T U M

INSTITUT FÜR INFORMATIK

Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description.

An overview on description concepts, models and methods.

White Paper of the IBM Corporation and TUM Technical Report

Manfred Broy, Mario Gleirscher, Peter Kluge, Wolfgang Krenzer, Stefano Merenda, and Doris Wild

TUM-I0915

Juli 09

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-07-I0915-0/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©2009

Druck: Institut für Informatik der
Technischen Universität München

Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description

An overview on description concepts, models and methods

Technical Report
of the
Technische Universität München

White Paper
of the
IBM Corporation

Summary

This paper discusses the concept of model driven system architecture development for the automotive industry. It outlines best practices and methods and introduces specific abstraction layers for complexity reduction.

A common standardised Automotive Architecture Framework is postulated to allow common description structures, increase re-use and better collaboration within the automotive value creation community.

Technische Universität München

Manfred Broy, Mario Gleirscher, Stefano Merenda, Doris Wild

IBM Deutschland GmbH

Peter Kluge, Wolfgang Krenzer

Munich, June 2009

Automotive Architecture Framework IBM TUM White Paper.doc

Table of Contents

TABLE OF CONTENTS 2

1 INTRODUCTION 3

2 BASICS OF SYSTEM ARCHITECTURES 4

2.1 DEFINITION AND SCOPE 5
2.2 LEVELS OF ARCHITECTURES AND ARCHITECTURE FRAMEWORKS 5
2.3 META ARCHITECTURE FRAMEWORK (MAF) 7
2.4 COMMON ARCHITECTURE FRAMEWORK (CAF) 9
2.4.1 TAXONOMY 9
2.4.2 ABSTRACTION LAYERS 11
2.5 DOMAIN-SPECIFIC ARCHITECTURE FRAMEWORK (DAF) FOR THE AUTOMOTIVE DOMAIN 13
2.5.1 OVERVIEW 13
2.5.2 AAF VIEWPOINTS 14
2.5.3 AAF LEVELS OF ABSTRACTION AND DETAIL 16
2.6 OPEN-MODELS.ORG – A PLATFORM FOR COMMON SENSE ARCHITECTURE FRAMEWORKS 17

3 ARCHITECTURE FROM A DEVELOPMENT PERSPECTIVE 19

3.1 INTRO – REASONING FOR INTERACTION OF METHODOLOGY AND PROCESS 19
3.2 USING A MODEL DRIVEN APPROACH 19
3.3 THE ROLE OF ARCHITECTURE IN THE MODEL DRIVEN SYSTEM DEVELOPMENT PROCESS 20
3.3.1 HOW DOES THE ARCHITECTURE INTERACT WITH THE CREATION OF DESIGN ARTIFACTS 20
3.3.2 OUTLINE OF THE SOFTWARE ENGINEERING METHODOLOGY 22
3.4 APPLYING MODEL DRIVEN ARCHITECTURE (MDA) CONCEPTS 22

4 GOALS AND REQUIREMENTS AS ARCHITECTURAL DRIVERS IN VIEWPOINTS 23

5 SUMMARY AND OUTLOOK 24

6 BIBLIOGRAPHY 25

1 Introduction

Modern vehicles have been growing to highly complex systems with considerable electronics and software content. Already today this complexity is difficult to master, but with the expected further growth of vehicle functionality and complexity new measures are needed to reduce this complexity of the vehicle and to streamline the related development and support processes. A proven way to reduce complexity is the introduction of the architecture paradigm. Architecture is generally defined as

> "the fundamental conception of a system in its environment embodied in its elements, their relationships to each other and to its environment, and the principles guiding its design and evolution" (/ISO42010/).

It is broadly understood to be the inherent structure, and an abstraction of a system (e.g. vehicle) in parallel.

An architecture description is an artifact that documents a system's architecture by means of decomposition, separation and integration of concerns thus allowing for dealing with them more easily.

Architectures are created for a number of reasons. From a practical perspective, experience has demonstrated that the management of large development projects across many different organizations (as typical in the Automotive industry) demand a structured, repeatable method for

- sufficiently describing and evaluating the system to be developed,
- communication among the system stakeholders,
- evaluating investments and investment alternatives, based on scenario and impact driven implementation decisions
- planning and managing the development activities, such as division of labour and integration
- verifying the compliance of the system's (or sub-system's) implementation.

Development of a vehicle architecture – its modelling and description – therefore is an important engineering step: A manifestation of the earliest design decisions (/Clements et al 2002/), and a prerequisite for a deeper understanding of a vehicle system and an effective reasoning and communication about it.

The Automotive industry can be regarded as a complex network (ecosystem) of highly interdependent organizations which collaborate in all phases of the vehicle lifecycle. Because of this characteristic it is necessary for the members of this ecosystem to agree on a somewhat common way for structuring and describing a vehicle in order to increase overall efficiency within the ecosystem.

The introduction of, as well as the agreement on some general elements of an architecture within a given domain (or ecosystem) has proven to be very beneficial for all members of the ecosystem. A set of general elements of an architecture described by use of a concise and consistent terminology is called an Architecture Framework. Several examples of architecture frameworks exist: The DoDAF (/DoDAF/) and the Reference Architecture for Space Data Systems (/RASDS/) are well known examples. Also there is an international draft standard (/ISO42010/) available from ISO/IEC and IEEE for the development of domain specific architecture frameworks.

An architecture framework provides a foundational framework with guidance and rules for modelling, documenting, developing, understanding, analysing, using, and comparing architectures based on a common denominator (/ISO42010/) across a (virtual) development organization (i.e. value net). It provides insight for external stakeholders into how a specific lead organization (e.g. OEM) develops products and product architectures. The intention of an architecture framework for the Automotive industry is to ensure that descriptions of vehicle architectures can be compared and related across different vehicle programs, development units and organizations, thus establishing the foundation for overall value creation efficiency, risk reduction and, ultimately, increased innovation.

In this paper a first draft of an architecture framework for the Automotive industry is presented as well as some guidelines of how to use it in the development process. In the following we refer to this architecture framework as Automotive Architecture Framework (AAF). The AAF draft tries to reflect existing standardization (e.g /ISO42010/) as far as possible while satisfying specific needs of the Automotive industry in parallel.

2 Basics of System Architectures

The term “architecture” stems from the Greek *αρχή* = beginning, origin and *τεχνη* = art. The second part of the term “architecture” may also stem from the Latin *tectum* = house, roof.

Applied in the context of systems architecture aims at the structuring of systems in terms of their subsystems and the way these subsystems are connected and cooperate forming a whole and thus constituting the overall system, product, or system-of-system. With the term architecture we always associate a notion of decomposition of a system into components, and the structure how these are connected and composed into a larger system or product.

Architecture helps in the following aspects in systems development:

- reducing complexity by
  - separation of concerns
  - decomposition: modularization of systems into subsystems
  - abstraction: information hiding
- facilitating communication amongst stakeholders
- organizing the development (process and organisation)
  - project planning and organization
  - divide and conquer: dividing the work packages into independent development tasks with separate specification (interfaces), implementation and verification (component test)
  - early verification
  - integration
- re-use
  - defining concepts, components, artifacts in general for (as units of) reuse
  - finding existing components for re-use, based on architecture and system properties

In the figurative sense the purpose of architecture traditionally is to ensure that the target system

- meets the requirements („utilitas“),
- is robust to changed / emerging requirements („firmitas“), and
- is clearly structured and well designed, following the concepts of sparceness and orthogonality, while avoiding redundancy („venustas“).

The better an architecture is described in the early phases of a system the better it can serve as a backbone for the entire development process. From the perspective of an industrial ecosystem like the Automotive industry, this requires some degree of standardization of basic architectural elements and the way these elements are described.

Such a standardization effort spanning the entire Automotive industry is AUTOSAR which aims at an open and standardized Automotive software architecture (/AUTOSAR/). Other examples for architectural standardization are the architecture frameworks which haven been introduced for the aerospace and defense industries (DoDAF, MoDAF, RASDS, etc.)[1].

For the architecture of a system we have to distinguish between the actual way the system is structured and the description of the architecture (architectural description).

[1] cf. /AUTOSAR/, /DoDAF/, and /RASDS/

2.1 Definition and Scope

A system is a group of interacting, interrelated, or interdependent elements forming a complex whole and providing a set of services that are used by a “user” (person, group of persons, enterprise, or other systems) to carry out a specific purpose (mission). System components consist of (mechanical, electrical) hardware, software, generate or process data, and interact with actors or consumers at the system or subsystem boundaries. A system is always clearly separated from its environment. Thus it has a system boundary, which defines the scope of the system. Often systems are connected to other systems or actors that interact with it. These entities outside the scope are part of the context of the system. It is usually the purpose of a system to interact with its context to fulfil a certain purpose and to deliver a certain services. The services of systems are also called their functions or their functionality. These services constitute themselves by the interaction of the systems with the actors of their context. The interaction can be in terms of force and energy (made available to the embedded system on the basis of sensors and actuators) or by pure information exchange. In a system model there is no essential difference between values denoting energy and values denoting pure information. Both cases can be captured in the modeling and description of systems by logical and mathematical models.

Only having defined the scope we can distinguish between the system and its context. The context is everything which is outside of the system but is relevant for the system (also called the environment), for the way the system functions, in particular, by interacting with the actors in the context.

2.2 Levels of Architectures and Architecture Frameworks

One of the basic ideas of Architecture Frameworks is to structure and decompose the system to develop in a common and well-defined way. Naturally some of the methods of structuring and decomposing are more generic and others are very specific to the modeled system. For example, the fact that a system can be described by its user-functions (black-box view) as well as by its implementing components (white-box view) is of a very generic nature. On the other hand decomposing a system into body, interior, chassis and engine is very specific to the automotive industry. These different types of generality lead to different levels of Architectures and Architecture Frameworks. (We use the term Architecture instead of Architecture Frameworks when we are closer to a concrete system to develop.) Similar to other approaches we distinguish the following levels ordered by its generality, beginning with the most concrete one:

- **Product Architecture (pA)** defines the architecture for a concrete product.
- **Product-line Architecture (plA)** defines the architecture for a specific product line.
- **Organization-specific Architecture Framework (oAF)** tailors the domain-specific Architecture Framework for a concrete company like BMW or Toyota.
- **Domain-specific Architecture Framework (dAF)** adds concepts, which are specific to a dedicated domain. The AUTOSAR architecture is an element of this level.
- **Common Architecture Framework (cAF)** defines concepts, which are necessary for any kind of system. Important representatives are *functional*, *logical* and *technical architecture*.
- **Meta Architecture Framework (mAF)** is the most generic layer, which is fully independent of any type of system to develop. It introduces terms like *component, interface, view*, *viewpoint* and *concern*.

Figure 1 illustrates the hierarchy of the different types of architecture frameworks and architectures establishing an architecture taxonomy which this paper is based on.

The basic two levels in this hierarchy introduce concepts of how to develop architectures for technical systems in general. Applying these concepts for a specific domain (e.g. an industry) leads to foundational commona-lities which will facilitate a more efficient interoperation of the stakeholders of the respective domain.

With regard to the Automotive industry a specific architecture framework will certainly contri-bute to improvements in all areas of product developing starting from the first reasoning about a new product, to communication between stakeholders of the value net, down to a more integrated development / tool environment.

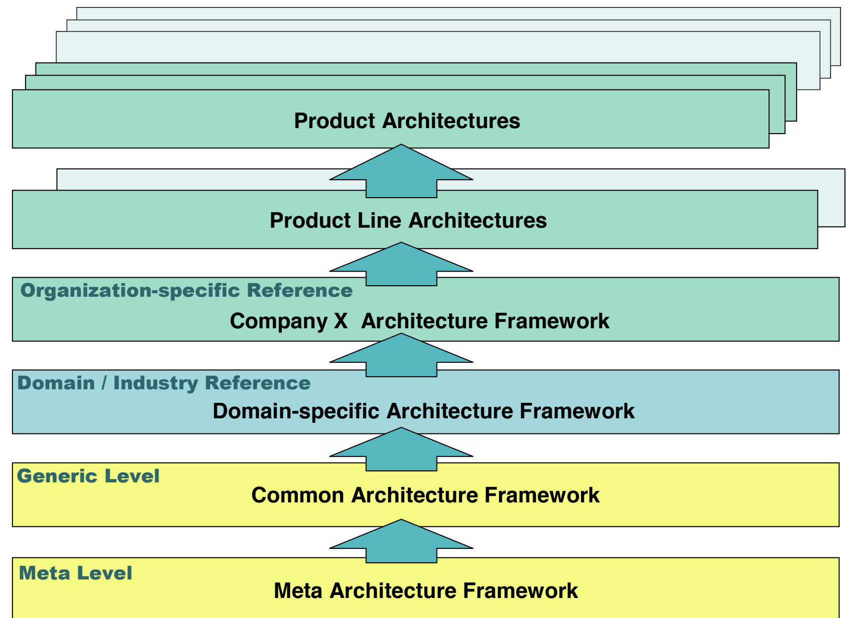

Figure 1: Hierarchy of Architecture Frameworks and Architectures

Having such a common base however does clearly not exclude the possibility to define specific architectures which support specific purposes (e.g. company specific goals). These architectures are represented in the upper three levels of the hierarchy (oAF, plA and pA). They are indeed based on the generic levels but nevertheless specific to a dedicated company and thus proprietary and not available for the public. For this reason we will concentrate on the first three layers in the following chapters. For the domain-specific Architecture Framework we will focus on the automotive domain.

2.3 Meta Architecture Framework (mAF)

The more complex a system is becoming the more difficult it is to understand. A technique which structures the system in “Views” from different “Viewpoints” (also known as “View Model”) has proven to be helpful to tackle the problems of systems’ complexity. In this section, we introduce some basic concepts, among others, namely *architecture views*, *concerns* and *viewpoints*, altogether used in many architecture frameworks. For the Meta Architecture Framework (mAF) we propose the following top level concept as illustrated in figure 2 which uses UML as a standardized way to describe static structures.

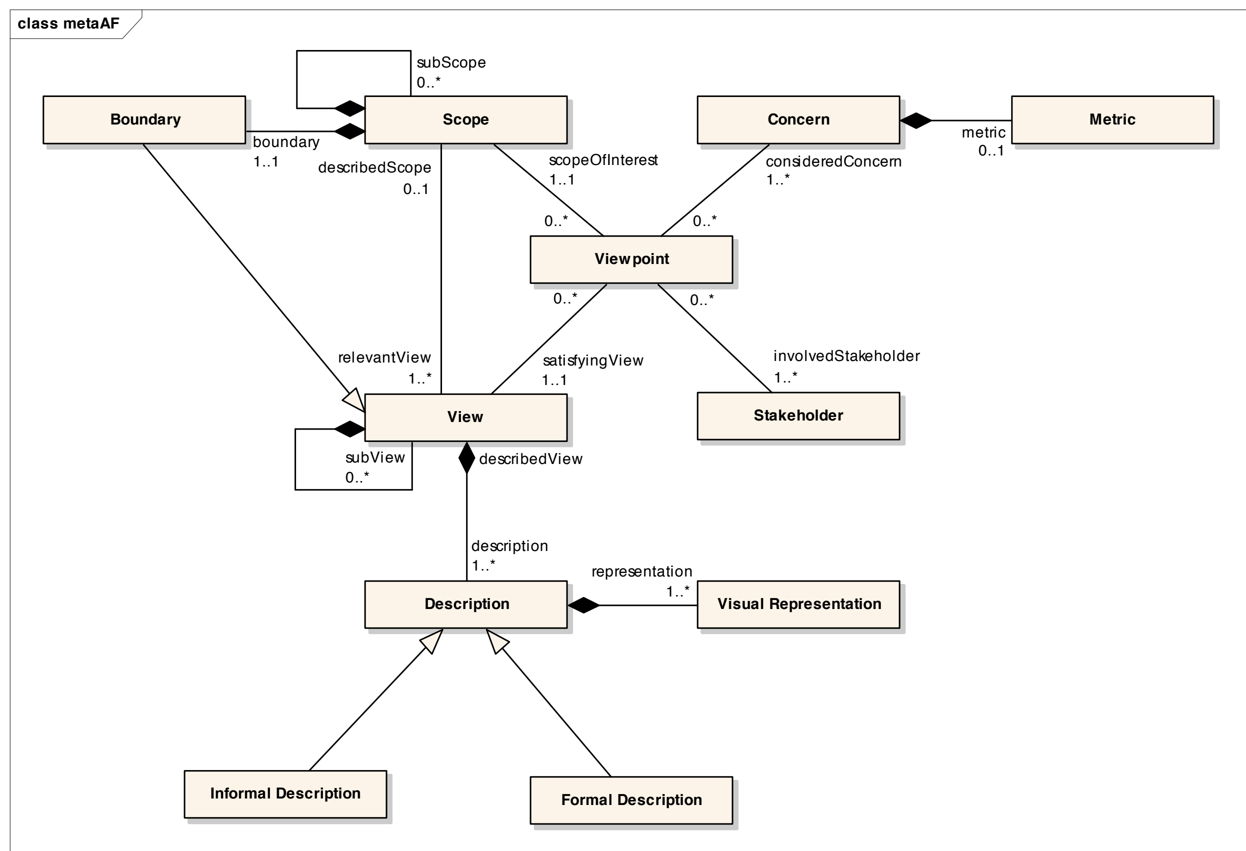

Figure 2: Metamodel/Taxonomy for the Meta Architecture Framework

Concern. Concerns have been introduced in systems engineering as specific areas of interest in a system pertaining to stakeholder goals and priorities (/ISO42010/) allowing interested stakeholders to focus on few things at a time (“separation of concerns”). Concerns are a way of expressing critical ‘holistic’ requirements which apply to the system as a whole rather than to any specific sub-set of its services or functionality (/Sommerville et al. 1998/). Hence, concerns are high-level criteria which reflect vague objectives, risks, and other problems. They usually refer to specific quality attributes like, e.g., safety, maintainability, reliability or cost.

Description. Descriptions are either formal or informal and separated from their representations:

- *Informal Descriptions* provide the contents of a view described by means of natural language and other informal ways of expression.
- *Formal Descriptions* provide the contents of a view described by means of logical and mathematical ways of expression and their model theories including formally defined semantics.
- *Visual Representations* (Notations) are textual, tabular, graphical or other visual kinds of representations. Opposed to that, non-visual kinds of representation would, e.g., be the sound of speech or electronically encoded data. Thus, representations are an externalization of informal or

formal descriptions.

Architecture Description. Architecture is the inherent structure or fundamental conception, and an abstraction of a system, e.g. a vehicle system. An architecture is documented in an architecture description in terms of the system's

- boundaries (cf. *scope*) and the interfaces to its environment, describing the way the system is connected to its context and interacts,
- traceable requirements
- decomposition into components or subsystems,
- the way the components or sub-systems are composed, connected and interact,
- the way the components or sub-systems contribute to the overall system functionality, and the
- principles guiding the system's design and evolution (/ISO42010/).

An architecture description has the goal to describe the structure and behavior of a system in a structured way. Structuring is achieved mainly through the introduction of architecture views and levels of abstraction.

Scope. A scope (of a system) is described by a set of relevant views. The relevance depends on the particular viewpoint. Regarding the notion of scope, we distinguish between two levels of maturity, *focus* and *boundary*:

- *Focus*. In early phases of system development, especially in requirements engineering (RE), scope is only characterized by focussing on a dedicated subject or object of analysis. We either do not know about or we may not be able to regard an exact boundary yet. Foci can be, e.g., interactors or parts of the product's interface and its operational environment, *indirect* stakeholders or characteristics of the considered application domain (/Sommerville et al. 1998/).
- *Boundary*. In architecture viewpoints, interface analysis or interface design decisions enable us to enhance a scope by a (system) boundary in order to precisely distinguish between the object of analysis and its context. We refer to a conceptual or technical boundary of a system (component) in order to analyse a set of (architecture) elements related – i.e. inside (system) or adjacent (context or environment) – to this boundary.

Stakeholder. A person – mostly a representative of a team or an organization – which is interested in the achievement of a set of goals (cf. Ch. 4) who emerge during the life-cycle of the system under consideration (SuC). Stakeholders strongly influence the content of a viewpoint, viz. its related *view*.

View. Architecture views represent a system from the perspective of an identified set of architecture related concerns which are framed in a so called architecture viewpoint. Hence, views can be seen as the content of a viewpoint, i.e., a subset of the description or modelling elements used in the architecture framework. We distinguish between two types of views: high-level views and ordinary views. Examples for high-level views are *architecture abstraction layers* like the view "logical architecture" (cf. Section 2.4). Ordinary views should satisfy viewpoints by providing the required content. During the phase of requirements analysis such content can be a subset of requirements coherent with the *viewpoint*'s *concern*s and *scope*.

An ordinary *view* is described by at least one *description*. By the help of scopes and views, we can distinguish between the structuring, i.e. organization, of several architecture views and the description of system structure and behavior, i.e. its manifold ways of decomposition, within a certain view. In other words, scope and view decomposition can and, for certain viewpoints, should be carried out orthogonally to system decomposition within the description techniques of one view, i.e. functional architecture and logical architecture.

Viewpoint. A viewpoint allows a user to examine a portion of a particular interest area. *Viewpoints* are considered as the central concept of organizing an architecture framework (AF) (/IEEE1471/). A viewpoint is a form of abstraction achieved regarding particular concerns with respect to the system (vehicle), its development process or its usage. Viewpoints and views enable stakeholders of the development process to comprehend complex systems and to organize the elements of the problem and the solution around domains of expertise. In our model, a viewpoint groups one or more considered *concerns* of one or more involved *stakeholders*. It thereby concentrates on a dedicated *scope* of interest. Every *viewpoint* is satisfied by exactly

one *view* of the system, which can be further partitioned into subviews. Architecture viewpoints are conventions for the construction, interpretation and use of a view and its model descriptions. In this draft a view conforms to one viewpoint (/ISO42010/).

Using a viewpoint approach like the ones from (/Sommerville et al. 1998/) or RUP-SE (/MDSD/) usually results in an individual set of viewpoints being specific to a domain or an organisation's product line. It can be reused in many similar development projects. In Section 2.4 we introduce a reusable set of common viewpoints also mandatory for the automotive domain. Furthermore, we regard viewpoints as an important point of integration with the activities and results of requirements engineering, cf. Chapter 4.

2.4 Common Architecture Framework (cAF)

In this section the *common Architecture Framework (cAF)* is described. It defines concepts, which are necessary for any kind of system. The *common Architecture Framework* is derived from the integrated architectural model presented in (/Broy et al. 2008/). The concepts of the *common Architecture Framework* are partitioned into three main parts, which are representing the total system on different layers of abstraction, namely the Functional Architecture, the Logical Architecture and the Technical Architecture.

2.4.1 Taxonomy

For the common Architecture Framework the following main concepts are identified as illustrated in figure 3.

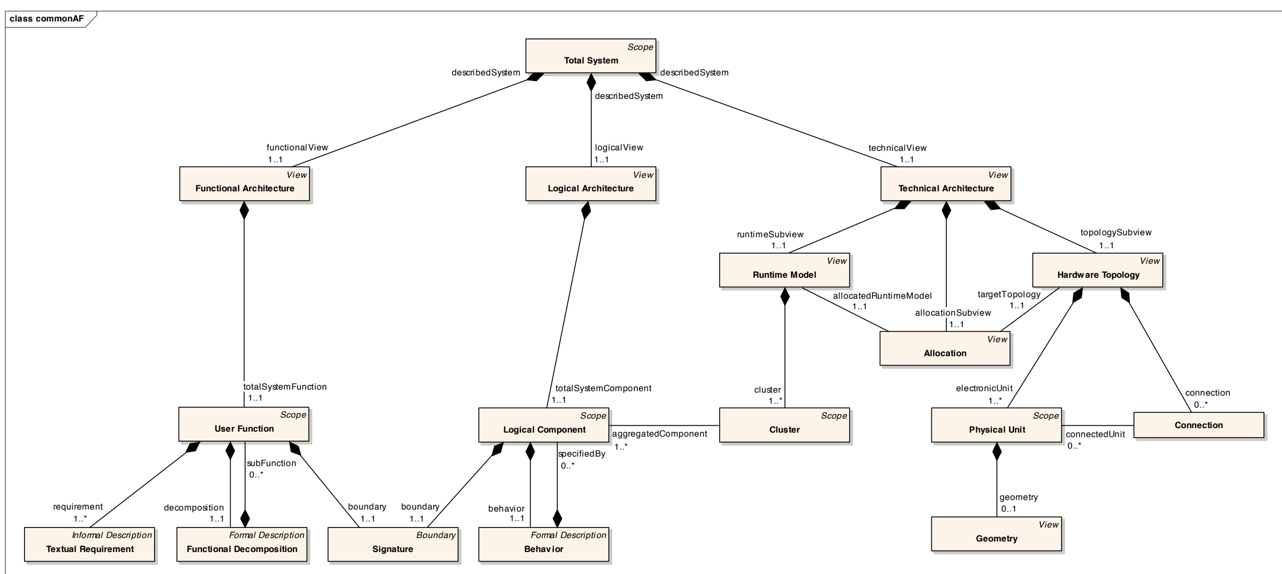

Figure 3: Metamodel/Taxonomy for the Common Architecture Framework

Allocation. The *Allocation* is a part of the *Technical Architecture*. It relates the elements of the *Runtime Model* to the elements of the *Hardware Topology*.

Behavior. According to figure 3 *Behavior* is part of *Logical Component*. It describes the dynamics of a L*ogical Component* which, in particular, manifests the system's logics and functionality in a formal and executable manner. With the help of the concepts of *Logical Component* and *Behavior* the functionality of physical existent parts of the *Total System* (i.e. a *Physical Unit*) can be described in a modular way and – moreover – can be simulated. When describing the *Behavior* of a *Total System* or a *Logical Component* the designer of the *Logical Component* does not need to know, how the *Total System* or the *Logical Component* will be realized. A *Logical Component* can be realized in the form of software, which is running on an ECU, or in the form of a mechanical device or an electronical/mechanical combination.

Cluster. The *Cluster* is the central element of the *Runtime Model*. It aggregates parts of the *Logical Architecture*. A *Cluster* describes the functionality of a *Physical Unit* or a part of it and thus a *Cluster* may represent a part of software, which is schedulable and deployable, or a description of the functionality of a mechanical device.

Connection. The *Connection* is a part of the *Hardware Topology*. It represents all hardware components which are used for the physical connection between two or more *Physical Units*.

Functional Architecture. The *Functional Architecture* describes the *Total System* from the black-box-perspective. It consists of a function hierarchy that contains the description of the functionality that is offered by the system to its outside world. Properties of the given hardware are not considered in this view.

Functional Decomposition. The *Functional Decomposition* separates the *Total System* into a family of *User Functions*. It describes the functionality of the *User Function* in a structured and formal way.

Geometry. The *Geometry* is a part of a *Physical Unit*. It describes the geometrical characteristics of a *Physical Unit*.

Hardware Topology. The *Hardware Topology* is a part of the *Technical Architecture*. It describes the structure of the used hardware platform. It mainly consists of *Physical Units*, which represent ECUs, sensors, mechanical components etc., and *Connections*, which represent busses, wires or the like.

Signature. The *Signature* is a part of a *Logical Component*. It describes the inputs and outputs (signals, events, messages) of the *Logical Component*.

Logical (Component) Architecture. The *Logical Architecture* describes the *Total System* from the white-box-perspective. It reflects the decomposition of a system into a number of *Logical Components* that interact and cooperate to offer the functionality, described in the *Functional Architecture*. Properties of the given hardware are not considered in this view. A part of the *Logical Architecture* can also describe the functionality of a subsystem, which is a defined part of the *Total System*.

Logical Component. The *Logical Component* is the central element of the *Logical Architecture*. It represents a modular unit comprising a *Signature* and a *Behavior*. Each *Logical Component* may in turn further consist of *Logical Components*. A *Logical Component* can also describe the functionality of a subsystem, which is a defined part of the *Total System*.

Physical Unit. The *Physical Unit* is a part of the *Hardware Topology*. It represents all physical components, which are interacting with the environment (e.g. sensors and actuators or mechanical devices), are processing software (e.g. microcontrollers) or are providing some functionality (e.g. hard-wired controllers or mechanical components).

Runtime Model. The *Runtime Model* is a part of the *Technical Architecture*. It describes the system's behavior on the abstraction level of the *Technical Architecture*.

Technical Architecture. The *Technical Architecture* describes the *Total System* from the realization-perspective. It describes, how the system that is specified by means of *Logical Components* can be integrated into a given hardware platform. It therefore consists of three parts: The *Runtime Model*, the *Allocation*, and the *Hardware Topology*.

Textual (Functional) Requirement. The *Textual (Functional) Requirement* is a part of the description of a *User Function*. It describes a *User Function* or its properties in an informal manner.

Total System. The *Total System* is a group of interacting, interrelated, or interdependent elements forming a complex whole and providing a set of *User Functions* that are used by a "user" (person, group of persons, enterprise, or other systems) to carry out a specific purpose (mission). It consists of (mechanical, electrical) hardware, software, data, and actors or consumers and it has a system boundary, which defines the scope of the system. It is described by the means of three main views, which are representing the total system on different levels of abstraction, namely the *Functional Architecture*, the *Logical Architecture* and the *Technical Architecture*. With the help of these three views parts of the *Total System* can be described, too. These parts are called *subsystems* and they can be seen as subscopes (cf. 2.3) of the *Total System*.

User Function. A *User Function* is a part of the *Functional Architecture*. It describes a piece of functionality which is visible to the system's environment. It consists of a *Textual Requirement*, a *Functional Description* and a *Signature*. Each *User Function* may in turn further consist of *User Functions*.

2.4.2 Abstraction Layers

Layers of abstraction are introduced to support a holistic understanding of the system and its architecture, as well as iterations between the requirements level and a preoccupation with the (overall) integrated system and lower level details of it. This iteration is sometimes referred to as “flow up and down” (/MDSD/).

Top down we distinguish a conceptual architecture (consisting of a functional architecture and a logical architecture), which is independent of technical implementation details, and a technical architecture, where we are considering the influence of the given hardware or other domain specific implementation decisions.

The functional architecture describes the system’s behavior from the black-box-perspective (cf. User Function in 2.4.1). It offers models, which allow for a formalization of functional requirements, representing them in form of hierarchical structures and additionally illustrating dependencies between these functional requirements. The functional architecture therefore provides the basis to detect unwanted interactions between functions at an early stage of the development process. Thanks to this basis and its high level of abstraction, the functional architecture is also well suited for an extension by product line concepts.

The logical architecture describes the system’s behavior from the white-box-perspective. It offers models, which allow for a structuring of functionality into specialist-specific components. The functional requirements formalized in the functional architecture are realized by a network of hierarchical logical components, which are, however, independent from the underlying hardware (cf. Logical Component and Behavior in 2.4.1). The model of the system on the layer of logical architecture is executable and can be simulated. It is thus amenable to an earlier architectural verification. Due to the modular design and this layer’s independence from hardware, the complexity of the model is reduced and a high potential for reuse is created.

From functional to logical:

- In a black box view the internal structure of a system is neglected and only its interaction over its system boundary to its context is considered (cf. User Function in 2.4.1); this view is also called the interface of the system to its context. Depending on the form of interaction the interface of a system to its context can be separated into a static and dynamic aspect. In the static aspect it is described which events and values of interaction can occur in principle (cf. Signature in 2.4.1). In the dynamic aspects the system’s interface behavior (cf. Functional Decomposition in 2.4.1) is described which shows the causal relationship between the sequences of actions provided by the actors of the context and the sequences of actions being the reactions of the system as exchanged and observed on the system’s boundaries.
- In a white box view the internal structure of a system is described (cf. Logical Component and Behavior in 2.4.1). This internal structure can consist of a family of logical components or of a state machine description. The components interact. Depending on the form of interaction the interface of the components can be separated into a static and dynamic aspect. In the static aspect it is described which events and values of interaction can occur in principle. In the dynamic aspects the components’ mutual behaviors are described which shows the causal relationship between the sequences of actions provided by the components as exchanged and observed including those on the system’s boundaries.

The black box view is an abstraction of the white box view called interface abstraction.

These two views indicate that on one side we are interested to understand what is going on beyond the system boundaries, i.e. what the effects of the systems with respect to its environment are, and on the other side how the system is internally structured, i.e. how the system behavior is reflected in terms of the internal system behavior.

When the involvement with the system becomes deeper additional views from predefined viewpoints are helpful to reason or communicate about the system.

The technical architecture finally describes the realization consisting of physical components and their behavior in an abstract way. It offers suitable models which describe the behavior of hardware and software uniformly (cf. Cluster in 2.4.1) and which allow to describe the influence of the hardware used on the behavior of the entire system. Here, the level of abstraction is chosen that way as to make it possible to conclude whether real-time requirements can be met and that at the further transition from technical

architecture to implementation only software-technical transformations (e.g. middle-ware calls) but no modifications of the behavior will take place.

All these abstraction layers can then be partitioned further using proven domain-specific structuring techniques of the different engineering disciplines mechanics, electrics/electronics and software as outlined in figure 4.

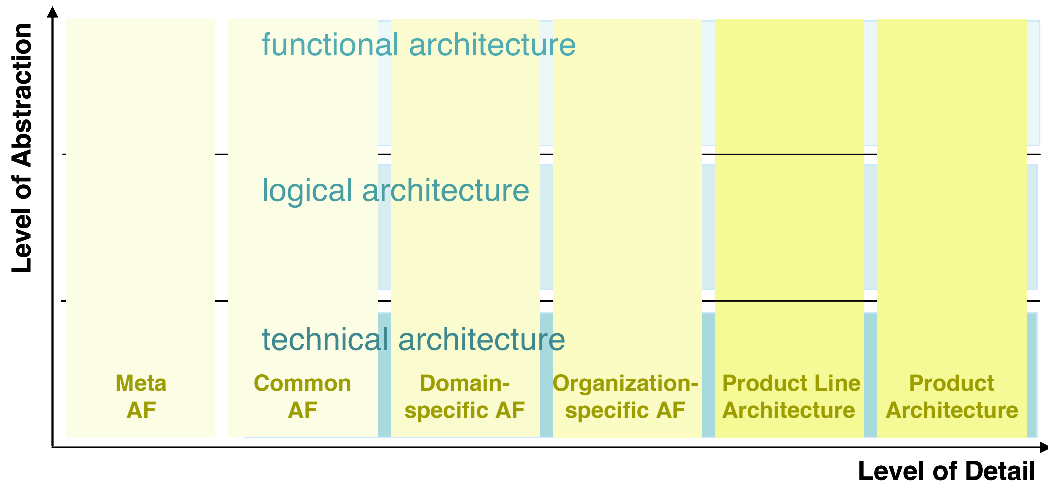

Figure 4: Level of Abstraction and Level of Detail

2.5 Domain-specific Architecture Framework (dAF) for the Automotive Domain

2.5.1 Overview

A domain-specific architecture framework (dAF) is defined as the foundational framework providing a common denominator (terminology, structure, methods, architecture models, guidance and rules) for developing, representing, understanding, and comparing domain-specific product architectures across a (virtual) development organization (i.e. Automotive value net). It provides insight for external stakeholders into how a specific lead organization within this domain or industry (e.g. Automotive OEM) develops products and product architectures.

The intention of the dAF is to ensure that architecture descriptions can be compared, related and re-used across different vehicle programs, development units and organizations, thus establishing the basis for

- increase in quality and overall value creation efficiency,
- risk reduction, and ultimately,
- increased innovation

within the domain or industry of interest.

Applying the concepts described on the meta / generic level (the lower two levels in figure 5: The AAF within the Architecture Taxonomy) from the perspective of the Automotive industry leads to the creation of an architecture framework which is specific for the Automotive industry: The Automotive Architecture Framework.

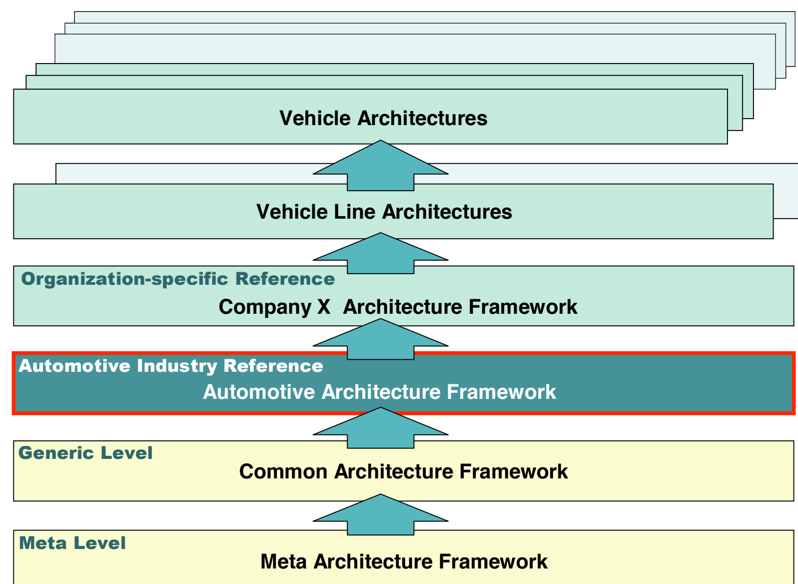

Figure 5: The AAF within the Architecture Taxonomy

The Automotive Architecture Framework suggested here can be understood as an instantiation of emerging architecture standards. It is intended to serve as a reference for OEM specific architecture frameworks which again serve as a reference for vehicle line and specific vehicle architectures (figure 5: The AAF within the Architecture Taxonomy). The AAF – once complete – will consist of the following elements:

- An Automotive specific, uniform way of describing, structuring and comprehensively modeling system architecture including the specification of predefined levels of abstraction, view points and views.
- Suggestions for families of artifacts that document the architecture and the methods and activities of working it out:
  - Modelling methods for architectures and their properties.
  - Meta language and meta models for describing the structure and the parts of an architecture description. See also chapter 3.4.
  - Pragmatic and methodological facets of architectures including principles, rules and best practices. See also chapter 3.2.
  - A general terminology with precise definitions of the relevant notions to be used to describe, discuss, and evaluate architectures (key terms are described in this document)
- Methods and approaches to assess and evaluate architectures (Not elaborated in this paper. For further information please refer to /Clements at al. 2002/).

- A clear understanding and definition of the function and role of architecture in the development process. Chapter 3.2 et. seq.

The general goal of an architecture framework is to provide a basis for a common understanding of architecture and also a detailed structuring that can be used as a basis for tooling supporting architecture design, architecture artifact management and the application of architecture in system development.

It should be noted that this paper presents just the concept, respectively a first draft of a future AAF which – clearly ! – needs to be agreed upon, and enriched by major stakeholders of the Automotive industry.

2.5.2 AAF Viewpoints

Development of modern vehicles has become a highly complex task. One reason for this is that vehicles are complex systems that have to fulfill numerous, often contradicting requirements. Another reason is the complexity of today's Automotive ecosystem spanning the entire breadth of value creation disciplines from mechanical engineering to electrics, electronics, software, services and branding.

These complexities cannot be easily addressed using a single view or in a single framework. Rather vehicle architectures must be described by multiple architectural views which are developed from multiple viewpoints, each of the latter focusing on different aspects of the overall problem.

As described earlier, viewpoints can be thought of as a perspective on a vehicle that defines the objects and rules for constructing views and that permits only a subset of objects and representations relevant for a given concern to be analyzed (/RASDS/). It is obvious that different viewpoints have relations or "correspondences" to other viewpoints.

This AAF suggests two sets of architectural viewpoints: Viewpoints which we consider to be general since they are independent of specific product strategies, and viewpoints which are "optional" since they reflect specific OEM focus.

Regarding the general viewpoints the attempt has been made to keep these close to the ones already proven in other manufacturing industries (e.g. space industry: (/RASDS/)) as well as related standardization initiatives (e.g. ISO/IEC & IEEE, RM-ODP[2]).

The mandatory viewpoints suggested are:

Functional Viewpoint:

The functional viewpoint looks at vehicles from the perspective of vehicle functions and their logical interactions. Functional viewpoints have been adopted since long in all engineering disciplines. Although it may have been used more implicitly in the mechanical domain, it has been used explicitly in the electrics, electronics and software domain. Examples for related methodologies are Petri nets, parallel path expressions, ladder logics, etc.

A resulting functional view describes the functional composition of a vehicle, its functional entities, interfaces, interactions, interdependencies, behavior and constraints.

Technical Viewpoint:

From the technical viewpoint we look at a vehicle from the perspective of its physical components (including electronic and electrical hardware), their geometry and composition within superordinate geometric structures, their relationships, behavior (including physical aspects like thermodynamics, acoustics, vibrations, mechanical deformation, …), dependencies and constraints.

The resulting technical view describes the implementation of vehicle functions from a physical / technical perspective.

It is obvious that the technical viewpoint has strong correspondences with the functional viewpoint as well as optional viewpoints like for example "energy".

---

[2] Cf. www.iso.org, www.ieee.org, and www.rm-odp.net

Information Viewpoint:

The information viewpoint looks at vehicles from the perspective of information or data objects used to define and manage a vehicle. This includes the description of mechanisms, protocols, and standards that support information transfer between vehicle subsystems.

The resulting information view describes these objects, their metadata, properties, relationships, and configurations, as well as configuration constraints.

Driver / Vehicle Operations Viewpoint:

From this viewpoint we look at the vehicle from the perspective of the vehicle driver and the world around the vehicle.

The resulting view describes interactions, interfaces, interdependencies between vehicle and the driver (and the passengers), as well as the surrounding environment respectively. In addition to that it describes the related behavior, constraints, and priorities. The surrounding environment may include road, road side, other vehicles, Telematics and traffic control systems, etc.

Value Net Viewpoint:

From this viewpoint we look at the activities of, and the dependencies between the stakeholders of the end-to-end value creation process which happens within a specific value net. A value net is defined as a virtual organization (typically around an OEM) consisting of the OEM, its suppliers and engineering partners and other constituents involved in the process of creating customer value).

The resulting value net view describes major activities, roles, relationships between the roles, policies and agreements between stakeholders, etc. with the goal to optimize the efficiency of the value creation process. It can also be used to align the (value creation) strategies of the stakeholders which form the respective value net.

In addition to the general (or mandatory) viewpoints the AAF suggests a number of optional viewpoints which will reflect specific OEM concerns which cut across (are orthogonal to) the mandatory architectural views proposed above. Optional viewpoints suggested for the AAF are:

- safety
- security
- quality and RAS (reliability, availability, serviceability)
- energy (possibly including performance)
- cost
- NVH (noise, vibration, harshness)
- weight

and others which may be of specific importance for a specific OEM.

Together with the general viewpoints specific optional viewpoints which are in line with a given OEM's priorities are used to create OEM specific architectural views. Thus specific OEM architecture frameworks (as shown in fig. 5) are created which finally lead to different vehicle architectures which nevertheless all follow the same general principles.

The advantage of this effect is a direct one for participants (i.e. suppliers) in the Automotive ecosystem which serve more than one OEM: They do not have to deal with numerous architectures which are totally different to each other.

The OEMs will benefit from an indirect advantage which is the result of an increased value net efficiency delivered by those ecosystem players who enjoy the direct advantage. Another direct effect for the OEM is a better efficiency in describing systems consistently, which has a direct impact on quality, time and cost of the development process.

2.5.3 AAF Levels of Abstraction and Detail

Levels of abstraction allow us to model vehicles as appropriate where each level represents a different model of the same system, involving a unique set of components and compositions that are applicable only to a particular view that abstracts from certain details of implementation.

The functional architecture follows a black box viewpoint and models the system's functionality.

The logical architecture models the decomposition of the system in interacting logical components, which are describing the functionality of the whole system or parts of it (subsystem) in a formal and executable manner.

The technical architecture aims at the implementation viewpoint and exhibits all parts of the technical system such as hardware, software, and mechanics.

The functional view is implicitly contained in the logical architecture, and the logical architecture is implicitly contained in the technical architecture. Vice versa, the functional view is an abstraction of the logical architecture, and the logical architecture is an abstraction of the technical architecture. The AAF is the lowest level of detail and provides the guidance for more specific architectures up to architectural elements.

For each level of abstraction again more abstract or more detailed models exist. For instance, we can choose finer granularities of time or of interaction to model the functional and the logical views.

Some examples for further detailing the technical architecture level are given in figure 6.

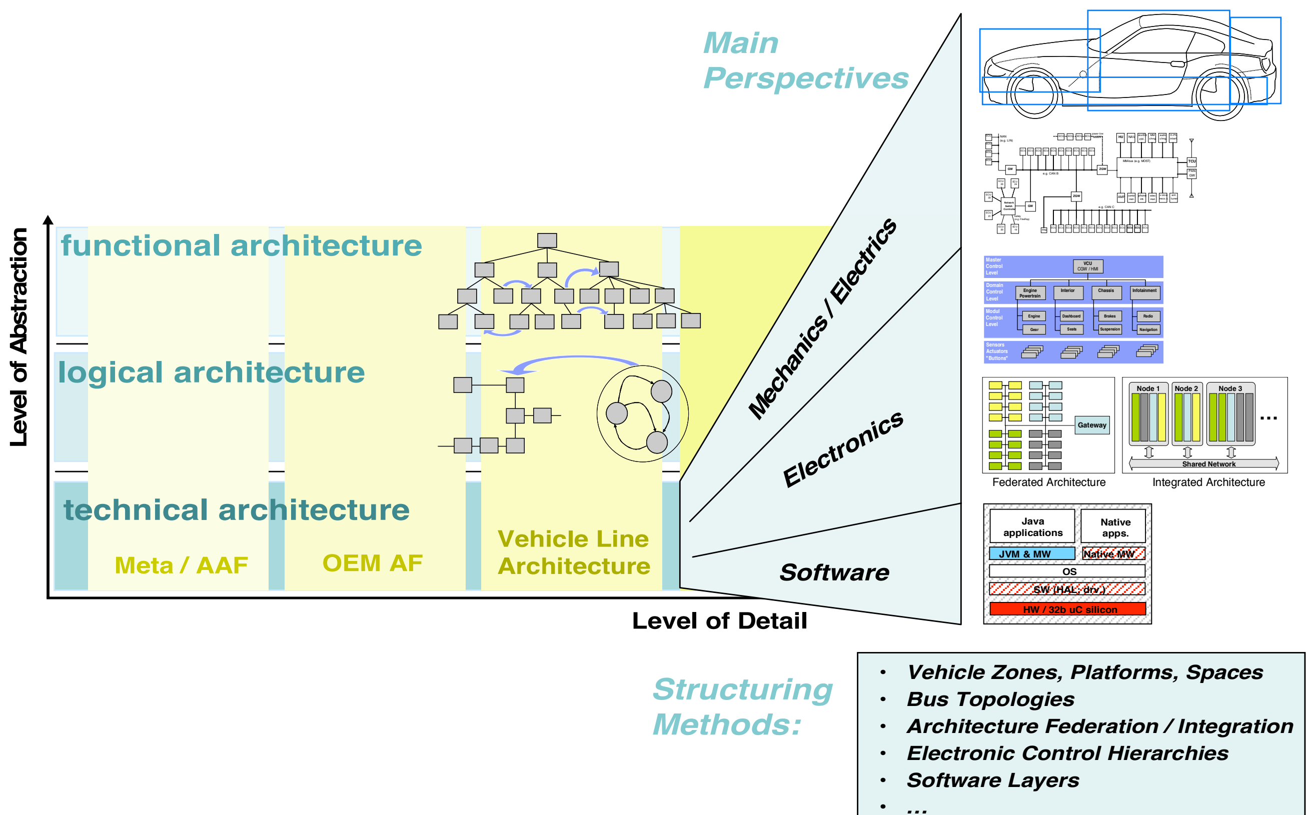

Figure 6: Levels of Detail for the Technical Architecture Level of Abstraction

2.6 Open-MODELS.org – A platform for common sense Architecture Frameworks

We have shown that elaborating an architecture framework leads to establishing multiple layers of architecture frameworks. These layers are justified by different levels of generality. The common Architecture Framework unifies all the cross-domain modeling aspects which are relevant, independent of the concrete domain of the system to model. Several domain-specific Architecture Frameworks again refine this common Architecture Framework but are still independent of a concrete company. This whole story makes it necessary that both the common AF as well as the domain-specific AFs have to be established as a common sense in the industry.

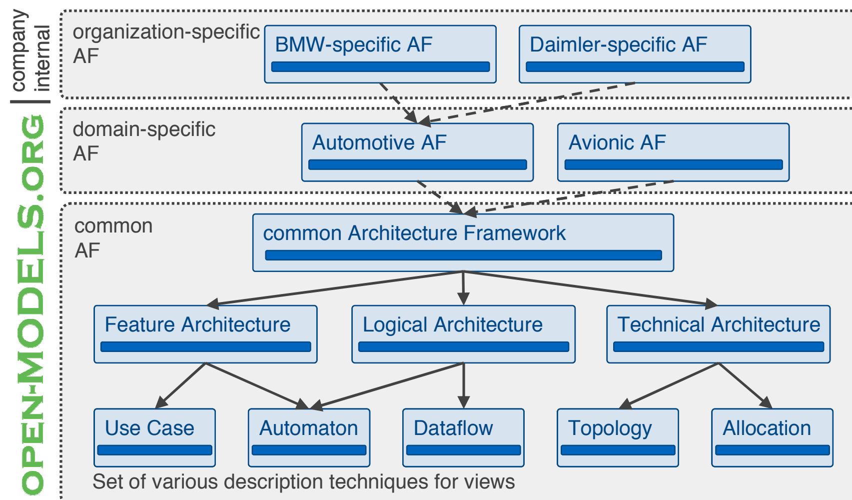

Figure 7: Interoperation between different AFs and description techniques

In the long run Architecture Frameworks consists of extremely large and detailed definitions and specifications. In order to handle such a huge specification and standardization process a common platform for developing Architecture Frameworks themselves has to be established. Therefore the Technische Universität München together with OOMEGA (/OOMEGA/) initiates the platform open-MODELS.org (/OpenMODELS/), which proposes a public accessible metamodel repository in particular for Architecture Frameworks. “Open-Model” provides an integration platform. For instance the AUTOSAR metamodel could be placed in “Open-Models” and related to the AAF meta model.

As shown in figure 7, the common platform open-MODELS.org should collect and provide a set of various description techniques like automata and dataflow diagrams. Based on these description techniques as well as the meta AF (which should also be defined at the platform) a common AF is defined. This will be the basis for all the domain-specific AFs. Up to this level all the Architecture Frameworks should be common sense and thus are specified at open-MODELS.org. Based on this knowledge every company can customize their own company-specific Architectural Model. The grade of how much it differs from the standardized domain-specific AF depends in a flexible way on internal structures and the strategy of the company. The initial drafts for the metaAF and commonAF as described in this paper are already available on this platform.

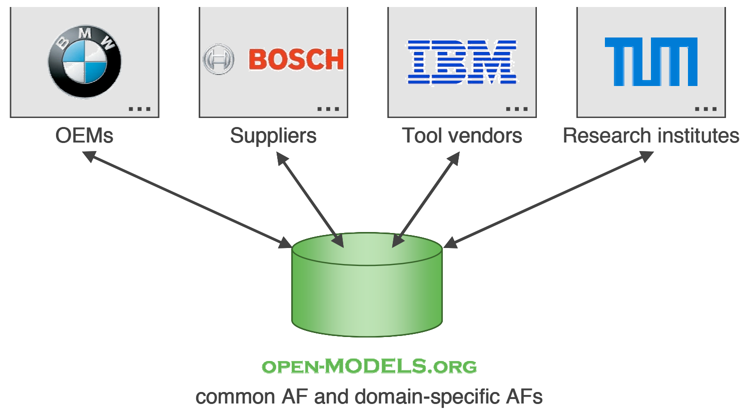

Figure 8: AAF and other meta models shared between different stakeholders

A single company should not and could not force others to take over a common AF. The different stakeholders should work together instead. Only if all the stakeholders share their knowledge, coordinated by a common platform, both a suitable and accepted Architecture Framework can be established. As shown in figure 8, the major types of stakeholders in the automotive industry are OEMs, suppliers, tool vendors and research institutes.

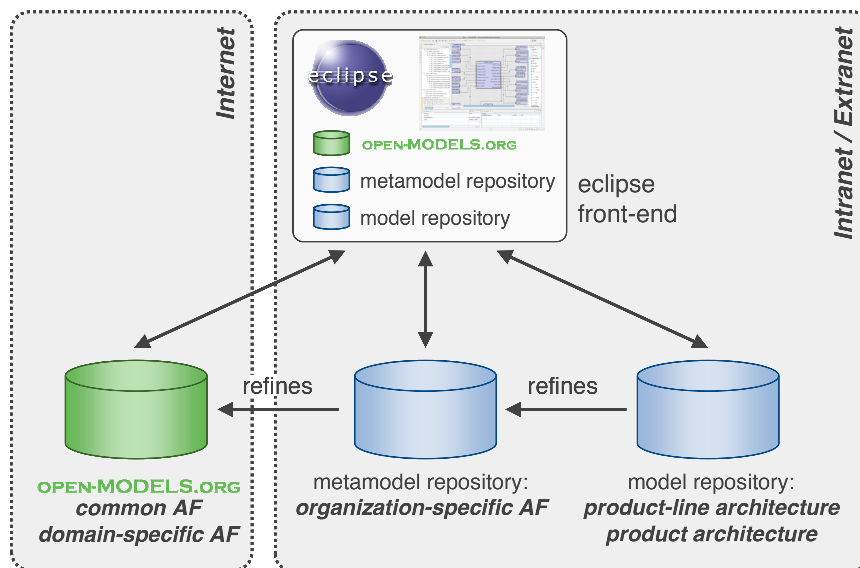

Figure 9: The principle usage

The principle usage of open-MODELS.org is shown in figure 9: Via an Eclipse-plugin the open-MODELS.org repository can be accessed. For the company specific AF an own company-internal repository is established. The models for both products and product lines are stored in a third repository and should be accessible via all the company's system-engineering tools. Such a model repository will also allow a much more integrated and seamless development process in the future. The recently introduced collaboration platform Jazz could be a perfect platform for such a scenario (/Jazz/).

3 Architecture from a Development Perspective

Architecture is indispensable for a systematic development. It is governing the separation of concerns and responsibilities. In Chapter 2.5.1 the linkage of architecture driven concepts with the necessary development methods and process activities was already addressed in the context of an AAF. This chapter will elaborate further on that relation.

3.1 Intro – reasoning for interaction of methodology and process

The concept of introducing an architecture framework like the AAF has a large impact on the specific methodology and process to develop architecture driven systems as already outlined in chapter 2.5.1. Best practices use a number of iterative development phases and also use model transformations to travel from a certain architecture representation state at a higher abstraction level towards the next level of detail on subsequent definition layers. To enable and manage this approach, product development needs to be governed by a capable process as well as an applicable methodology to achieve an efficient implementation.

In particular, we have a workflow relationship to the architecture model as well as a relation between development steps and certain parts of the architecture. In addition, the architecture and its definition layers play an essential role in guiding the creation of specific design artifacts in the development methodology, controlled by the process execution and project management oversight.

3.2 Using a model driven approach

In order to apply the architectural development concept within the relationship space of methodology, process and artifacts, the most efficient way to implement this in an integrated fashion is by an iterative model driven approach.

The classical, mostly document driven, product development has several setbacks often using decomposition of requirements into specifications rather than decomposing systems to subsystems and deriving requirements for each substructure. This usually causes the following problems:

- Various parts of the organization decompose requirements from various origins and write specifications from their own viewpoint.
- Levels of requirements are not clear, thus confused.
- Specifications may be inconsistent.
- The process is paper-based or single file based at best - no one place to look for the answer.
- The content in different areas tends to diverge and it is not easy to get it back into alignment
- There is no clear exit criteria to determine when the design work is complete

The following aspects show why the interaction of model based methodology and process is necessary:

1. **System requirements are not complete nor static at the time of design to start**

   In today's (and future) innovative system concepts it is quite unrealistic to assume or expect all requirements to be present and complete at a beginning of a design project. This is especially true for software centric systems or complex mechanical systems.

   The methods as well as the process (which is derived from the methodology chosen) need to provide for the appropriate iterative steps to introduce new requirements or adapt existing requirements to the new system needs. Due to the fact that requirements are not completely stable along the lifecycle and the dynamic change of relations between requirements and developed artifacts, it is hard to keep control without a traceable iterative way of linking requirements to a layered structure of a system

model. Having that allows capsulation and an isolated view on sets of requirements and the impact of a change to the related system structure.

As the detailing of the architectural layers increases, the amount of new requirements allowed to flow in has to be regulated by the project management, using the guiding principles of performance, cost and time. The development process used has to cover this limitation as well, by (dynamically) defining the formal boundaries that need to be taken before requirements can be changed or added at any phase in the process.

2. **Model based engineering allows to stay in control of the system complexity**

   Models allow limitation of complexity through layers of abstraction. At higher abstraction levels the models can be shared with stakeholders with sufficient amount of detail to discuss concepts and ideas. At lower levels the concepts get enriched to a state where implementation and creation of final product can be realized. The architecture approach keeps the layers in context and prevents loosing sight of important and critical aspects of the entire system.

3. **Functional representations of a system need to allow early validation of the logical architecture layer**

   Early validation at that level is only feasible using system models and simulation of these models. Also, the application of formal methods to ensure consistency during decomposition and functional enrichment in the various design phases needs to be supported by the modeling environment. This modeling approach ensures the correct implementation of functions and the architectural structure prior to the physical implementation. Physical prototypes or samples are too late in the process to capture any logical or structural design flaws, due to their cost and timing implications.

   The development process needs to drive both the conceptional modeling and creation of related design artifacts as well as the appropriate validation schemes and its execution early enough in the lifecycle so that effective implementation decisions can be made.

4. **Reuse is part of the architectural concept**

   Architectures can also define certain standardization areas, like interfaces, functional components, protocol structures etc. If the architecture can be kept stable over a certain time period, then existing solution components can be re-used more easily. As a product strategy, architectures can be planned and setup in a way, that they stimulate and foster re-use as an immanent principle, which has a significant influence on development cost, product quality, and time to market. Model based methods support this approach very well.

3.3 The Role of Architecture in the Model Driven System Development Process

Explaining a certain methodology in detail is not in the scope of this article. There are several publications already on this topic (/MDSD/, /Harmony/, /Harmony SE/).

Yet the following tries to relate architectural aspects to the model driven systems development (MDSD) process steps to show the importance of an architecture framework to more simplify the development lifecycle and to reduce risk.

3.3.1 How does the architecture interact with the creation of design artifacts

The following shows an outline of the MDSD approach in several layers of abstraction, very similar to the architecture layers discussed in chapter 2.4.2. In fact the various layers of the system decomposition can be supported and fed with components from an architecture framework as shown in figure 10: Outline of the model driven systems development (MDSD) method with architecture relation in order to decompose the system, as well as to derive requirements and to write specifications for the system and each subsystem in its own context.

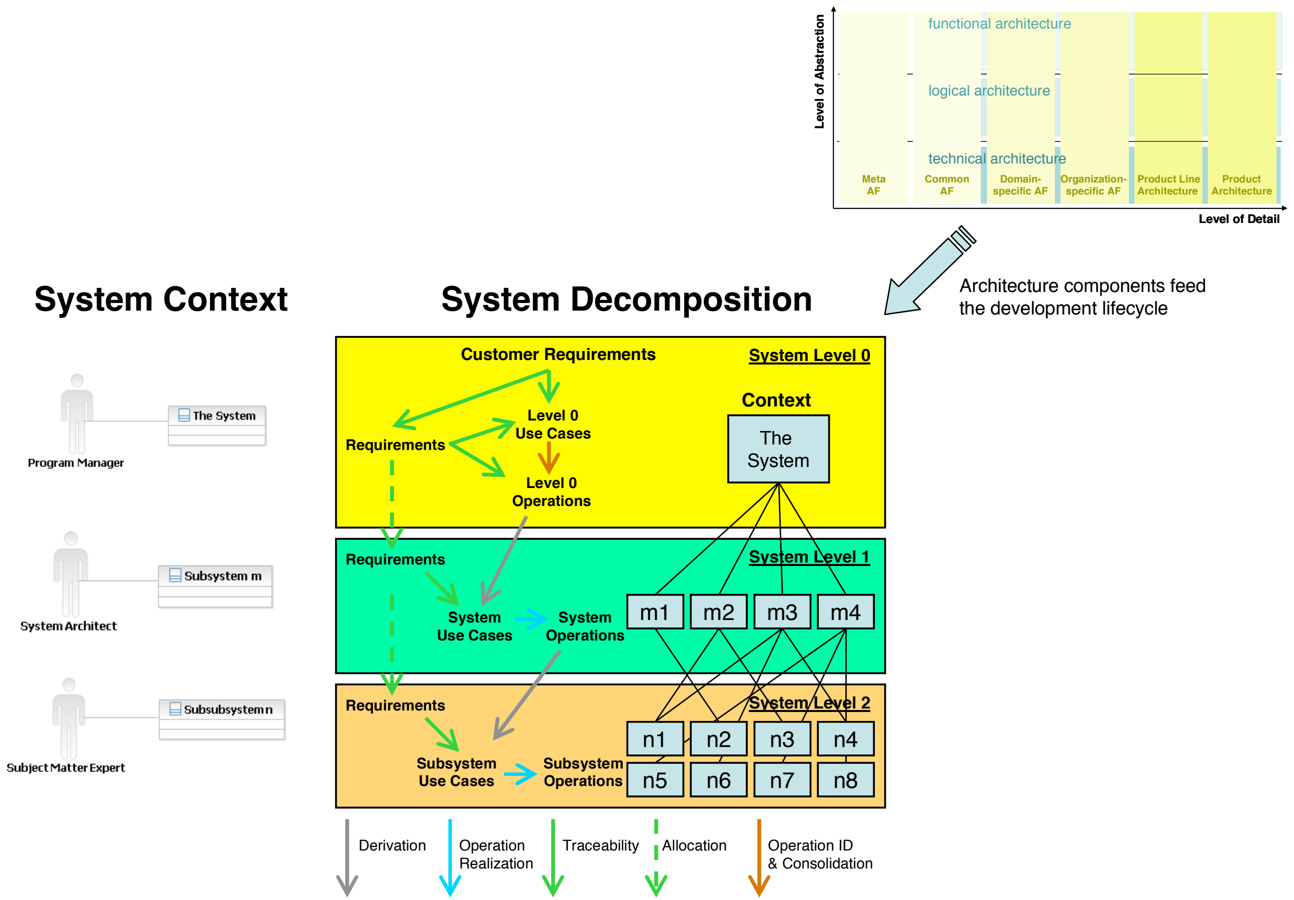

Figure 10: Outline of the model driven systems development (MDSD) Method with architecture relation

3.3.2 Outline of the Software Engineering methodology

The following steps are forming the model driven capability pattern, showing the relation to the intrinsic architecture creation. (/MDSD/)

1. **Gather Source Requirements**
   Understand the stakeholder's needs, collect and prioritize requests on the system functionality. Define the overall vision of the system, including the problem to be solved, the scope/boundary of the system, key features (requirements) and constraints as much as known. This gets enriched during the development lifecycle through the iterative phases.
2. **Establish Systems Context**
   Generate an initial top level view (mostly use case model) showing the system, its interfaces, and its relationships with its actors. Also define and describe the external I/O entities (i.e. data) that flow between actors and the system.
3. **Refine the System Definition**
   Develop supplementary requirements describing the system (that do not apply to specific use cases), define and manage relationships and tracebility between requirements, and further detail the system description.
4. **Synthesize Systems Architecture**
   Construct and assess a System Architecture Proof-of-Concept, showing the feasibility of defining a solution which will satisfy the architecturally significant requirements, to proof that the system, as envisioned, can be constructed.
5. **Define Candidate System Architecture**
   An initial representational concept model of the target system architecture is created during this phase.
6. **Analyze System Behavior**
   Transform the behavioral descriptions provided by the requirements into a set of elements or components which define the basis for the detailed system design.
7. **Refine System Architecture**
   Keep the system architecture in synch with the changes arising during the various design iterations due to requirement modifications or design detail enrichment, design refactoring and reusability issues.

3.4 Applying model driven architecture (MDA) concepts

In a model driven approach the architectural (meta-) components are used as a basis for system structures that fulfill a certain functional portfolio. If we take the principles of MDA (Model Driven Architecture) from the SW domain into the systems engineering domain we view the initial functional model as a platform independent model (PIM) that gets transformed into the logical model using predefined architectural rules and (meta-) components as the platform model (PM). The result would be the next layer of platform specific model (PSM) which could again serve as the basis for the next transformation using a set of derived requirements and new rules applicable for that specific detail layer. The MDSD process outlined before provides a supporting methodology framework to include the architecture centric concept.

As the content on each detailing layer gets enriched and completed iteratively, further transformations take place to include the modifications made during the design process. The modelling techniques as well as the tool chain have to ensure the consistency and traceability within the data representing the system.

It is also mandatory to have closed and automated process governance within the development environment to define and enforce the usage of architectural components within the design.

On a higher maturity level the architecture framework development is a continuous background task and a capable process is needed to connect the architecture development stream and its evolution with the actual product development activities for specific platforms and its generations.

4 Goals and Requirements as Architectural Drivers in Viewpoints

In this section, we sketch how our notion of viewpoint in the presented concept of architecture framework methodologically enables us the integration between requirements engineering and architecture-oriented systems engineering.

The role of architecture in the development – i.e. analysis, design, implementation – of a product also implies its role in the activities of decision making, planning and organisation /Sangwan, Neill 2007/. These topics give account to the terms of goal, requirement and architectural driver. We consider goals as abstract requirements in the sense that they characterise results to achieve at a higher level. Architectural drivers are requirements which have been determined and decided to cause substantial impact to the structure or design of an architecture (/Wojcik et al. 2006: Section 6.1, page 15/). Architectural drivers maybe specified at a higher, goal-based level similar to concerns or they maybe specified as non-functional system requirements at a lower level. In each case a corresponding metric should be provided to evaluate development artifacts during certain steps of quality assurance w.r.t. to the intention of the architectural driver.

A viewpoint's concerns as motivation for architectural drivers. Each viewpoint has its specific concerns, maybe a single one only. Concerns may either refer to functional or non-functional quality attributes. Most often they characterise non-functional qualities. To make the intentions behind a concern more tangible, we use the term concern as purpose, vague goal or end of a viewpoint and, in the context of requirements engineering, as a driver of goal refinement and requirements specification during the requirements analysis phase.

Requirements as a viewpoint's content and architectural drivers. Elicited goals and requirements can be classified by the concerns they refine and the scopes they belong to in order to represent the content (view) of an early viewpoint. Now, we may select a set of goals or requirements, whose refinement was motivated by the viewpoints' concern, as drivers of architecture, its analysis and design at the abstraction layers of functional, logical and technical architecture. A concern about functionality may be used for selecting refined goals and requirements for the functional architecture view. A concern referring to a non-functional quality, e.g. maintainability, cost, reliability or performance, may be used for classifying goals and requirements and may drive design decisions for the three mentioned architecture layers or a specific cost view.

Concerns, goals and requirements often lack a measurable characterisation of the regarded quality attribute. To cope with that issue and for precise evaluation and decision support, we can define (architecture) metrics to quantify these statements where applicable and necessary. After having defined such metrics (cf. Fig. 2 “meta AF” in Section 2.3) design decisions can take place and the selection of the relevant means – viz. architectural characteristics like platform strategies and practices, design patterns, implementation tactics etc. – can be carried through in a controllable manner. During the application of the architecture framework to a specific product architecture, we aim to bridge the gap between planning and design by attaching means to the concerns of the determined architectural drivers. Once more, we may use the concern and scope of a viewpoint to restrict the kind and amount of analysis information contained in the viewpoint's corresponding view. The specific usage of means should then be discussed in detail in a view's description.

Finally, an architecture framework strengthens its role as a documentation, communication and management instrument for product managers, requirements analysts, system architects and designers. The notion of viewpoint discussed in Section 2.3 and, especially, the aforementioned conceptualization of architectural drivers as refined viewpoint concerns constitutes a good methodological starting point for

- goal- and requirements-based assessments at the abstraction layer of functional and logical architecture, as well as
- subsequent choices of technology and deployment strategies at the abstraction layer of technical architecture.

By integrating the three notions of goal, requirement and architectural driver with the aforementioned definition of architecture framework and its central concepts of viewpoint and view, we contribute to bridging the gap between requirements engineering and architecture design.

5 Summary and Outlook

Complexity of vehicles has grown to levels which hardly can be handled in an efficient manner with today's development practices. Yet another increase in complexity can be seen with the trend to new drive technologies (e.g. hybrids, electric drives). Consensus is growing that the introduction of the architecture paradigm will help to tackle the complexity problem.

In fact, a number of architecture initiatives have been started – AUTOSAR being the most prominent example – with basically all of them concentrating on rather deep technical levels (e.g. SW abstraction layers, communication protocols, data formats).

What is missing is an overall architectural concept which satisfies the following needs:

- it is defined from a total vehicle perspective
- it provides an umbrella for behavioral systems description as a common layer across all functional domains to maintain functional traceability
- it is defined from a strong development-methodological background
- it provides the context for more detailed (technical) architectural work
- it spans the entire Automotive industry reflecting both the interdependencies between the stakeholders (OEMs, suppliers, engineering partners, tool vendors, …), and the necessities coming with the requirements of the vehicle life cycle

An architecture framework for the Automotive industry can be an appropriate means to help introduce the architecture paradigm in a more effective way to Automotive product development leading to more efficient value creation in this industry.

This paper laid out the concept and some basic elements of such an architecture framework. The intention was to drive the thought process within the Automotive industry towards the requirements stated above having successful examples of similar foundational work in mind (e.g. DoDAF).

The authors of this paper are well aware of the immature nature of this document and the content of the AAF as it is outlined here. They do hope however that this paper will help to trigger an industry wide initiative which discusses this concept and – eventually – agrees on some common structures, terminologies, and methodologies which will facilitate the exploration of the power of the architecture paradigm for the benefit of future Automotive vehicle development.

6 Bibliography

/AUTOSAR/ www.AUTOSAR.org

/Broy et al. 2008/ Manfred Broy, Martin Feilkas, Johannes Grünbauer, Alexander Gruler, Alexander Harhurin, Judith Hartmann, Birgit Penzenstadler, Bernhard Schätz und Doris Wild, *Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme*, Technischer Bericht TUM-I0816, Technische Universität München, 2008.

/Clements et al. 2002/ Evaluating Software Architectures – Methods and Case Studies" from Paul Clements, Rick Kazman, Mark Klein, Addison-Wesley, 2002

/DoDAF/ US Department of Defense (DoD), *DoD Architecture Framework* version 1.5, 2007

/Harmony/ The Harmony Process, Whitepaper, Bruce Powell Douglass, 2007

/Harmony SE/ Harmony for SE Deskbook, Hans-Peter Hoffmann, Rev. 1.51 Telelogic AB 2006, Rev. 3.0 IBM Rational 2009,

/IEEE1471/ IEEE Std 1471-2000: *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 2000.

/ISO42010/ ISO/IEC working draft Std WD3 42010: *ISO/IEC Systems and Software Engineering - Architectural description,* 2008.

/Jazz/ www.jazz.net

/MDSD 1/ Nolan, Brown, Balmelli, Bohn & Wahli, *Model Driven Systems Development With Rational Products*, IBM Redbook Publications, SG24-7368-00, 2008.

/MDSD 2/ Balmelli, Brown, Cantor, Mott, *Model Driven Systems Development*, IBM Systems Journal Vol. 45, 2006

/RASDS/ The consultative committee for space data systems CCSDS: Reference Architecture For Space Data Systems, Recommended Practice CCSDS 311.0-M-1, 2008

/Sangwan, Neill 2007/ Sangwan, R. S.; Neill, C. J., *How business goals drive architectural design,* IEEE Computer, 40, 8, pp. 85-7, 2007.

/Sommerville et al. 1998/ Ian Sommerville, Pete Sawyer, Steve Viller, *Viewpoints for Requirements Elicitation: A practical Approach,* in: Proceedings of the Third IEEE International Conference on Requirements Engineering (ICRE'98), pp. 74-81, 1998.

/OpenMODELS/ http://www.open-models.org: Platform for Collaborative Language Engineering

/OOMEGA/ http://www.oomega.net: Framework for Model-based Software Engineering

/Wojcik et al. 2006/ Wojcik, R.; Bachmann, F.; Bass, L.; Clements, P.; Merson, P.; Nord, R.; Wood, B., *Attribute-Driven Design (ADD)*, Version 2.0, Defense Technical Information Center, 2006.

Acknowledgement

We would like to thank our IBM colleagues, Hans Windpassinger and Dr. Heinz Rybak, who deserve special mention with regard to their technical advice and review.

P. Kluge and W. Krenzer.