object interface. By associating with a DAS an application service that is relevant in the actual application context, the mental effort in understanding the various application services is reduced.

In addition, the identification of DASs is controlled by the criticality of the application services. In general, the realization of safety-critical services is fundamentally different from the design of non safety-critical services. While the first incorporate fault-tolerance functionality and focus on maximum simplicity to facilitate validation and certification, the latter are usually characterized by a larger amount of functionality and the requirement of flexibility and resource efficiency. By separating functionality with different criticality via dedicated DASs early on in the design testing, certification, and validation efforts can be matched to the criticality of each subsystem.

A major focus of the PIM is also the specification of the linking interfaces of the jobs within a DAS, as well as the specification of the interfaces between DASs. A linking interface consists of one or more ports through which jobs communicate. The linking interface specification captures operational properties (syntax, temporal constraints, interface state) and meta-level properties (e.g., dependability) of the messages exchanged via the port. The linking interface specification enables the independent development of jobs through separate vendors, because it provides each job developer with the required information about obligations and available services at the interfaces to jobs. Thus, a job developer knows which services must be provided and which services can be relied upon. In addition, a precise linking interface specification is also a prerequisite for reuse of jobs in different systems.

The PIM – including the specification of the linking interfaces – is expressed in Unified Modeling Language (UML) and constrained by a respective meta-model [18].

4.2. Resource Specification

The transformation of the platform independent DAS representations into the platform-specific representation of the integrated system is accomplished by the *Virtual Integration* (cf. Figure 1). As implied by the name, the Virtual Integration does not perform an integration of the DASs on a physical hardware platform, but operates on a virtual platform described by the *Resource Specification*. The aim of the Virtual Integration is to find a feasible allocation of jobs to partitions and mappings of virtual networks to the time-triggered core network. Based on the specification of the available resources of a concrete hardware platform, several checks can be performed that allow the identification of infeasible integration results at an earlier stage of the design phase than after physical integration on a particular platform.

Essential resource categories described in the *Resource Specification* are:

Computational Resources like processors and memory elements (volatile or non-volatile), which primarily determine the job-to-component allocation during the Virtual Integration. Sufficient processing power and memory capacity are prerequisites for hosting jobs on a particular component.

Communication Resources including communication interfaces, communication controller, and connectors determine the physically available network resources, the number of physical links to other components or the environment as well as their compatibility with respect

to supported physical layers or protocols. Only if the communication demands of a job (i.e. bandwidth, latency, and latency jitter) are fulfilled by the communication resources and a feasible communication schedule can be found, a job-to-component allocation might be considered as valid.

Special Purpose HW constitutes the third type of resource primitives that are described in the Resource Specification and used for judging the correctness of the Virtual Integration. They include, e.g., I/O ports and devices, sensors, actuators, or application-specific hardware blocks (e.g., hardware video decoder) provided by a particular component.

Besides the overall information on the available resources, the Resource Specification also includes the internal structure of the constituting components of the cluster, i.e. the allocation of resources to components and their internal communication. The Resource Specification is constraint by a meta-model, which is introduced in Section 5.

4.3. Platform-Specific System Representation

A PSM extends the specifications in the PIM with the details that define how the system uses the available platform resources. The generation of the PSM out of a PIM is constrained by dependability requirements, constraints with respect to hardware resources (e.g., CPU, memory), and constraints with respect to network resources (e.g., bandwidth). The PSM extends the PIM with the following information:

Allocation of jobs to partitions. The PSM contains a mapping between jobs and partitions within components. The establishment of this mapping is a complex optimization problem that needs to take into account numerous constraints. Firstly, the available component resources (e.g., CPU time, memory, special hardware) limit the number and the type of jobs that can be allocated to a component. Furthermore, when using n-modular redundancy, replicated jobs need to be allocated to different components that fail independently. In addition, specific application requirements can enforce constraints, e.g., physical proximity to sensors/actuators or collocation of jobs on the same component in order to prevent exchanged messages from consuming bandwidth on the time-triggered core network.

Mapping of virtual networks to the time-triggered core network. The PSM defines a communication schedule for the time-triggered physical network, as well as the subdivision of the communication resources for the different virtual networks. The media access control strategy of the time-triggered core communication service is Time Division Multiple Access (TDMA), thus dividing the channel capacity into a fixed number of slots. Every component is assigned a unique component slot that periodically reoccurs at a priori specified global points in time. In the PSM a component slot is further subdivided in so-called *virtual network slots*. Each virtual network slot contains those messages that are produced by jobs in the component that are connected to the respective virtual network.

Parametrization of high-level services. The actually deployed high-level architectural services are selected and configured during the transformation of the PIM to the PSM. This parametrization includes e.g., the instantiation and configuration of the generic architectural gateways between virtual networks via timed automata [14] or the definition of diagnostic checks [19].

5. Resource Specification Meta-Model

During the virtual integration phase of the DECOS system design (cf. Figure 1), the functional blocks of the distributed embedded system are mapped onto the physical building blocks of the DECOS hardware platform. Subject to this mapping are the time-triggered core network and the components of the cluster.

It is the purpose of the *Resource Specification Meta-Model* to enable the description of the available resources relevant for this mapping process. Relevant characteristics of the platform include amongst others computational resources (e.g., CPU and memory), communication resources, and dependability properties. However, depending on the concrete realization of the virtual integration, e.g., the objective function of the job–to–component allocation algorithm, the required information on the hardware platform varies. For instance, if multiple components are potential candidates for hosting a particular job and the minimization of hardware costs is an optimization objective, various information on the vendor, shipment and price of the involved hardware elements is required. Thus, one main objective of the meta-model is to provide a high degree of flexibility and extensibility with respect to the types and characteristics of the resources that are able to be described.

Specifying the characteristics of the exploited physical building blocks is a time-intensive engineering task. Since it is likely that identical hardware elements are deployed several times on the same hardware platform (e.g., a cluster consisting of identical components) or are also deployed on different platforms (e.g., an upgraded version of an already existing platform), the Resource Specification Meta-Model supports the definition of so-called *resource primitives*, which can be reused in different platform descriptions (see Section 5.1).

In order to complete the concept of composing DECOS platforms out of predefined building blocks, a framework for guiding the composition of resource primitives to larger physical hardware blocks is provided, ending up in a complete description of DECOS components (see Section 5.2). This framework reflects the DECOS *Component Model* as defined in [11], which strictly separates safety-critical and non safety-critical application functionality on an integrated mixed-criticality component.

5.1. Definition of Resource Primitives

Resource primitives are the smallest physical hardware units whose characteristics are captured and utilized for the virtual integration of DASs. They form the lowest level of the platform description with the highest level of detail concerning the provided information. In the *Resource Primitive Model* one can distinguish two categories of resource primitives. The first category is represented by a predetermined list of resource primitive types that are likely to be required in most of the described hardware platforms. The second category includes special purpose resource primitives realizing special features of a particular hardware platform (e.g., transducers, I/O devices, etc.). The resource primitives belonging to the first category are listed in the following.

Processor: The processor models the hardware unit that is responsible for the execution of the operating system of a hardware element, middleware services, and application jobs contained in partitions.

Memory: The Resource Primitive Model permits the distinction of volatile and non-volatile memory resources. During the virtual integration the available memory resources constrain, e.g., the allocation of jobs to particular components.

Communication Interface: Every DECOS component is at least connected to one physical network: the core network. Therefore, every component requires at least one communication interface that is associated with an adequate communication controller as well as a compatible physical connector.

Communication Controller: A communication controller in the Resource Primitive Model is a resource that performs self-contained access to component internal or component external networks.

Connector: It represents a physical connector that is required for establishing physical links between a physical hardware unit and a network (either component internal or external).

FPGA: The existence of FPGAs on a DECOS component is modeled by this resource primitive.

One aim of the Resource Primitive Model is to provide a high degree of flexibility and extensibility to platform descriptions with respect to the number and type of attributes captured for each resource primitive. This goal is addressed by the representation of resource primitives and their characteristics. Instead of explicitly defining a set of known resource primitives and characteristics, an open, expansible approach is followed in the Resource Primitive Model that allows the definition of new resource primitive types and characteristics.

The main challenge with this approach is to establish a common view on the meaning of a particular resource characteristic. Consider, e.g., the access time of a memory element. Depending on the deployed technology (SRAM, DRAM, Flash, hard disk, etc.) different definitions of "access time" with varying interpretations exist. For instance, the access time of a data element stored on a hard disk varies with its location due to different factors (e.g., seek time, internal hard disk cache hit, etc.) while it is constant for RAM elements. Thus, without a common interpretation of the resources and its characteristics the virtual integration process would become intractable. An approach to tackle this challenge is the establishment of so-called *Technical Dictionaries*. Technical Dictionaries originate from the field of electronic business–to–business relationships [22]. They are divided into several categories and each category consists of various components and sub-components with a common interpretation. Examples for such Technical Dictionaries are the RosettaNet Technical Dictionary[1] and the Component Data Dictionary of the IEC 61360 standard[2].

In the Resource Primitive Model the concept of Technical Dictionaries is utilized to establish a common understanding of the resources and their characteristics. Therefore, besides an attribute containing a natural language description of the entity, each resource primitive and its associated characteristics consist of a pair of attributes, *Standard* and *ID*, which creates a unique link to an entry in a Technical Dictionary. For instance *AAA061* in the *IEC 61360 Component Data Dictionary* denotes a micro-controller and *AAF224* its characteristic clock frequency.

5.2. Composition of DECOS Components

While the Resource Primitive Model guides the description of resource primitives and their characteristics, the next step when describing the hardware platform of the DECOS integrated architecture is concerned with the composition of components out of previously modeled resources. A typ-

[1] http://www.rosettanet.org/

[2] http://dom2.iec.ch/iec61360/iec61360.nsf

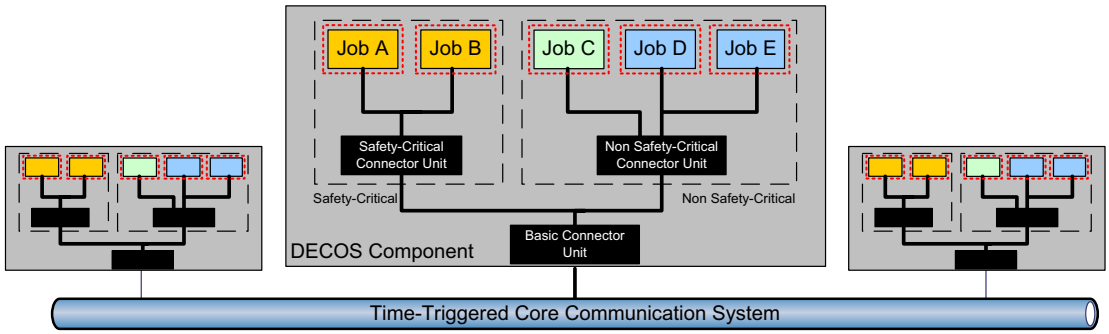

Figure 2. Components in the DECOS Integrated Architecture

ical setup of a DECOS cluster and the internal setup of its components is depicted in Figure 2.

A typical DECOS component is vertically structured into two subsystems. The *safety-critical subsystem* is an encapsulated execution environment for ultra-dependable applications. The *non safety-critical subsystem* offers an environment for those applications having less stringent dependability requirements. The safety-critical and non safety-critical subsystems are established by means of *spatial* and *temporal* inner-component partitioning [20]. The access of jobs to the time-triggered core communication system is controlled by *connector units*. The primary purpose of a connector unit is the allocation of network resources within a component that is vertically structured into two or more subsystems by ensuring that each subsystem obtains a predefined share of the overall network resources. In the DECOS component model we distinguish between three types of connector units (cf. Figure 2).

The *Basic Connector Unit (BCU)* performs the primary allocation of the physical network resources, as required for the separation of the safety-critical and non safety-critical subsystems of a component. The *Saftey-Critical Connector Unit (SCU)* allocates network resources to the jobs of the safety-critical subsystem and realizes the safety-critical high-level services (e.g., voting functionality). In analogy to the BCU, simplicity of the SCU is of major concern in order to control certification efforts. The *Complex Connector Unit (XCU)* performs the allocation of network resources for the non safety-critical subsystem of a component. Like the SCU, the XCU does not directly access the communication controller, but builds on top of the BCU. This way, the XCU is not involved in the fault isolation and error containment between the safety-critical and non safety-critical subsystem of a component, as this separation is performed by the underlying BCU. Therefore, the XCU and the non safety-critical subsystems of a component need not be certified to the highest criticality levels and the XCU can provide increased functionality at the cost of increased complexity. An analysis of the DECOS component model with respect to certifyability, encapsulation and independent development aspects can be found in [11].

A physical hardware unit that is capable of realizing (parts) of a component (e.g., a single board computer, an FPGA board, etc.) is denoted in the Resource Specification Meta-Model as *Hardware Element*. Hardware Elements are modeled by the composition of several resource primitives to a larger physical unit. The meta-model defines a set of mandatory resources that must be available on any Hardware Elements of the DECOS platform (e.g., at least one communication controller, communication interface, and connector).

The explicit modeling of Hardware Elements eases the management and the reuse of the modeled hardware for two reasons. First, it is likely that already generated descriptions of particular resource primitives are utilized for the composition of several Hardware Elements. These resource primitives need to be modeled only once and then are referenced by all Hardware Elements. This entails the advantages that the number of hardware descriptions can be held lower and changes to the description of particular resource primitives have to be performed only once and not in all Hardware Elements comprising those resources. Second, several components of a DECOS cluster may consist of similar or even identical physical hardware units. When explicitly modeling Hardware Elements as the constituting parts of DECOS components, hardware similarities can be exploited by reusing an already modeled Hardware Element several times in different components or by deriving descriptions of similar physical hardware units with small effort.

The concrete realization of a DECOS component is not predetermined by the component model. The functionality of each Connector Unit (BCU, SCU, or XCU) and Application Computer has to be realized by a Hardware Element of the component; however, whether a single Hardware Element realizes the entire component or multiple discrete Hardware Elements are utilized, is not restricted by the

component model of the DECOS architecture. If a component is made up of several discrete Hardware Elements, an internal communication infrastructure has to be set up. The Resource Specification Meta-Model comprises a set of constraints that ensure that the deployed Hardware Elements provide sufficient communication resources for establishing an intra-component communication (e.g., a Hardware Element realizing the BCU requires at least two communication interfaces: one towards the core network and one towards the intra-component network).

5.3. Resource Specification at Cluster Level

The last level of the hierarchical composition of the hardware platform of a DECOS cluster is formed by modeling the interconnection of the previously described components. In the DECOS integrated architecture the communication between components is established by a time-triggered core communication service. The Resource specification Meta-Model covers performance (bandwidth), temporal (latency and latency jitter), and dependability-related (redundancy and network topology) properties of the core network. Furthermore, the minimum number of required components in order to guarantee a successful startup in a bounded time or to fulfill the applied fault-hypothesis can be specified.

6. Implementation

The Hardware Specification Meta-Model is realized by UML models (using UML class diagramms) in order to formalize the rules and constraints guiding the description of hardware platforms for the DECOS architecture. Figure 3 depicts a partial UML representation of the meta-model showing the highest two hierarchy levels of the bottom-up platform modeling concept. The full UML-based implementation of the Hardware Specification Meta-Model can be found in [8].

In UML it is common practice to attach notes to respective UML entities in order to express additional constraints that are not covered by the formalism of UML. However, this practice is far away from defining constraints in a formal, machine-readable manner. Therefore, the UML models are augmented with Object Constraint Language (OCL) constraints tightening the specifications of the UML models. OCL provides means for formally specifying additional constraints, without enhancing the complexity and reducing the readability of the UML models.

OCL is a formal language for describing expressions on UML models. It can be used to specify invariants that must hold during the whole lifetime or only in particular states. With OCL, constraints at model level and meta-model level

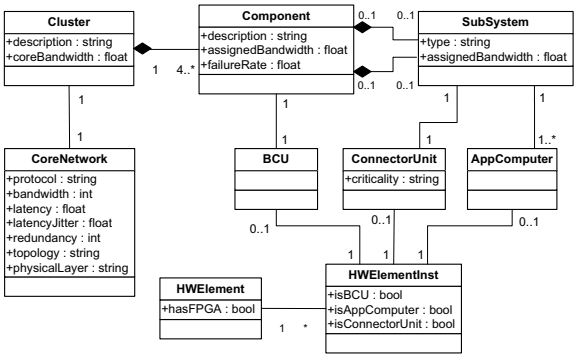

Figure 3. Partial UML Representation of the Hardware Specification Meta-Model

can be described. Thus, they can be used (i) to check instances of the model (objects) against the model (model level constraints) and (ii) to check the model itself against the UML specification (meta-model constraints). The entire list of OCL constraints developed for the Hardware Specification Meta-Model can be found in [8].

6.1. Tool-Aided Platform Modeling

This section elaborates on a workflow for tool-supported modeling of DECOS platforms, which is realized by the interconnection of Commercial-Off-The-Shelf (COTS) tools using the XML Metadata Interchange (XMI) file format. Since modeling the resources of DECOS platforms is a highly time-intensive and error-prone task, exploiting the benefits of UML, OCL, and COTS tools features various advantages. Especially when capturing the resources of a new target platform for the DECOS architecture from scratch, it is important to minimize the number of possible mistakes introduced by the system designer when instantiating the UML models. UML tools allow online-monitoring of the correctness of generated objects and links according to the meta-model and thus structural errors of the resource description are avoided. Furthermore, by the use of an OCL checker operating on UML entities, a high number of modeling errors can be identified automatically at an early stage of the modeling process.

The identified workflow is presented in Figure 4. Its central parts are a UML modeling tool, an OCL checker, and a transformation tool. The UML modeling tool is used to create a UML model (a UML object diagram) of the DECOS platform according to the mental model of the system designer and constrained by the meta-model introduced in the previous section. Thereby, the required resources are either modeled from scratch or imported from already existing platform descriptions. In order to validate the correct-

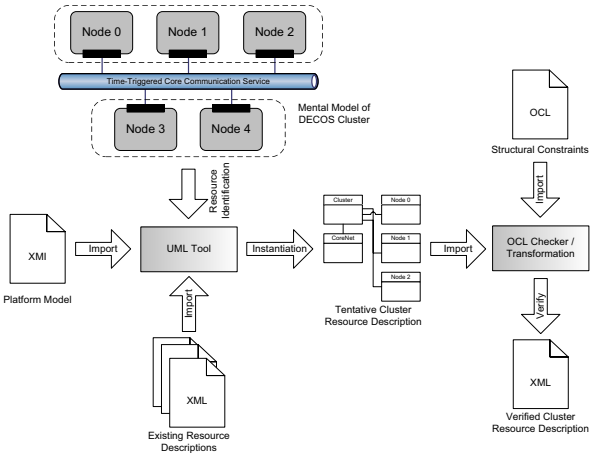

Figure 4. Platform Modeling Workflow

ness of the UML model, the Hardware Specification Meta-Model has to be imported to the modeling tool. Therefore, XMI is utilized.

Then, the resulting tentative description of the platform resources has to be validated using the OCL constraints, which mainly restrict the structure and interconnection of the objects and the concrete values of the attributes of the objects. Here again, XMI is used for importing the tentative cluster resource description into the OCL checker as well as for producing the output.

Since the interface to subsequent levels in the DECOS design flow, particularly the virtual integration, is specified using the Extensible Markup Language (XML) [8], a final transformation of the approved resource description has to be performed. This transformation is realized via an Extensible Stylesheet Language Transformation (XSLT). A transformation expressed by means of XSLT, denoted as *stylesheet* is specified as a well-formed Extensible Markup Language (XML) document. A stylesheet contains rules for transforming a source tree into a result tree. These rules consist of two parts: a pattern and a template. The pattern is matched against the source tree. In case of a successful match, the associated template is instantiated in order to create a part of the result tree. A final automatic validation ensures that the resulting XML document conforms to the defined specification.

7 Discussion

The presented framework for the specification of the hardware platform of a DECOS system facilitates designers in the meeting of stringent time-to-market constraints. Hierarchically organized libraries with reusable resource building blocks enable designers to amortize efforts for capturing resources across different components in a cluster, as well as across different projects. In addition, the resource specification framework supports the independent construction of the constituting parts of a hardware platform model through different designers. Thereby, we establish an important prerequisite for platform-based design with support for virtual integration in the context of the MDA.

Rapid Modeling of New Hardware Platforms and Modification of Existing Ones. The presented solution for the resource specification in the DECOS architecture supports the compositional definition of node computers and complete clusters. Consequently, a designer that needs to capture a hardware platform of a DECOS system needs not to start from scratch, but can build on resource building blocks stored in a library.

On the one hand, the resource specification framework provides a library with resource primitives (e.g., processors, memory chips, communication controllers). The library comprises elementary resources, which are no further decomposed in the resource specification. The library provides predefined resource categories, but is kept extensible and supports the addition of new instances of resource primitives. The library is not only reusable across different projects, but also speeds up modeling within a project. Even heterogeneous components of a DECOS cluster contain identical resource primitives. For example, consider a DECOS cluster with FlexRay as its time-triggered core communication protocol. Although the cluster can be composed of heterogeneous node computers (e.g., different host computers, special hardware), all node computers will contain a FlexRay communication controller for accessing the core network.

In addition to the resource primitive library, the resource specification framework provides a library with derived resources denoted as *Hardware Elements*. For example, the library can define an application computer as a hardware element, which is made up of a specific processor and memory chip. On their behalf, hardware elements can be further composed to form DECOS components and entire clusters.

Modular Resource Specification – Division of Work Among Experts. We utilize standardized technical dictionaries for ensuring the compatibility of the parts of the resource specification model, which are taken out of the libraries or defined by independent developers. The standardized technical dictionaries capture those attributes of resources that are significant for ensuring a seamless integration into complete components and clusters. Hence, different parts of the resource specification can be delegated to the respective experts. For example, a specific type of sensor can be modeled by an expert with profound knowledge in this domain.

Design Flow According to MDA and Virtual Integration Support. A resource specification model is a necessary prerequisite for a design process according to the MDA. The resource models and the tool support presented in this paper are enabling technologies for the design of integrated systems according to the MDA design methodology. Designers can perform a virtual integration of software subsystems for the identification of design faults early on in the development process. In addition, the platform-independent modeling of applications facilitates reuse, which is crucial in domains such as the automotive sector, where a particular software subsystem is subject to mass customization and needs to be deployed in different car types and car series.

Through the accompanying tool support (see Section 6), virtual integration efforts are minimized for mapping a PIM to different hardware platforms. Hence, designers are enabled to experiment freely with different hardware platforms. They can determine at relative ease a cost-effective platform, which meets all functional (e.g., availability of special purpose hardware) and non-functional requirements (e.g., temporal properties such as network bandwidth or CPU speed) identified in the PIM.

Acknowledgment

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] Aeronautical Radio, Inc. *ARINC Specification 651: Design Guide for Integrated Modular Avionics*, Nov. 1991.

[2] Aeronautical Radio, Inc. *ARINC Specification 653-1 (Draft 3): Avionics Application Software Standard Interface*, July 2003.

[3] E. Coelingh, P. Chaumette, and M. Andersson. Open-interface definitions for automotive systems - Application to a brake by wire system. In *Proc. of the SAE 2002 World Congress*, Detroit, MI, USA, Mar. 2002. SAE.

[4] C. Fraboul and F. Martin. Modeling and simulation of integrated modular avionics. In *Proc. of the 6th Euromicro Workshop on Parallel and Distributed Processing*, pages 102–110, 1998.

[5] P. Giusto, A. Ferrari, L. Lavagno, J.-Y. Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli. Automotive virtual integration platforms: Why's, What's, and How's. In *Proc. of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 370–378, Sept. 2002.

[6] B. Hardung, T. Kölzow, and A. Krüger. Reuse of software in distributed embedded automotive systems. In *Proc. of the 4th ACM International Conference on Embedded Software*, pages 203–210, New York, NY, USA, 2004. ACM Press.

[7] H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Maté, K. Nishikawa, and T. Scharnhorst. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proc. of the Convergence International Congress & Exposition On Transportation Electronics*, Oct. 2004. SAE.

[8] B. Huber, R. Obermaisser, P. Peti, and C. El-Salloum. Resource specification of the DECOS integrated architecture. Technical Report 54/2005, Vienna University of Technology, Institute of Computer Engineering, 2005.

[9] 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 3rd International Workshop on Intelligent Solutions in Embedded Systems*, pages 3–16, 2005.

[10] H. Kopetz and G. Bauer. The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software*, Jan. 2003.

[11] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22/2004, Vienna university of Technology, Institute of Computer Engineering, 2004.

[12] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proc. of 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 51–60, May 2003.

[13] A. Marchetto. Structured Definition of Modular Avionics Architectures Using Blueprints. In *Digital Avionics Systems Conference*, volume 1, pages 3.2–24–31 vol.1, 1997.

[14] R. Obermaisser and P. Peti. Specification and execution of gateways in integrated architectures. In *Proc. of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Catania, Italy, Sept. 2005.

[15] R. Obermaisser, P. Peti, and H. Kopetz. Virtual networks in an integrated time-triggered architecture. In *Proc. of 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 241–253, Feb. 2005.

[16] OMG. Model driven architecture: A technical perspective. Technical Report OMG Document No. ab/2001-02-01/04, Object Management Group, Feb. 2001.

[17] OMG. Model Driven Architecture (MDA). Technical Report document number ormsc/2001-07-01, Object Management Group, July 2001.

[18] A. Pataricza *et al*. Report about decision on meta model and tools for PIM specification. DECOS Project Deliverable D1.1.1, 2004.

[19] P. Peti, R. Obermaisser, and H. Kopetz. Out-of-norm assertions. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005.

[20] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA.

[21] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[22] M. Sundaram and S. Shim. Infrastructure for B2B exchanges with RosettaNet. In *Proc. of the 3rd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, pages 110–119, 2001.