A Model–Based Design Methodology with Contracts to Enhance the Development Process of Safety–Critical Systems

Andreas Baumgart[1], Philipp Reinkemeier[1], Achim Rettberg[2], Ingo Stierand[2], Eike Thaden[1], and Raphael Weber[1]

[1] OFFIS, Escherweg 2, 26121 Oldenburg, Germany
[2] Carl von Ossietzky University Oldenburg, OFFIS, Escherweg 2, 26121 Oldenburg, Germany

Abstract. In this paper a new methodology to support the development process of safety–critical systems with contracts is described. The meta–model of Heterogeneous Rich Component (HRC) is extended to a Common System Meta–Model (CSM) that benefits from the semantic foundation of HRC and provides analysis techniques such as compatibility checks or refinement analyses. The idea of viewpoints, perspectives, and abstraction levels is discussed in detail to point out how the CSM supports separation of concerns. An example is presented to detail the transition concepts between models. From the example we conclude that our approach proves valuable and supports the development process.

1 Introduction

In many application domains the field of distributed embedded systems has an increasing impact on the development of products. Since the products are often *safety critical systems* where erroneous behavior may lead to hazardous situations, standards such as DO–178B [12] (avionics) and ISO 26262 [6] (automotive) describe development processes in order to ensure functional correctness and safety properties. Only if a product is developed according to the relevant standards the respective certification authorities will approve the final product.

Another arising issue is the increasing size and complexity of the software (and hardware) of such systems. There are a number of reasons for this: The implementation of new functions often leads to inconsistent systems. Other functions have to be modified and in quite a few cases a modified technical architecture might be necessary to compensate the overall impact of new functions. It is a challenge to manage the development of such complex systems. A crucial part is a continuous and strong methodology to aid the development process in multiple dimensions.

In order to deal with complexity, existing design methodologies provide abstraction and refinement techniques. Abstraction allows the designer to concentrate on the essentials of a problem, and to enable concurrent development. Another important feature is the support for re–use. With this, systems can

evolutionary be constructed saving time and costs. Many methodologies provide re–use by a concept of components. Components encapsulate logical units of behavior that can be instantiated in different contexts. This also enables an incremental development process by a concept of refinement for components.

While many existing meta–models partially support such methodologies, they often do not cover the whole design flow from initial requirements down to a final implementation and do not support traceability of that process. A meta–model, to our understanding, provides the designer with the necessary modeling entities to comprehensively compose a system. In order to derive verifyable and executable embedded system specifications a meta–model needs to be interpreted which can be done by an underlying semantics. A methodology thereby describes the design steps of how a system should be modeled utilizing a meta–model.

In this paper we present a meta–model currently under development in the projects SPES2020 [16] and CESAR [14]. Some underlying concepts were already outlined in [17] and will be further detailed in this work. In the following, we will refer to our meta–model as the ***C**ommon **S**ystems **M**eta–model* (CSM). CSM features concepts to support component–based design, to specify formal and non–formal requirements and to link them to components. It supports the seamless design flow from the initial requirement specification down to the implementation. The meta–model for *Heterogeneous Rich Components* (HRC), developed in the SPEEDS [15] project, has been extended and constitutes the foundation of CSM. Thus, it benefits from the semantic foundation of HRC and provides analysis techniques such as compatibility checks or refinement analysis.

There already exist many meta–models, some of which will be briefly outlined and compared to CSM in Section 2. Since HRC is the core of our meta–model, we will give a short introduction to HRC in Section 3. Section 4 details how the CSM generically divides models to master complexity. In Section 5 we describe important CSM concepts along with an example model. The last section will draw a conclusion and give an outlook.

2 Architecture Description Languages

Many well established meta–models address a component–based development process including composition concepts. Some of them focus on a particular application domain and/or a certain stage of the development process. In the following we will describe the most relevant meta–models.

EAST–ADL [1] for instance aims at providing an open integration platform for modeling automotive systems and concentrates on early phases of development, i. e. specifying features of the product and analyzing its functions. However, modeling concepts for behavioral specification of functions are not part of EAST–ADL. According to the EAST–ADL specification UML state machines can be used here, but the semantics of their integration is not well defined. The CSM does not support the modeling of behavior directly. This is achieved through the concept of HRC contracts inherited from SPEEDS project to specify requirements for functional and non–functional behavior of components in con-

junction with assumptions on their environments. In EAST–ADL requirements can be categorized (e. g. different ASIL–levels) and also be linked to components — but they have no underlying formal semantics. The idea of having predefined abstraction levels in EAST–ADL is a valuable concept to separate models with different concerns, i. e. a functional model and an architectural design model. Thus, in CSM we adopt this concept in a more general way, which we call *perspectives*.

AUTOSAR [3] is based on a meta–model also targeting the automotive domain, but is utilized in a later step in the development process as it is more concerned with software configuration and integration aspects. The provided concepts for software component specification are nearly agnostic to the actual functionality a software component realizes. Instead they concentrate on defining modeling artifacts coupled with rules for code–generation to gain properly defined interfaces in order to ease the integration task. Concepts supporting functional modeling or earlier stages of the development process like abstraction levels are intentionally missing. Additionally, AUTOSAR does not consider requirement specifications, except for the new *Timing Extensions* [2]. In contrast to AUTOSAR CSM is meant to support a comprehensive modeling process starting from early requirements and going down to the actual implementation in a domain–independent way. If used in an automotive context, CSM models can be used to generate AUTOSAR artifacts which then can be used in native AUTOSAR configuration tools.

The AADL [5] is a modeling language that supports modeling and early and repeated analyses of a system's architecture with respect to performance–critical properties through an extendable notation (annexes). Furthermore, a tool framework and a formal semantics improves the usability and the usefulness in conjunction with analysis tools. AADL supports modelling of soft- and hardware components and their interfaces. These components serve as means to define the structure of an embedded system. Other features include functional interfaces like data IOs and non–functional aspects like timing and performance properties. The combination of components (connecting data in- and outputs or deployment of software on hardware) can be defined and modeled. Each of these components can contain a group connectors between interfaces of components forwarding control- or data–flows. Another feature of the AADL are so called modes attached to components. They represent alternative configurations of the implementation of a component. Like in EAST–ADL there is a way to describe system components, their interfaces, and the data they interchange. However, the designer starts to work at a stage where the software and hardware components are already explicitly identified and separated, i. e. modeling on higher levels of abstraction is not possible. Special annex libraries would have to be defined for exploiting the full potential of AADL, i. e. a formal specification of requirements and behavior not only limited to real–time specific constraints. Modeling on different levels of abstraction with refinement relations between abstract and more concrete artifacts is not supported in the AADL but might be useful to address e. g. separation of concerns or process–specific design steps.

The Systems Modeling Language (SysML) [9] standardized by the Object Management Group (OMG) extends UML2 with concepts for the development of embedded systems and is not tailored to a specific application–domain like EAST–ADL. Hardware, software, information, personnel, and facility aspects can be addressed. SysML can be used for the general notation of such a system so there is no explicit semantics for all elements and relationships. Furthermore, the language does not define architecture levels but provides means to describe them. There are structural block diagrams for component–based modeling, behavioral descriptions, requirements, and use cases. Moreover, SysML provides trace links which can be used to relate elements of different architectural levels: An element of an architectural level can realize another element, and an architectural property can be allocated to a property of another architecture model. Many concepts of SysML have been adopted in CSM, but key concepts like the concept of contracts with a rigid formal semantics are missing.

The *UML profile for Modeling and Analysis of Real-Time Embedded Systems* (MARTE) [10] adds capabilities for the real–time analysis of embedded systems to UML. It consists of several subprofiles, one of which providing a general concept of components. A subprofile called *NFPs* provides means to attach non–functional properties and constraints to design artifacts, that would later be subject to a real–time analysis. While this profile and its meta–model is agnostic to the application–domain, it does not detail the way constraints are specified. Despite being modular and open to other viewpoints, MARTE does not have a common underlying semantics that can be utilized for all viewpoints. In contrast, the HRC foundation of CSM has an automata–based semantics that can also be used for other viewpoints like safety and thus enables the specification of cross–viewpoint dependencies. This HRC foundation will be described in more detail in the next section.

3 Heterogenous Rich Components

HRC (*Heterogeneous Rich Component*), which originates from the european SPEEDS project, constitutes the core of the newly proposed meta–model. This section gives a short introduction to the concepts of HRC.

3.1 Structure

The meta–model provides basic constructs needed to model systems like components with one or more ports and connections (bindings) between them. A port aggregates multiple interaction points of the component typed by interfaces. These interfaces can be either flows that type data–oriented interaction points or services which type service–oriented interaction points. The interaction points of HRCs can be connected by means of bindings: Flows to flows and services to services. Either interconnections between ports and all their aggregated interaction points can be established or a subset of these interaction points are bound.

Components may have an inner structure consisting of subcomponents, their bindings between each other and their bindings from/to ports of their owning component. In the latter case the bindings specify a delegation of the flows or services to the inner component. This approach to model structure also common with most other meta–models for component based design, provides great reuse–capabilities and supports decomposition. Figure 1 depicts the concepts for modeling structure and their relations.

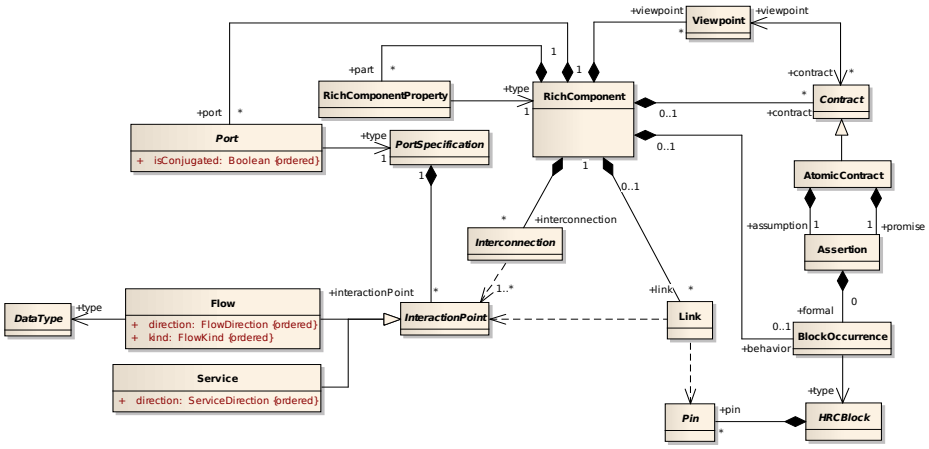

Fig. 1. HRC meta–model cut–out.

3.2 Behavior

The dynamics of an HRC can be specified by HRC state machines, which are hybrid automata with a C-like action language. These automata are wrapped by so called *HRCBlocks*, that expose parts of the automata on their *pins* to interact with them. In turn these pins can be linked to interaction points of the owning HRC, thus exposing the dynamics of the component.

The concept of HRCBlocks is used in different contexts, where a dynamic specification is needed. First the implementation of a component can be specified as an HRCBlock, consisting of an HRC state machine. More important HRCBlocks can be used when specifying the requirements of a component by so called contracts. These concepts for specifying behavior and contracts are depicted on the right hand side of figure 1. While the component–port–interface concepts, inherent to many other metamodels, allow to specify a static contract for a component, they often do not account for the dynamics of that interface. One of the key concepts in HRC is the ability to abstract from the actual implementation of components and to specify the required behavior using *contracts*. The idea of contracts is insprired by Bertrand Meyer's programming language

Eiffel and its *design by contract* paradigm [8]. In HRC contracts are a pair consisting of an assumption and a promise, both of which are specified by an HRC block. An assumption specifies how the context of the component, i. e. the environment from the point of view of the component, should behave. Only if the assumption is fulfilled, the component will behave as promised. This enables the verification of virtual system–integration at an early stage in a design flow, even when there is no implementation yet. The system decomposition during the design process can be verified with respect to contracts. Details about the semantics of HRC are given in [11], and [7] describes what a design process utilizing HRC would look like by means of an example.

3.3 Viewpoints

HRC allows to group one or more contracts together and assign them to user–defined viewpoints. Examples for viewpoints are realtime, safety, performance, or power consumption. While contracts associated with different viewpoints are based on the same underlying semantics, viewpoints support the developer to separate different concerns.

4 Modeling along the development process

When developing an embedded system an architecture is regarded in different *perspectives* at several *abstraction levels* during the design process as mentioned before. Figure 2 illustrates a generalized V–model which has become a well established model for many development processes. The V–model shows how an embedded system can be developed along several abstraction levels starting with user requirements for operational scenarios, analysis of functional blocks with refined requirements, design decisions with derived requirements and a final implementation. The developed product has to be tested on all levels so the implementation integration is tested. The behavior of the system design is verified, the functionality is validated and the product is evaluated during its operation. On each level the product architecture is regarded in different perspectives. So on an operational level e. g. non–functional features and use cases, on functional analysis level a perspective with functional blocks and on design level logics, software, hardware and geometry can be considered. Models on each perspective reflect different viewpoints. A viewpoint “Safety” might be regarded in every perspective but a viewpoint “Realtime” is not regarded in a geometric perspective and viewpoint “Cost” is not regarded when considering operational use cases.

4.1 Abstraction Levels

In CSM we introduce the generic concept of abstraction levels. In a model there can be several concrete abstraction levels to represent the different levels of granularity. We furthermore define an ordering relation between abstraction levels

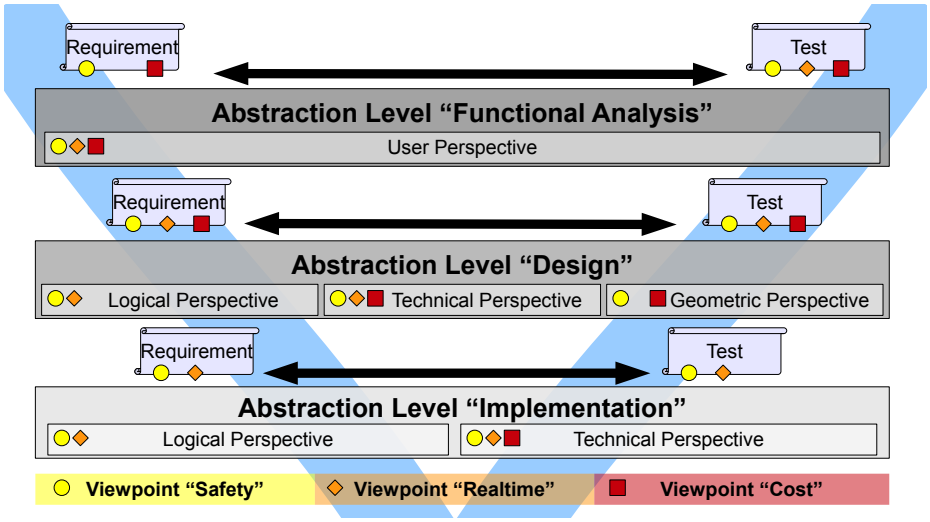

Fig. 2. Process model with different abstraction levels, perspectives and viewpoints.

where the highest abstraction level has the coarsest granularity in terms of description of the components. Lower abstraction levels are allowed to rearrange the functionality that is specified in a next higher abstraction level into different sets of components with respect to certain well–formedness properties. While the modeling granularity on lower abstraction levels is finer, the components still have to respect the functional and non–functional properties defined for their higher–level counterparts. This relation is formally defined in HRC based on HRC contracts as dominance relation (term used in SPEEDS) or entailment relation (renamed concept). Details for this can be found in Section 5.4.

For the CSM we observed that design processes are very diverse for different domains like automotive and avionics. A fixed set of predefined abstraction levels is not sufficient to handle the differences of multiple application domains. For example, in terms of CSM the EAST–ADL abstraction levels are just one possible instanciation of our generic abstraction level concept. In other application domains e. g. avionics or even for different companies in the same domain, the set of actual used concrete abstraction levels can be tailored to the specific needs. Each abstraction level contains one ore more perspectives which will be explained in the following.

4.2 Perspectives

Apart from the distinction of different user–defined abstraction levels, an abstraction level itself contains a set of model representations that reflect different aspects of the whole product architecture on the respective level of abstraction. We call such a distinct model representation on one abstraction level

a “Perspective”. Such a perspective can contain a model representation containing hierarchical elements and element interactions. Currently we distinguish four perspectives on each abstraction level: A user perspective which contains function–centric models, requirements engineering models, etc.; a logical perspective which describes the logical structure of the system with components, ports, connections, contracts, etc.; a technical perspective that focusses on physical hardware systems including communication buses; and a geometric model of the product.

The generic concept of different perspectives in abstraction levels is derived from the partitions of EAST–ADL abstraction levels and from an analysis of architecture partitioning. Taking a look at EAST–ADL one can see that one distinct EAST–ADL architecture level a parted into different perspective models: The design architecture level contains a functional as well as a hardware perspective. In [4] three architectures are defined namely a user level, a logical architecture and a technical architecture which are perspectives that can be found on several abstraction levels with a certain level of granularity.

On each perspective different viewpoints such as “Safety”, “Realtime” or “Cost” are regarded which will not be further discussed in this paper. Combining all perspectives of one abstraction level provides a model description of the whole architecture at one level of granularity. Element interfaces of different perspectives do not need to be compatible to each other. However, elements of one perspective can be allocated to elements of another perspective e. g. a feature is allocated to a function and a function is allocated to a hardware element and so on. In a graphical notation language such as SysML such an allocation is denoted by an “allocate” abstraction link.

The palette of actually used perspectives and elements can be different at each abstraction level e. g. distinction between hardware and software is not regarded in one abstraction level but in another. Thus, when descending to a lower level of abstraction elements of one perspective may be realized by elements in multiple finer–grained models of different perspectives.

5 A new Meta–Model with Example

This section will cover the most important features of our meta–model CSM starting from a top abstraction level where there is no hard- or software but only requirements and a basic idea, what the final product should be able to do. During the design process components are conceptualized to iteratively support the deduction of solutions from requirements. Further down on the lower levels of abstraction more concrete components can be devided into functions mapped to resources (e. g. software running on hardware). The descriptions will mainly focus on the support of the meta-model for development-processes involving different abstraction levels and perspectives wrt. the system under development. We will illustrate the usage of the concepts by a running example (the wheel braking system of an aircraft), that has been inspired by an example from the standard ARP4761 [13].

5.1 From required Features to Components

When starting the design of an embedded system one usually has consider a) the desired functionality of the system, b) the dedicated environment wherein the systems should work, and c) the existing regulations for building such systems. From this information the initial requirements concerning the system to design can be derived. As an example, a subset of required features concerning an aircraft's wheel braking system is displayed in Figure 3.

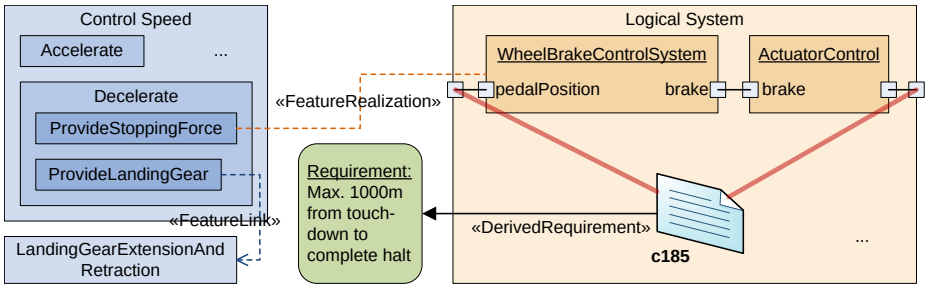

Fig. 3. Top abstraction level: Feature decomposition and initial logic architecture.

After the initial requirements specification via features a first high level logic component decomposition is done. In our example the aircraft to design has (among other components) a wheel braking control system and an actuator controller. To trace from where these components originate one can annotate feature realization connections to represent which feature is realized through which component. Required features can also depend on other features, this is displayed with a feature link. In our example there is a requirement which informally states that the aircraft shall stop within 1000 meters after touchdown. On the right side, in the logical system, this requirement was refined into multiple other requirements, one of which appears as the formally specified contract *c185*. This contract, for instance, requires the system to react to a pilot pushing the brake pedal with a signal towards the actuator control within a certain time–frame (*Assumption:* "PedalPosition is available && no system error" *Promise:* "Delay from *pedalPosition.e* to *brake.e* within [10,18] time unit [ms]").

5.2 Decomposing the System: Towards Lower Abstraction Levels

So far we presented a high level view on the system under development, usually referred to as System Model. In Figure 4 the logical perspective on the system abstraction level is shown on top. Below that the logical perspective of the next lower abstraction level can be seen. We used a one–on–one realization relation between components on the different levels. The contract *C185* used on the higher level is connected to a derived contract *C185.13* on the lower level using an *Entailment–relation*. This relation states that both contracts are not necessarily

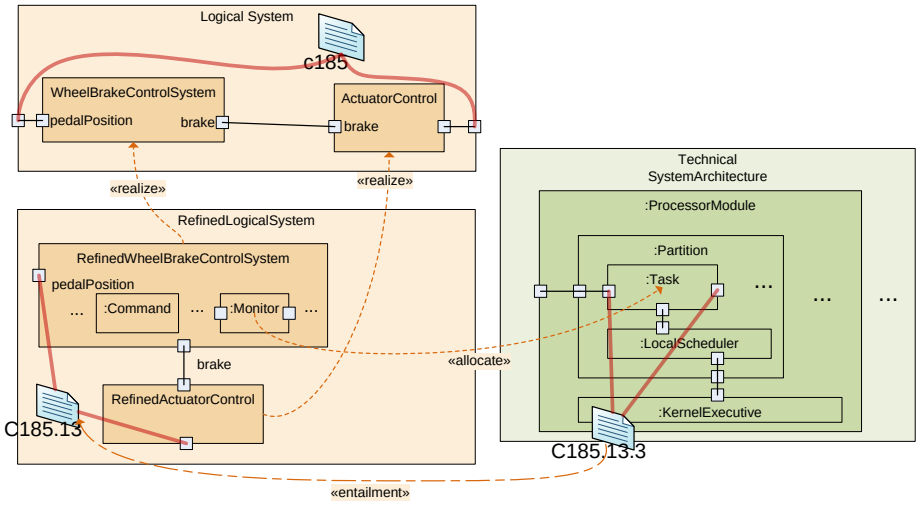

Fig. 4. System refinement and function allocation

identical, but the contract on the lower level is a refinement of its pendant on the higher level according to the SPEEDS semantics [11]. A new perspective is introduced on the lower abstraction layer: The *Technical Perspective*.

5.3 Allocating Logical Functions to Hardware Architecture Elements

As already discussed in Subsection 4.2, a model–based development process usually involves creating models for different perspectives of the system. The structure of the modeled system is not necessarily the same in the various perspectives, which requires means to correlate the different models to each other. Here we concentrate on the *logical* and *technical* perspective and the relationships among the contained models. As Figure 4 shows, we refined the initial logical functional specification and its requirements specified as contracts. Note that this is still part of the *logical perspective* of the system. On the right hand side of Figure 4, an excerpt of the model of the technical architecture is depicted along with an exemplary allocation of an instance of the function *Monitor* to a *Task*, that is scheduled with other tasks inside a partition of a hierachical scheduler running on an electronic control unit (ECU). This modeling–example also illustrates the benefit of the concept of perspectives as they allow to separate different design concerns. The functional specification of a *WheelBrakeControlSystem* is, at least in the early stages of development, independent of the underlying plattform hosting and executing the functions.

Note that the CSM also provides means to specify properties of tasks and their schedulers such as priorities, execution times and scheduling policies. But

as this is outside the scope of this paper these features are only briefly mentioned here.

5.4 Semantics of Realize and Allocation

So far we have introduced the concepts of abstraction levels and perspectives and illustrated their usage by an example. However, the power of CSM lies in its rigid semantic foundation allowing to apply model–checking techniques. Thus, we need a definition of *realize* and *allocation* according to these semantics. Figure 5 sketches this definition that relies on *HRC state machines*. The idea is to relate the observable behavior of components exposed at its ports.

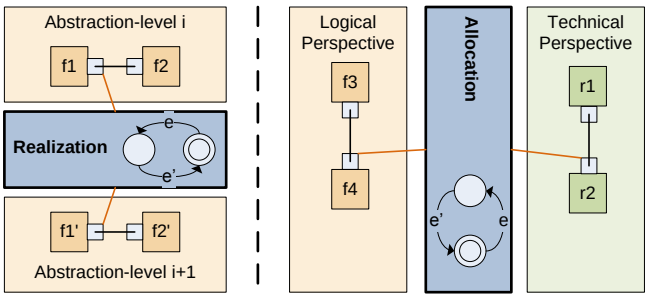

Fig. 5. Semantics of refinement and function allocation

Realization: Realizations are relationships between ports of components on different abstraction levels. Intuitively a realization–link states, that a component (e. g. *f1*) has somehow been refined and is now more concrete in regards to its interface and/or behavior (e. g. *f1'*). This cannot always be captured by a pure decomposition approach. Thus, we define the realization of a component by introducing a state–machine that *observes* the behavior of the refined component *f1'* and translates it into according events, that are observable at a port of component *f1*.

Allocation: Allocations are relationships between ports of components in different perspectives. Intuitively an allocation–link states that the logical behavior of a component (e. g. *f4*) is part of the behavior of a resource (e. g. *r2*), to which it has been allocated. Here we consider the same link–semantics as for the realization: It is defined as a state–machine that *observes* the behavior of the resource *r2* and translates it into according events that are observable at a port of the allocated component *f4*.

6 Conclusion

A new methodology to support the development process of safety–critical systems with contracts has been described in this paper. We first compared existing

meta–models also stating their short–comings in relation to our approach. Then we introduced HRC as the semantic foundation of our meta–model. In Section 4 we described our concepts of abstraction levels, perspectives, and viewpoints. We described the transition concepts between models in Section 5 along with an example.

It is hard to measure (in numbers) how well our model–based methodology competes with other approaches. But we think, from this work it can be seen how valuable these concepts can be to support a development process. Especially the realize and allocate relations deserve more in–depth research in the future since they hold the key to a persistent system design process.

References

1. The ATESST Consortium. *EAST ADL 2.0 Specification*, February 2008.
2. AUTOSAR. *Specification of Timing Extensions*, November 2009. Version 1.0.0.
3. AUTOSAR GbR. *Technical Overview*, August 2008. Version 2.2.2.
4. M. Broy, M. Feilkas, J. Grünbauer, A. Gruler, A. Harhurin, J. Hartmann, B. Penzenstadler, B. Schätz, and D. Wild. Umfassendes Architekturmodell für das Engineering eingebetteter Softwareintensiver Systeme. Technical report, Technische Universität München, May 2008.
5. P. H. Feiler, D. P. Gluch, and J. J. Hudak. *The Architecture Analysis & Design Language (AADL): An Introduction*. Carnegie Mellon University, Pittsburgh, USA, 2006.
6. International Organization for Standardization (ISO). *ISO 26262: Road vehicles – Functional Safety*.
7. B. Josko, Q. Ma, and A. Metzner. Designing Embedded Systems using Heterogeneous Rich Components. *Proceedings of the INCOSE International Symposium 2008*, 2008.
8. B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.
9. Object Management Group. *OMG Systems Modeling Language (OMG SysML ^{TM})*, November 2008. Version 1.1.
10. Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, November 2009. Version 1.0.
11. Project SPEEDS: WP.2.1 Partners. SPEEDS Meta-model Behavioural Semantics — Complement do D.2.1.c. Technical report, The SPEEDS consortium, 2007. not publically available yet.
12. Radio Technical Commission for Aeronautics (RTCA). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*.
13. Society of Automotive Engineers. *SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Warrendale, USA, December 1996.
14. The CESAR Consortium. CESAR Project. `http://www.cesarproject.eu`.
15. The SPEEDS Consortium. SPEEDS Project. `http://www.speeds.eu.com`.
16. The SPES 2020 Consortium. SPES 2020: Software Plattform Embedded Systems. `http://www.spes2020.de`.
17. J. Thyssen, D. Ratiu, W. Schwitzer, A. Harhurin, M. Feilkas, and E. Thaden. A system for seamless abstraction layers for model-based development of embedded software. In *Proceedings of Envision 2020 Workshop*, 2010.