

UML Metamodel Extensions for Specifying Functional Requirements of Mechatronic Components in Vehicles

Jörg Petersen¹, Torsten Bertram²

Gerhard-Mercator-University
Institute of Information Technology¹, Institute of Mechatronics and System Dynamics²
47048 Duisburg, Germany
[joerg.petersen@uni-duisburg.de]

Andreas Lapp, Kathrin Knorr, Pio Torre Flores, Jürgen Schirmer, Dieter Kraft

Robert Bosch GmbH, FV/SLN
70442 Stuttgart, Germany

Wolfgang Hermsen

ASSET Automotive Systems and Engineering Technology GmbH, SF/EAS1
70442 Stuttgart, Germany

Abstract: Increasing demands concerning safety, economic impact, fuel consumption and comfort result in a growing utilisation of mechatronic components and networking of up to now widely independent systems in vehicles. The development of networked electronic control units (ECU) as the most frequent mechatronic applications contains three core aspects: the development of the (control) functions itself, and their realisation in hardware and software as embedded systems. A co-ordinated, systematic and concurrent function, hardware and software development process including co-engineering and simulation environments requires a detailed specification in early development phases and a formalised description to improve the clearness of these specifications, decrease contradictions and increase information density. The Unified Modeling Language (UML) offers such a formalised description facility. A UML metamodel will be presented used for a mapping of automotive domain specific functional models onto UML models including constraints formalised by Object Constraint Language (OCL) expressions. The model also comprises the specification of functional interfaces together with a hierarchical decomposition of the system. The UML automotive domain models are basis for the system design and architecture and support aspects like re-use, exchangeability, scalability and distributed development. Particular importance is attached to the implementation of the UML model in a commercial tool together with a prototype checker of OCL expressions realised in Java.

1 Motivation and Introduction

Increasing demands concerning safety, economic impact, fuel consumption and comfort result in more and more complex vehicle functions. In order to fulfil these demands, an increasing utilisation of mechatronic systems and the creation of a car wide web,

connecting the up to now usually independently working systems in a vehicle, seem to be an appropriate solution. Electronic control units (ECU) are today the most frequent mechatronic applications; in upper class vehicles already over 80 ECUs control and monitor more than 170 control functions supported by approximately 450 electric servo motors. Individual components in the vehicle, in particular sensors, actuators, communication hardware and ECUs, are usually supplied by various manufacturers. To ensure quality, reliability and safety of such a complex networking system with components supplied from various manufactures, a detailed specification of the system is essential. In the analysis phase of system development such a detailed specification should cover an accurate requirements specification, the assignment of these requirements to functional units, the description of communication relationships between these units as well as a definition of the necessary interface data to enable these communications.

Actually, several groups of car manufacturers, suppliers and universities are working on architectural descriptions of these specifications in the automotive domain, e.g. [AW99], [Ho01], [Ko01], [La01a], [To01]. In the FORSOFT II project six abstraction levels of embedded systems in the automotive domain are identified: scenarios, functions, functional-networks, logical system architecture, technical system architecture and implementation [BR01]. In each level and development phase, different special suited tools are used, and notations of the Unified Modeling Language (UML) are proposed for the more abstract levels. The UML as an international standard passed by the Object Management Group [OMG00][OMG01] seems to be appropriate for a consistent modelling support throughout the entire development process. Despite some lack of semantical preciseness [EK99], the UML is widely used in the automobile industry, last but not least because of its support by diverse commercial tools. The UML offers extension mechanisms as well as the Object Constraint Language (OCL) suited to add domain related semantics to object-oriented models. A fundamental constraint on all extensions of the UML is to be strictly additive to the standard UML semantics.

Within this work, the UML is used to describe a structural automotive domain model with respect to functional requirements [Be98]. This so-called CARTRONIC function architecture as one element of an open and modular CARTRONIC system architecture [La01b] is based on a structuring concept for all control systems in a vehicle. UML metamodel extensions for formalising the mapping of CARTRONIC models onto UML models are presented in this paper. These extension mechanisms comprise stereotypes, constraints, tag definitions and tagged values. A proper stereotype design as well as constraints formalised by OCL expressions are introduced to secure the precision and consistency of mapped CARTRONIC models. Especially the stereotypes are a very powerful feature expressing domain specific restrictions and have to be designed very carefully [BGJ99]. Compared to the abstraction levels described in [BR01] and [Be01], the functions and functional-networks layer are the main focus of mapped CARTRONIC models, not the later software architecture layers.


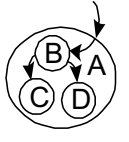
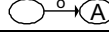
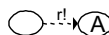
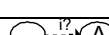
Section 2 gives a short overview of the CARTRONIC structural function architecture including a simplified torque control scenario as an example. Subsection 3.1 describes the idea of mapping CARTRONIC models onto UML models. The UML metamodel

extensions for formalising this mapping are presented in the rest of section 3. In the second subsection a hierarchy of stereotype classes is introduced, in the third the meta-modelling of communication relationships between the CARTRONIC components. Subsection 3.4 focuses on the specification of (real) time requirements for (control) functional parameters, in the last subsection the definition of OCL constraints being input for an automatic checker is introduced. The paper closes with a short summary, some hints on current work, and remarks on the relation of these UML metamodel extensions to UML-RT.

2 An Overview of the CARTRONIC Structuring Concept

The CARTRONIC structuring concept is a method to organise all control functions and systems in a vehicle. The CARTRONIC concept comprises clearly defined modelling elements as well as modelling rules to structure functional requirements in the analysis phase of system development. The result of a structuring according to CARTRONIC is a modular, hierarchical and expandable structure, which complies with these rules. The main ideas of the structuring concept are the hierarchical decomposition in functional units with defined tasks and defined functional interfaces. These functional units, so-called functional components, and communication relations between these components are the modelling elements of the CARTRONIC function structure. Functional components represent clearly defined tasks and are encapsulated by exactly defined functional interfaces. There are three different types of components: coordinators with mainly coordination tasks, operators with mainly operational tasks and information providers. The components do not automatically represent a physical unit in the sense of a constructing element or a control unit, but have to be understood as logical units. Components can be subdivided into sub-components representing the concepts of abstraction and encapsulation. Seen from outside, i.e. from the point of view of neighbouring components, a component of a function structure has to be interpreted as a system. Seen from inside, i.e. from the point of view of the sub-systems or the sub-components, the original component is only an enclosure integrating the entirety of their sub-systems.

Table 1: CARTRONIC modelling elements.

<i>Element</i>	<i>Description</i>	<i>Notation</i>
Component	Logical functional unit.	
System	A system consisting of several components respectively subsystems (view from inside to outside).	
Enclosure	Detailed component delegating communications to inner components expressing a part-of relationship (view from outside to inside).	
Order	Order to a component with the duty to execute a function.	
Request	Request to a component to execute a function with no obligation.	
Inquiry	Requirement for some pieces of information.	
Rule type	Structuring and modelling rules.	

The interaction of components is modelled by three different types of communication relations: order, request and inquiry. An order is characterised by the obligation to be executed by the instructed component. If the order is not executed, the receiving component has to give response to the ordering component, why the order could not be fulfilled. A request describes the demand of a source component to a target component to initiate an action or realise a function. Nevertheless, in contrast to the order there is no obligation to fulfil a request. This communication relation is used e.g. for the realisation of competing resource demands concerning power or information. The inquiry is used to get information needed for the fulfilment of an order or a request. If a component is not able to provide the requested information, it can notify the inquiring component accordingly. These communications offer a basic description of functional interfaces. Table 1 summarises the CARTRONIC elements with their graphical representations.

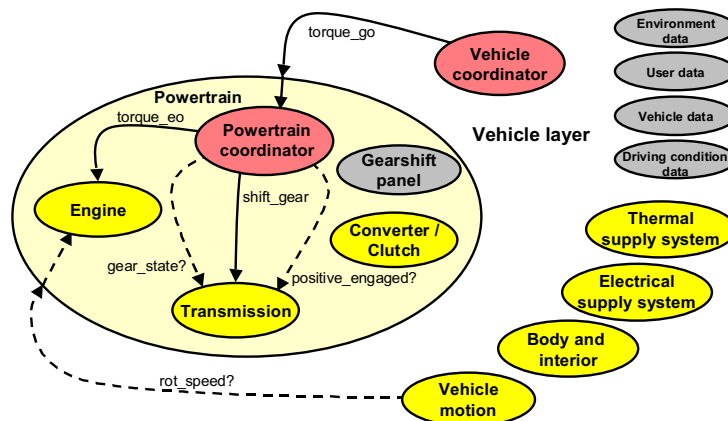


Figure 1: Example of a CARTRONIC function structure on a high level of abstraction (Vehicle layer) and the refinement of the functional component Powertrain.

Figure 1 shows an example of a CARTRONIC function structure. The Vehicle layer on the highest abstraction level contains a main component Vehicle motion, since the main task of a vehicle is the motion from one location to another. Additionally, there are three components to supply mechanical, electrical and thermal power: Powertrain, Electrical supply system, and Thermal supply system. As further components, there are Body and interior with mainly operational tasks and the Vehicle coordinator to coordinate the task and resources of the operative components. Four components serve as information providers: Environment data, User data, Vehicle data and Driving condition data. Since the complexity of these components is high, they have to be hierarchically decomposed into sub-components. As an example, this is shown for the Powertrain. It is structured into a set of more detailed components, i.e. Engine, Transmission, Converter/Clutch, Gearshift panel and Powertrain coordinator. These components have to be refined further for a more detailed description. In addition to the functional components, examples of communication relationships are given in figure 1:

- the order `torque_go` (to provide a certain torque at the gear output) is given by the Vehicle coordinator to the Powertrain and forwarded to the 'entry component' Powertrain coordinator,

- the order torque_eo (to provide a certain torque at the engine output) from the Powertrain coordinator to the component Engine,
- the inquiries gear_state? (present gear state) and positive_engaged? (transmission positive engaged) from the Powertrain coordinator to Transmission,
- the order shift_gear (to change the transmission gear) is given by the Powertrain coordinator to the component Transmission,
- as well as the inquiry rot_speed? (present rotational speed) from the Vehicle motion to the Powertrain, which is transferred to the responsible component Engine.

The following main features of a CARTRONIC function structure, developed in accordance with the structuring and modelling rules, can be listed [Be98]:

- defined, consistent structuring and modelling rules on all levels of abstraction,
- hierarchical decomposition of the system structure,
- hierarchical flow of orders with each component being assigned to only one orderer,
- high level of individual responsibility for each component,
- control elements, sensors and estimators are equal information providers,
- encapsulation, so that each component is only as visible as necessary and as invisible as possible for other components,
- realisation independency.

The CARTRONIC structuring concept can be used as the fundament to develop different types of vehicle and ECU configurations. It is intended to be open and neutral regarding automotive manufactures and suppliers. The resulting function structure is a basis to interconnect functions and systems of different origin, i.e. automobile manufacturers and suppliers, to a system compound. Essential for such an interconnection is the standardisation of the interfaces.

3 Extending the UML Metamodel for Mapping CARTRONIC Models onto CARTRONIC UML Models

Conceptually, a CARTRONIC model forms a structural architecture of components and connectors with respect to functionality. The mapping of a CARTRONIC model onto a formalised description is fundamental to improve the consistency of the domain model, to increase the degree of precision and specification, and to enable automated data exchanges between tools as well as transformations of models. For this formalised description the UML seems to be suited. It is a de-facto industry standard and comprises powerful extension mechanisms affecting the structure and adding semantics to user defined models by using stereotypes, constraints, and tagged values. In subsection 3.1 a consequent use of stereotypes for mapped components as well as relationships between these components is introduced. All further subsections describe the underlying extensions of the UML metamodel restricting the use of UML model elements and altogether implementing the CARTRONIC modelling rules.


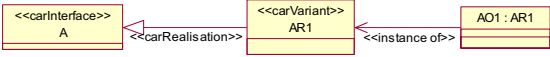
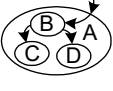
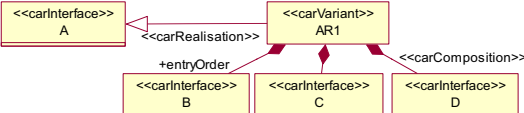
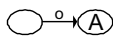
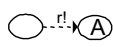
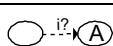
3.1 Mapping CARTRONIC Models onto UML Models and UML Objects

The mapping of CARTRONIC modelling elements onto UML modelling elements regarding the evolving UML profile for CARTRONIC is summarised in table 2. To avoid name clashes with existing UML stereotypes or well known profiles as UML-RT, most introduced classifier and relationship stereotype names are prefixed by *car*.

A CARTRONIC component A is mapped onto an (abstract) interface class named A (e.g. `<<carInterface>> A` in table 2). Such an interface class is used to collect all operations specifying the functions of a component, i.e. its external visible behaviour given by the CARTRONIC function structure. Each CARTRONIC interface class encapsulates the internal behaviour of this component. An interface A can be realised in one or more different variant classes `<<carVariant>> AR1, AR2`, e.g. representing and realising different physical or technical realisation principals. The UML abstraction dependency between an interface class and a realisation variant is stereotyped by `<<carRealisation>>`. Object instances like `AO1:AR1` of these variant classes ultimately represent realisations of CARTRONIC components as they are lastly implemented in ECUs. Modelling behaviour such object instances are used in UML behaviour diagrams. Finally in later software design and implementation models, most variant classes become singletons instantiated only once by a single object.

The hierarchical assignment of sub-components to a superior component is given by UML composition relationships stereotyped `<<carComposition>>` connecting the variant class, representing the superior component, with the interface classes of the sub-components. As an example, the refinement of component A onto the components B, C, and D is shown in table 2. Mapped CARTRONIC components managing all incoming orders additionally get the role name `entryOrder`.

Table 2: CARTRONIC modelling elements and their mapping onto UML.

<i>Element</i>	<i>Notation</i>	<i>Mapping onto UML modelling elements</i>
Component		
System Enclosure		
Order		UML-Operation with stereotype <code><<Order>> o</code> of <code><<carInterface>> A</code> .
Request		UML-Operation with stereotype <code><<Request>> r</code> of <code><<carInterface>> A</code> .
Inquiry		UML-Operation with stereotype <code><<Inquiry>> i</code> of <code><<carInterface>> A</code> .
Rule type		Extensions of the UML metamodel with stereotypes, relationships between stereotype classes, multiplicities, tags, and OCL expressions as constraints.

3.2 Extensions of the UML Metamodel for the Hierarchical Component Structure

In the following, CARTRONIC modelling rules are expressed by extending the UML metamodel. New stereotype classes are introduced, which are arranged in a generalisation hierarchy [OMG01, p. 2-87]. The introduced stereotype class hierarchy together with additional relationships between these classes restrict the UML mapping of CARTRONIC modelling elements. Locally as well as globally defined restrictions of CARTRONIC specific modelling rules are comprised as constraints formalised by OCL expressions.

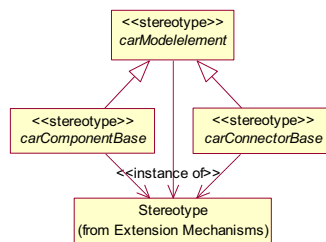


Figure 2: The root of the CARTRONIC UML metamodel stereotype hierarchy.

All newly-defined stereotype classes are instances of the UML metamodel class `Stereotype` from the `Extension Mechanism` package. Abstract stereotype classes are typed in italic names, they mainly serve in structuring the metamodel. The abstract root stereotype metaclass is called *carModelelement* (figure 2). Since a CARTRONIC UML model forms a structural architecture described by functional components and connectors, two principally different types of stereotype metaclasses are derived from the metamodel root, the abstract metaclasses *carComponentBase* and *carConnectorBase*.

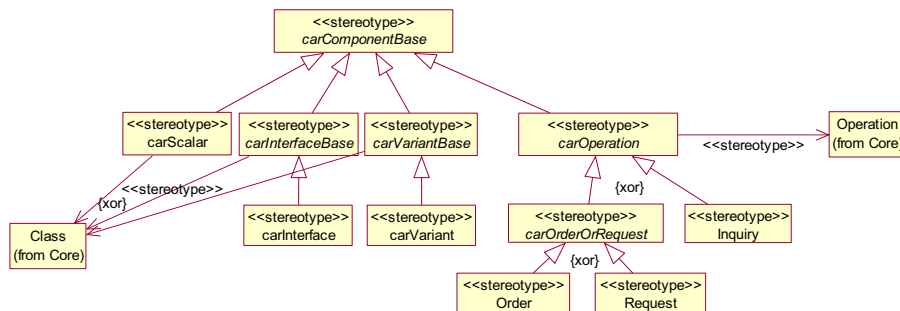


Figure 3: The stereotype hierarchy extending the UML metamodel for CARTRONIC components.

Figure 3 shows the stereotype hierarchy extending the UML metamodel for CARTRONIC components. The abstract metaclasses *carInterfaceBase* and *carVariantBase* are derived from the metaclass *carComponentBase* and associated to the UML metaclass `Class` from the `Core` package. From these classes the stereotype metaclasses *carInterface* and *carVariant* are derived. In later versions, additional

types of interface and realisation classes may be added. The stereotype *carScalar* for scalar values is also associated to the UML metaclass *Class*. The *xor*-constraint expresses, that only one of the three stereotypes can be assigned to a class in a CARTRONIC UML model.

The receiving component of a CARTRONIC communication relation offers a kind of functional service to the sending component. These functionalities are mapped onto UML operations in the UML *<<carInterface>>* classes with stereotypes *<<Order>>*, *<<Request>>* or *<<Inquiry>>*. In the extensions of the UML metamodel these three different metaclasses are inherited from the abstract stereotype *carOperation* (figure 3) respectively *carOrderOrRequest*. *CarOperation* is associated to the UML metaclass *Operation* from the *Core* package, the *xor*-constraint expresses, that only the usage of one of these stereotypes will be correct.



Figure 4: Extensions of the UML metamodel: each interface class contains at least one operation.

In figure 4 the aggregation between *carInterfaceBase* and *carOperation* specifies, that each *carInterfaceBase* class requires to have at least one offered *carOperation*, expressed very efficiently by the multiplicity expression *1..n*.

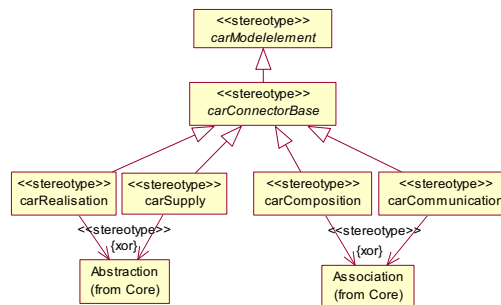


Figure 5: Stereotype hierarchy extending the UML metamodel for CARTRONIC relationships.

Connectors describe the allowed structural relationships between CARTRONIC components. For building component structure hierarchies, the two metaclasses *carRealisation* and *carComposition* are derived from the root metaclass *carConnectorBase* (figure 5). The stereotype *<<carRealisation>>* emphasises a UML realisation, an *Abstraction* derived from a *Dependency* relationship in the UML metamodel *Core* package. UML composition relationships are used for modelling the hierarchical assignment of sub-components to a superior component. A UML association with the stereotype class *carComposition* always connects a realising class with an interface class of a sub-component.

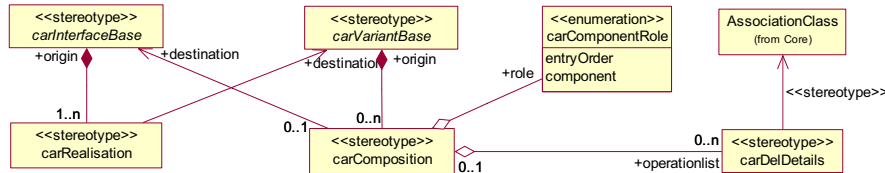


Figure 6: Hierarchy building relationships in the CARTRONIC UML metamodel extensions.

Figure 6 summarises the allowed hierarchy building relationships. They can be efficiently expressed by additional relationships between stereotype classes in the CARTRONIC UML metamodel extensions. Each metaclass *carInterfaceBase* has one or more *carRealisations* (multiplicity expression 1..n), but each *carRealisation* belongs to exactly one (encapsulating) *carInterfaceBase* (not written multiplicities are 1 by default) with public role name *origin*. A *carRealisation* connects (a *carInterfaceBase*) to exactly one *carVariantBase* with public role name *destination* at the association end of *carVariantBase*.

Each *carVariantBase* may have no (zero) *carComposition* relationships (a leaf in the CARTRONIC hierarchy) or an arbitrary number (multiplicity expression 0..n). Vice versa each *carComposition* starts from exactly one *carVariantBase* with role name *origin* (system/refined component). At the other end of a *carComposition* relationship exactly one *carInterfaceBase* is connected with role name *destination* (sub-component/subsystem), and vice versa each *carInterfaceBase* is part of exactly one *carComposition*. The multiplicity 0..1 is used because exactly one exception exists at the root of the composition tree; this exception has to be specified by an additional OCL constraint.

In the refinement of a component all operations of this component have to be delegated to the sub-components, and there has to be exactly one component as the target component for all forwarded orders called 'entry component' in the functional structure. The UML composition relationships, which are used for modelling the hierarchical structure, implicitly include the mechanism of delegation of operations respectively messages. To exactly specify a unique delegation mechanism, the stereotype class *carDelDetails* associated to *AssociationClass* in the Core package of the UML metamodel is defined (figure 6). Additionally, the unique entry component is specified based on a UML composition relation with the role name *entryOrder* defined in an <<enumeration>> *carComponentRole*. An additional OCL expression as a constraint can be specified to guarantee, that exactly one sub-component gets the role name *entryOrder* (see subsection 3.5).

As an example, figure 7 shows the refined functional component *Powertrain* from figure 1 without incoming communication relationships from other components. It is mapped onto the UML class <<carInterface>> *Powertrain* realised by the <<carVariant>> *PowertrainR1*, which is refined into the five sub-components: *Powertrain_coordinator*, *Engine*, *Transmission*, *Converter_Clutch* and *Gearshift_panel*, and all of them have the stereotype <<carInterface>>. They are

connected by (part-of) `carCompositions` to the class `<<carVariant>> PowertrainR1`. The `carInterface` of the component `Powertrain` offers the two operations `<<Order>> torque_go` and `<<Inquiry>> rot_speed` (compare to figure 1). The refined component `PowertrainR1` delegates `<<Order>> torque_go` to `<<carInterface>> Powertrain_coordinator` as the entry component for all incoming orders and `<<Inquiry>> rot_speed` to `<<carInterface>> Engine`. Their unique delegation is specified by the two associated classes with stereotype `carDelDetails`.

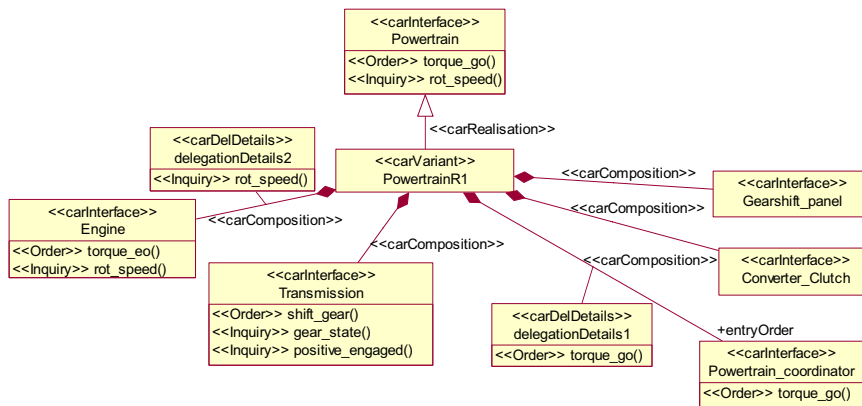


Figure 7: Example of mapping the refined CARTRONIC component `Powertrain` from figure 1 with definition of a unique delegation.

3.3 Extensions of the UML Metamodel for the Communication Relationships

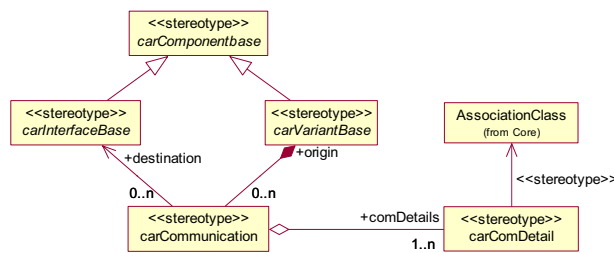


Figure 8: Communication relationships in the CARTRONIC UML metamodel extensions.

In figure 8 the allowed communication relationships are specified. The association between the stereotype metaclasses `carVariantBase` and `carCommunication` with multiplicity `0..n` expresses, that as much as required communication relationships from a realising component are allowed (including none). Vice versa the `carVariantBase` is a unique sender with role name `origin` for the `carCommunication`. A `carInterfaceBase` is the unique receiver in the role `destination` of a

torR1 to the <<carInterface>> Transmission prohibits a second <<Order>> shift_gear from another realising variant.

By and by combining, integrating, and enriching these UML models in the analysis phase of the entire development process will lead to more and more complete domain models being bases of the software architecture and design.

3.4 Scalar Data Interface Specifications in the UML Metamodel Extensions

For a complete function architecture, the so far discussed structural description has to be added by a behavioural specification. For automotive applications, this includes event-driven and time-driven functionality as well as (hybrid) combinations of both. Examples are wiping functionality, light switching, braking system (ABS), motor control, gear control or adaptive cruise control (ACC). Regarding especially the time-driven mechatronic systems, a data flow oriented behavioural description based on differential equations is used usually. These behavioural descriptions are essential for a consistent system development and mostly realised as simulation models in tools like Matlab/Simulink [Ma02]. By using simulation in an early stage of development, possible inconsistencies, errors and risks can be discovered and reduced or even eliminated. To use the function structure as a bases for a behavioural description, interface data have to be specified in more detail as discussed so far. This is especially relevant for the data values exchanged via the functional interfaces, which is discussed in the following.

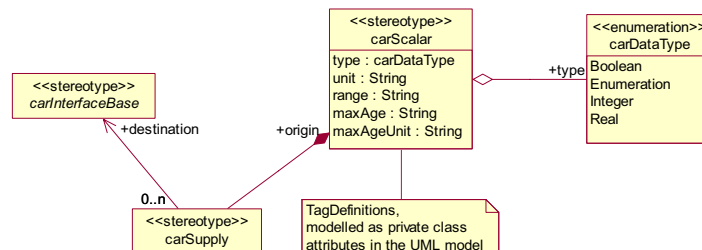


Figure 10: Tag definitions as physical data properties in the extended UML metamodel.

An inquiry for information like <<carInquiry>> rot_speed in figure 9 delivers an information, which is a scalar. Looking at such a scalar, it is important to specify the unit and the quality of the value. In terms of get-value-operations carInquiry asks for carScalar values (or compounds of them) with globally agreed type, unit, range, maxAge and maxAgeUnit properties (figure 10). The two properties maxAge and maxAgeUnit as tags describe the supplied actualisation rate, i.e. which (real-) time requirements the providing realisation classes have to guarantee for this scalar value later on. Additionally, in the UML metamodel extensions the stereotype metaclass carInterfaceBase is related to carScalar via association relationships to the metaclass carSupply (see also in figure 5 the association between carSupply and AssociationClass from the Core package).

Since the interpretation of tagged values is intentionally beyond the scope of UML, the semantics must be determined by user or tool conventions; as an easy to use convention, in figure 9 the use of private class attributes instead is shown. The <<carScalar>> `rot_speed` is supplied by <<carInterface>> `Powertrain` and <<carInterface>> `Engine` via the <<carInquiry>> `rot_speed` with a guaranteed actualisation rate of 5 ms.

3.5 OCL Expressions as Constraints for Further Domain Specific Restrictions

The extension mechanisms of the UML metamodel include the definition of constraints as domain specific restrictions. OCL expressions are used for a formalised specification of CARTRONIC specific modelling rules. These formalised rules allow an automated consistency check of fully stereotyped CARTRONIC UML models. Without such a formal metamodel only a manual check is possible.

In the following, some examples are given. OCL expression (1) specifies the CARTRONIC rule, that each component has to receive at least one order. This can be formalised by an OCL expression as an invariant for the stereotype metaclass `carInterface`. The constraint expresses, that in the set of all UML operations of a mapped functional component at least one has to have a stereotype `Order`:

```
context carInterface inv: (1)
    self.carOperation->exists(self.oclIsKindOf(Order))
```

Another domain restriction defines, that all orders reaching a refined component have to be forwarded from the enclosure to exactly one sub-component with the role name `entryOrder` (see figure 6), which has to co-ordinate all incoming orders. Based on the metamodel extensions, this can be expressed as an invariant (2) for the metaclass `carVariant` counting occurrences of `carComposition` relationships with role name `entryOrder`:

```
context carVariant inv: (2)
    self.carComposition->size > 0 implies
        self.carComposition->collect(role='entryOrder')->size = 1
```

Operations or attributes, which have the purpose to simplify OCL expressions describing constraints, can be defined as OCL pseudo-operations or pseudo-attributes [OMG01, p. 6-55][Wa99, p. 68, 71] serving e.g. as shortcuts in complex navigation expressions. For example, collecting all realising `carVariantBases` for a `carInterfaceBase` is defined by (3):

```
context carInterfaceBase def: (3)
    let directRealisations : Set(carVariantBase) =
        self.carRealisation.collect(destination)->asSet
```

Such shortcuts are very useful defining more complex restrictions, e.g. a correct use of the connectors `carRealisation` and `carComposition` leads to a directed graph structuring the defined components. A hierarchical tree structure of the components is given, if all nodes have input degree 1 (except for the root with input degree 0) and this directed graph is cycle free.

Looking at the allowed `carCommunication` relationships, a CARTRONIC UML model defines a directed communication graph. OCL expressions can be defined to restrict the graph to communication relationships allowed by the CARTRONIC modelling rules.

4 Summary, Conclusion and Future Work

The demand for innovative and advanced vehicle functions at reasonable costs leads to more and more complex vehicle functionalities as well as an increasing and more complex networking of mechatronic components. Therefore, a detailed and systematic specification of the entire vehicle functionality will be required based on a sound semantic foundation. Semiformal models as the CARTRONIC functional structure assist in structuring and managing complexity. A formalised and tool supported model of specified functional requirements can be achieved by mapping CARTRONIC functional components and communication relationships onto UML models. Therefore, in an early analysis phase of the entire mechatronic system development process the CARTRONIC UML model is a domain model with respect to functional requirements. The mapping is based on UML metamodel extensions including stereotypes, tag definitions, tagged values, and constraints. A hierarchy of stereotype metaclasses is introduced. Relationships between these stereotype metaclasses with multiplicities and constraints given as OCL expressions represent automotive domain specific semantics regarding control systems in vehicles. Such formalised constraints allow an automated consistency check of given CARTRONIC UML models concerning the domain specific modelling rules. Choosing standard UML, different currently available commercial UML tools can be used. Up to now, neither the commercial UML tools provide a full support of the UML metamodel extension mechanisms nor support the definition of OCL expressions together with an integrated checking mechanism. Therefore, a prototype of a checker is implemented independently from the UML tool in the programming language Java. The checker is reused from the object oriented modelling concept for software of electronic control units in vehicles (OMOS) [He00]. The CARTRONIC UML models are exported in a CARTRONIC XML file format given as input for this checker. This XML export file, the CARTRONIC UML metamodel extensions, exported as an XML file too, and the specified OCL expressions are input files to the checker. At the moment, the generated output is a textual report of violated multiplicities and constraints of checked models.

The CARTRONIC functional structure assists in structuring and managing complexity in the analysis phase of the overall development process. Functional architectures influence the subsequent architectures including the software architecture. For the simulation of the dynamic behaviour of the system in early development phases, the structural description of CARTRONIC UML models can be coupled to data flow oriented ones

like Matlab/Simulink models usually used in function development. Identifying such coupling points in an incremental mechatronic system development process model based on the V-model, one focus of current work is an automated structure and information exchange based on an XML files [Kn02]. From single simulation models of domain model parts further requirement data in the entire domain model can be gained. Putting them together into one complete model comprising different variants, the calculation of global requirements, integrity as well as aspects of correctness, consistency, and completeness of a chosen variant can be proven. Such functional variants, modelled by different realising classes in a CARTRONIC UML model, require an adequate variant handling supporting re-use, exchangeability, and scalability. Continuing in the software development process, the functional architecture model is a basis for the creation of different software (design) architecture models, which obey different non-functional requirements like hardware resources, costs, etc. Software architectures influence the subsequent design, and in general, breaking down such architectures into designs is not possible in a simple straight forward manner [He99]. UML-RT is conceptualised for the design of software architectures for embedded real-time software systems [SR98] and is also used for the design of embedded systems. Three principal constructs as UML metamodel extensions are defined. Capsules are complex, physical, possibly distributed architectural objects, interacting with their surroundings through ports; capsule functionality is realised by (networks of) state machines. Ports are interface objects with an associated protocol as abstract specification of the desired behaviour. Connectors are abstract views of physical communication channels between ports. Transforming and mapping the functional architectures described in this paper onto software architectures, regarding state machine based semantics the given counterparts are capsules and classes stereotyped by `carVariant`, ports and classes stereotyped by `carInterface` or `carComDetail`, connectors and classes stereotyped by `carCommunication` or `carDelDetail`. Using a software component model to describe software designs [La01b][MA00], strategies for grouping functional components to software components have to be developed based on defined criteria like characteristics and number of communication relationships, traffic analysis, actualisation rates of supplied scalar data, and so on, leading to domain specific types of design patterns.

Acknowledgements

We thank the anonymous referees for their helpful comments.

Bibliography

- [AW99] Advanced Information Technology - Workshop for Object Oriented Design and Development of Embedded Systems (AIT-WOODDES). <http://wooddes.intranet.gr/>, 1999.
- [Be01] Beeck, M. von der; Braun, P.; Rappl, M.; Schröder, Chr.: Modellbasierte Softwareentwicklung für automobilspezifische Steuergeräte. In [VDI01], 2001.

- [Be98] Bertram, T.; Bitzer, R.; Mayer, R.; Volkart, A.: CARTRONIC - An open architecture for networking the control systems of an automobile. SAE International Congress and Exposition. Detroit/Michigan U.S.A., 1998.
- [BGJ99] Berner, St.; Glinz, M.; Joos, St.: A Classification of Stereotypes for Object-Oriented Modeling Languages. In [FR99], 1999.
- [BR01] Braun, P.; Rappl, M.: Abstraction levels of embedded systems. OMER 2 Workshop, Herrsching, 2001.
- [EK99] Evans, K.; Kent, St.: Core Meta-Modelling Semantics of UML: The pUML Approach. In [FR99], 1999.
- [FR99] France, R.; Rumpe, B.: UML '99 - The Unified Modeling Language. Lecture Notes in Computer Science 1723. Springer-Verlag, Berlin, 1999.
- [He00] Hermesen, W.; Neumann, K. J.: Application of the Object-Oriented Modelling Concept OMOS for Signal Conditioning of Vehicle Control Units. SAE International Congress and Exposition, Detroit/Michigan U.S.A., 2001.
- [He99] Herzberg, D.: UML-RT as a Candidate for Modeling Embedded Real-Time Systems in the Telecommunication Domain. In [FR99], 1999.
- [Ho01] Hofmann, P.; Fasolt, J.; Geretschläger, P.; Sakretz, R.; Wohlgemuth, F.: Automotive UML - eine neue objektorientierte Entwicklungstechnik. Published in 'Elektronik Sonderheft: Automotive', 2001.
- [Kn02] Knorr, Kathrin; Lapp, A.; Torre Flores, P.; Schirmer, J.; Kraft, D.; Petersen, J.; Bourhaleb, M.; Bertram, T.: A Process Model for Distributed Development of Networked Mechatronic Components in Motor Vehicles. Accepted: IEEE Joint International Requirements Engineering Conference '02, Essen, 2002.
- [Ko01] Kokes, M.; Querfurth, A. v.: Methodik zur Entwicklung Elektronik im Fahrzeug. In [VDI01], 2001.
- [La01a] Lange, K. ; Bortolazzi, J.; Marx, D. ; Wagner, G.; Gresser, K.: Hersteller-Initiative Software. In [VDI01], 2001.
- [La01b] Lapp, A.; Torre Flores, P.; Schirmer, J.; Kraft, D.; Hermesen, W.; Bertram, T.; Petersen, J.: Softwareentwicklung für Steuergeräte im Systemverbund – Von der CARTRONIC-Domänenstruktur zum Steuergerätecode. In [VDI01], 2001.
- [Ma00] Mann, S.; Borusan, A.; Ehrig, H.; Große-Rhode, M.; Mackenthun, R.; Sünbül, A.; Weber, H.: *Towards a Component Concept for Continuous Software Engineering*. Berlin, ISST, ISST-Berichte 55, 2000.
- [Ma02] Mathworks. Simulink, Dynamic System Simulation for Matlab. <http://www.mathworks.com/products/simulink/>, 2002.
- [OMG00] Object Management Group (OMG): OMG Unified Modeling Language Specification V1.3, <http://www.omg.org/cgi-bin/doc?formal/00-03-01>, 2000.
- [OMG01] Object Management Group (OMG): OMG Unified Modeling Language Specification V1.4, <http://www.omg.org/cgi-bin/doc?formal/01-09-67>, 2001.
- [SR98] Selic, B.; Rumbaugh, J.: Using UML for Modeling Complex Real-Time Systems. Whitepaper, <http://www.rational.com/products/whitepapers/UML-rt.pdf>, 1998.
- [To01] Torre Flores, P.; Lapp, A.; Hermesen, W.; Schirmer, J.; Walther, M.; Bertram, T.; Petersen, J.: Integration of the ordering concept for vehicle control systems CARTRONIC into the software development process using UML modeling methods. SAE International Congress and Exposition, Detroit/Michigan U.S.A., 2001.
- [Wa99] Warmer, J. B.; Kleppe, A. G.: *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Object Technology Series, Reading/Massachusetts, 1999.
- [VDI01] VDI Tagung Elektronik im KFZ, Baden-Baden. VDI-Verlag, Bericht 1646, Düsseldorf, 2001.