Architecture Potential Analysis: A Closer Look inside Architecture Evaluation

Bastian Florentz, Michaela Huhn
Institute for Programming and Reactive Systems, Technical University at Brunswick, Germany
Email: {florentz,huhn}@ips.cs.tu-bs.de

***Abstract*— The share of software in embedded systems has been growing permanently in the recent years. Thus, software architecture as well as its evaluation have become important parts of the development of embedded systems to describe, assess, and assure sound architecture as basis for high quality systems. Furthermore, design space exploration can be based on architecture evaluation. To achieve an efficient exploration process, architectural decisions need to be taken into account as part of the architecture. In this paper, a method for analyzing architecture potential on the basis of dependencies between quality attributes is presented and applied. An explicit representation and correlation of such dependencies provides decision support for architectural concerns. Not only can suboptimal decisions be avoided but rather valuable options are highlighted. Besides the quality of an architecture, knowledge of how to achieve and even improve the quality can be analyzed. The latter is the concern of architecture potential analysis presented in this paper. Furthermore, architectural decisions can be documented and will be traceable and justifiable with respect to the development rationale. The ongoing development process can then be based on dependable and well documented architectural decisions. The predictability of change impacts is increased. Thus, time and costs can be saved by avoiding suboptimal changes.**

***Index Terms*— Embedded Systems, Architecture, Evaluation, Analysis, Design Space Exploration**

I. Introduction

High quality and low development effort are two superior goals for development processes of modern software-intensive systems. Not just model-based development processes but also explicitly underlying architectures are the upcoming way to achieve these superior goals. While high quality can be attributed to the system architecture itself, low development effort is mainly based on sound architectural decisions. Especially software-intensive high-quality systems with short innovation cycles can highly profit from time- and cost-efficient system development. Thus, not just the established components but their composition as well, defined by an adequate architecture, are important parts of the development process. To assure the quality of architecture, evaluation has to be performed and documented to identify the most promising variants and to profit from experiences made during the development process. These experiences in combination with results of sensitivity analysis build the basis for understanding architecture evaluation even in case of being a non-expert regarding some of the quality attributes contained in the evaluation. Furthermore, the dependencies in the architecture evaluation can be used to guide changes in architectures, i.e. applying architecture potential analysis.

Industrial projects (e.g. Florentz [1]) have shown that the superior goals can be only reached if architecture documentation and the communication of architectural decisions are supported. Both are provided by the explicit representation of dependencies in the architecture evaluation. In this paper, architecture potential analysis is presented to identify interesting dependencies and correlate them to get a closer look inside architecture evaluation.

First, a model for representing architecture, its elements, and their relevant properties will be defined. This model is motivated by the automotive domain and contains architecture views on software, hardware, and the mappings of software as well as communication to hardware (cf. AUTOSAR, Heinecke et al. [2], and EAST-EAA, Debruyne et al. [3]). It is based on the component-and-connector architecture viewtype (see Clements et al. [4]), which allows for application of component-based approaches besides a concise description of the system structure.

Second, the quality of architecture variants has to be evaluated. Therefore, quality attributes have to be defined to represent architectural requirements and evaluate architectures with respect to those requirements (cf. Clements et al. [5]). The compliance of architecture variants to quality attributes can be evaluated by certain techniques whose results have to be interpreted to a quality rate depicting the fulfillment of the architectural goals. A model for defining architecture goals based on quality attributes is described in Section VI. The structure of quality attributes is called QADAG (Quality Attribute Directed Acyclic Graph). The structure is the entry point for the architecture potential analysis. Furthermore, it allows for discussing the rationale of the architecture development.

Several approaches deal with the evaluation of software architecture. Ali Babar and Gorton [6], Ali Babar et al. [7], Bergner et al. [8], Dobrica and Niemelä [9], Ionita et al. [10], and Grunske [11]) have compared the different approaches in surveys. According to Abowd et al. [12], architecture evaluation methods are classified

Portions reprinted, with permission, from “Inside Architecture Evaluation: Analysis and Representation of Optimization Potential,” by Bastian Florentz, which appeared in the Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA), Mumbai, India, January 2007. © 2007 IEEE.

into qualitative, quantitative, and hybrid methods. Besides the integration of several architecture evaluation methods or techniques and the comparison of several architecture variants, the approach presented in this paper aims at the analysis of dependencies between the quality attributes based on quantitative methods as well as on the quantification of qualitative methods.

Third, the evaluation results have to be analyzed with respect to possible improvements of the quality in case of changes respectively further development of architecture variants. In addition to an evaluation that determines the quality *of* the architecture, analysis concerned with the dependencies of quality *on* the architecture has to be performed. Besides sensitivity analysis (see Bass et al. [5]), the Modular Performance Analysis (MPA for short, see Wandeler et al. [13]) will be applied to investigate dependencies of the performance quality attribute. The results of the analysis will be interpreted and fed back to the development process. In this paper, we illustrate how the evaluation structure, the QADAG, can be used to set dependencies into relation and to identify potential of an architecture. Thus, development tendencies can be derived and discussed to increase the quality of the architecture. Furthermore, development effort can be saved that may be lost if invested in unpromising variants.

The rest of this paper is organized as follows: The terms used in this work are introduced in Section II. Section III introduces the architecture model which is based on the component-and-connector (C&C) viewtype (see Clements et al. [4]). Architecture variants are built on this model according to three levels of architectural decisions presented Section IV. The impact of those decisions as well as its predictability are discussed in Section V. In Section VI, the structuring of quality attributes is described, which is used in the case study in Section VII that contains the evaluation result representation as well. Section VIII presents architecture evaluation sensitivity in some details that are important for the interpretation of architecture potential analysis and the motivation of further attempts in the development of architecture variants. In Section IX, dependencies in architecture and its evaluation are analyzed and correlated to get statements about the architecture potential. The results regarding some of the dimensions and the evaluation are discussed. Section X concludes.

II. Terminology

As our field of interest is slightly more general than pure software systems, we briefly review the terminology.

Architecture evaluation is directed to software as well as to system architecture in this paper. Embedded systems, consisting of hardware short of resources and software realizing the system functionality, build the center of interest.

A *quality attribute* is a quality goal requiring that the system under consideration at least meets the quality level given in the requirements. The *QADAG* represents a hierarchical structuring of the quality attributes as a directed acyclic graph. A quality attribute, containing scenarios and constraints, may (1) be decomposed into (sub-)attributes or (2) have an evaluation technique. This hierarchical composition can represent the Goal-Question as well as the Factor-Criterion relation of the Goals-Question-Metric (GQM) approach, Basili [14], respectively the Factor-Criterion-Metric (FCM) approach, McCall et al. [15]. An example of a quality attribute is the average bus load that arises from a particular application in a distributed controller network. The bus load may be a subattribute of the system performance.

An *evaluation technique* associated with a quality attribute describes exactly how to evaluate the architecture regarding that quality attribute taking into account the attached scenarios and constraints. This corresponds in some sense to the metrics part of GQM or FCM, respectively.

A *scenario* specifies current and future uses of a system that are relevant for architecture design (cf. Bass et al. [5]). Hence, it describes the interaction between the system and stakeholders.

An *evaluation result* is generated by applying an evaluation technique. The result of an arbitrary unit can be assigned by an *interpretation* to a quality rate.

The *quality rate* is a scaled value (0 and 100 %) representing the ratio of meeting the requirements represented by a quality attribute. The quality rate is also known as *utility* in the economic view of the Cost Benefit Analysis Method (CBAM, see Bass et al. [5]) in direct association to the development expenses spent to reach a particular quality. A quality rate of 100 % represents the best ratio of meeting the requirements, where as a quality rate of 0 % represents the worst ratio of meeting the requirements. It is possible to add a so called K.O.-flag (or just K.O. for short) to a quality rate. This means that the quality of the architecture, regarding a proper quality attribute, will not be acceptable because the system will not be working or even buildable if the quality rate is 0 %. Thus, improper variants can be instantly rejected.

Sensitivity analysis provides information on the dependency of evaluation techniques on the architecture (or at least parts of it). Sensitivity analysis is the first step in getting to know about dependencies between the architecture and its evaluation and dependencies inside the evaluation itself. In the context of this paper, sensitivity analysis is not just meant to identify sensitive points (see Clements et al. [16]) but also to provide explicit information on sensitivity over a certain range of values, i.e. dependencies of evaluation results on architecture artifacts.

A *dependency* describes the architecture and the quality attributes (actually their evaluation result). Furthermore, dependencies also exist between different quality attributes. The latter can be seen as indirect dependencies because they are based on dependencies between architecture and evaluation in most cases. Furthermore, there are dependencies between the properties of architectural elements, e.g. the costs of a more powerful controller will be higher than the costs for a less powerful one. Such dependencies will be necessary to correlate quality attribute dependencies on architecture elements. Actually, the interpretation of a raw value to a quality rate can be seen as one kind of dependency as well although it may be predefined by a system architect.

An *architectural decision* describes the determination of a particular part of the architecture in order to reach the quality requirements. Architectural decisions can be made on different levels which have various impact and predictability (see Section V).

Architecture potential is meant to be the possibilities for improving of the evaluation result of a particular architecture variant (see Florentz [17]). It depends on the actual decisions made for a variant and is analyzed by architecture potential analysis presented in Section IX.

III. Embedded Systems Architecture

The architectures to be evaluated in this approach are embedded system architectures in the automotive domain. The variability of such systems lies in the different hardware platforms used to realize a given function architecture, the mapping of functions to controllers, and the mapping of the communication. A brief introduction to automotive architecture models is given in this section.

The underlying concept is the component-and-connector viewtype (see Clements et al. [4]). The syntax and semantics of components and connectors are extended and described by several metamodels to achieve a well defined and domain-specific ADL. For a detailed description of the metamodels see Florentz and Huhn [18]. In the application domain of embedded automotive systems, the focus lies on functions virtually connected via common signals (provided and required resp.). These functions - building up the system functionality - are distributed to a controller network containing actuators and sensors. Depending on this function mapping, the communication of the functions has to be ensured by transmitting signals across communication lines (e.g. buses); Either between controllers as well as from sensors to a controller or from a controller to an actuator.

The class diagram in Figure 1 shows the assignment of data types to components and connectors on the level of the metamodel. Which properties to assign depends on the concrete components and connectors as well as on the quality attributes relevant in the evaluation process. Thus, properties may be added or refined during the architectural development process, because some architectures may got extended. Thus, more information is available and necessary for further and more precise evaluation. In Section VI, examples for properties in different development phases are provided.

IV. Levels of Architectural Decisions

Building an architecture is - like the whole software or systems development process - based on various decisions. Thus, architecture as structure or structures of a system (cf. Clements et al. [4]) is based on structural

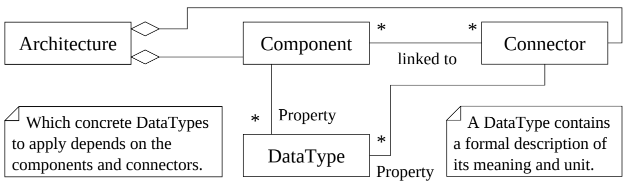
Architecture
Component
Connector
linked to
*
*
Property
*
DataType
*
Property
Which concrete DataTypes to apply depends on the components and connectors.
A DataType contains a formal description of its meaning and unit.

Figure 1. Architecture properties

design decisions regarding the selection of components and the way in which to combine them into a system or subsystem. These decisions are called architectural decisions. Although the structure should be determined before building the system, this is not done in a single step but in an accompanying manner. In this section, levels of architectural decisions are discussed according to the point in time to be made as well as their concern in the application domain of software-intensive embedded automotive systems (cf. Heinecke et al. [2] and Debruyne et al. [3]). Top-level decisions build the requirements for the system to be realized. High-level decisions determine fundamental principles on which the realization will be based. Low-level decisions determine how the system is structured on the fundamental principles to meet the requirements. Figure 2 shows a sequential order from top-level architectural decisions down to low-level ones. Actually, neither the sublevels of each level nor the levels themselves are totally independent. The sequential order shows the chronology of the main levels' decisions. On the right hand side of Figure 2, a differentiation between the impact of the levels' decision and its predictability is depicted.

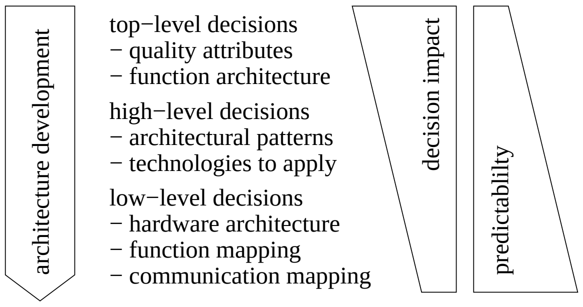

Figure 2. Decision levels

The function architecture, the hardware architecture, the function mapping, and the communication mapping provide views on embedded systems architecture as presented in Section III. In conformance to AUTOSAR and EAST-EAA, the views support the separation of software and hardware to decouple the development of the functionality from the one of controllers. This provides additional flexibility in system development as well as the substitutability of hardware components that may become unavailable over the years.

a) Top-level architectural decisions: They concern requirements on the system to be realized. With the choice of the system functionality by determining the function architecture, the functional requirements are set implicitly, whereas extra-functional requirements are explicitly represented by quality attributes. The relative importance of the quality attributes concerns the top level, too. Thus, what the software or system is meant to do with respect to which particular quality attributes are top-level decisions. Top-level decisions are invariant between several variants to achieve a fair respectively common basis.

b) High-level architectural decisions: They are concerned with the application of architectural patterns and various technical options in order to meet architectural requirements represented by main quality attributes. Thus, high-level decisions are mainly fundamental. They are to be made early in the architecture development process, therefore influencing most of the subsequent decisions. In contrast to top-level decisions, high-level ones may change over variants to explore the suitability of particular technologies and patterns to functional and quality requirements.

c) Low-level architectural decisions: They are closest to the system realization. They deal with the system decomposition, following the top- and high-level decisions. The first sublevel of low-level decisions is the choice of hardware components, i.e. controllers, sensors, actuators, communication lines, and their composition. After the so-called hardware architecture has been figured out, the functionality has to be deployed on the hardware, which is the second sublevel. The decoupling of software and hardware as one of the main intentions of e.g. AUTOSAR requires this step of composition. Furthermore, the function mapping determines which functions will communicate across controller boundaries. The third sublevel is the communication mapping taking the inter controller communication needs into account. Differences in low-level decisions provide most variation points for architecture variants.

V. Decision Impact and Predictability

Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be canceled.

Eoin Woods

The decision impact is greatest for earliest decisions, i.e. top-level directly followed by high-level decisions. Subsequent ones will depend on these and are restricted because they have to take earlier decisions into account. As a consequence, suboptimal early decisions bear many problems and are the hardest to fix. Top-level decisions which lead to concrete functional and extra-functional requirements are the most extensive. Actually, building a system without making these decisions, i.e. without requirements, is trivial as every realization of a system can meet no requirements (cf. Clements et al. [4]). At this point, the legitimate question arises, how top-level decisions can have impact and on what. After all, they actually set the requirements, and thus, have been made without considering concrete ones. Nevertheless, there are requirements to be taken into account. The functionality to be provided and the demanded quality represented by quality attributes are not given as arbitrary choices by the management. The superior goal is to build systems which are realizable, attractive, and competitive on the market. Actually, top-level decisions are made with this superior goal in mind. Once made, they are invariant regarding all architecture variants taken into account. High-level decisions restrict the options for later decisions. They fill the gap between the abstract top-level and the concrete low-level. Consequently, the impact of such decisions is quite high, hence bad decisions may be fatal for the whole project. Low-level decisions still have enough impact to support high quality systems as well as to spoil the system. Thus, they can be likewise fatal but are easier to revise.

The predictability of the impact of different decision level is contrary to the impact itself. The earlier a decision is made, the more impact it has (cf. Bontempi and Kruijtzer [19]), the less predictable the impact is. This is due to the fact that there is a wide range of consequences such a decision may have on the many options to be chosen by subsequent decisions. The predictability of the impact of top-level decisions is based on experiences concerning the buildability of the systems and tradeoffs of quality attributes involved (see Florentz [17]). High-level decisions again fill the gap between top and low-level decisions. Their impact can be predicted easier because the results of applying certain architectural patterns or technologies are well known. Nevertheless, the impact of low-level decisions is the first to be predictable more precisely, the reason for which is the strong relation to the system realization. It can be put into direct relation to one or more quality attributes (mostly subattributes of the main attributes). The impact of an architectural low-level decision can be expressed with means of sensitivity analysis. The dependency of quality attributes on the architecture can represent the impact of low-level decisions. These concrete dependencies allow quite precise predictions.

VI. Architecture Evaluation

The main elements of architecture evaluation are quality attributes which build the QADAG. Its metamodel is shown in Figure 3. Each quality attribute may have several associated scenarios to specify the requirements that have to be taken into account for architecture development. Scenarios are used to express what may happen to a system in its life cycle (see Kazman et al. [20] and [21]). In most cases, scenarios specify requirements relevant for architecture design decisions. An example from the automotive domain is that system architects have to be aware of late changes for certain hardware components. Semiconductor suppliers may announce at any time that some product is discontinued soon and substituted by a defined set of successor products (which partially differ in their properties). Another scenario may anticipate the evolution of the system in future series. To specify further requirements and restrictions that refer to the architecture

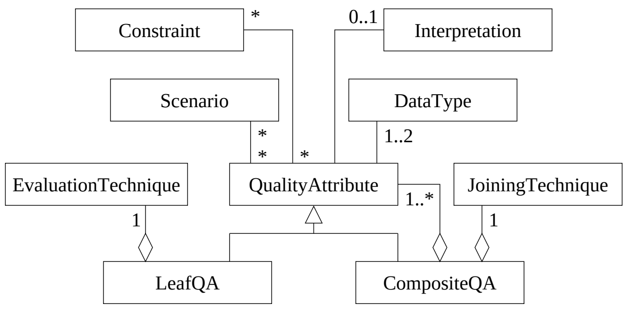

Figure 3. QADAG metamodel

development and realization, constraints are assigned to an attribute. An example for a constraint is that only certain vendors or technologies must be considered.

Further elements of the QADAG are introduced in the following sections. A case study containing an example QADAG is presented in Section VII.

A. Evaluation Techniques

Architecture evaluation can be (1) done in early design phases for a time saving and coarse-grained design space exploration and (2) redone in later design phases for assuring architecture quality. The granularity of the architecture models depends on the design phase in which the evaluation takes place. Therefore, the evaluation techniques applied in the evaluation process may change during the design process to take additional details of the architecture into account.

Initially, several architecture variants participate in the evaluation process. An efficient design space exploration requires a rapid exclusion of inadequate variants. Problems of imprecise and incomplete architecture and design details have to be handled by approximating evaluation techniques performable on the low level of detail available.

One example for changing respectively growing details is the communication in a controller network. It depends on the function mapping as well as the communication mapping. These architectural parts are variables of an architecture and have to be build or modeled during the development process. To keep modeling effort within bounds, concrete communication details for each variant are not provided because most of the variants may not be processed any further. Thus, only abstract information on the communication is available in early design phases. In this case, a simple scheduling algorithm of Liu and Layland [22] is applied to evaluate the network utilization

$$U = \sum_{i=1}^{m} (C_i/T_i)$$

with U as the utilization, C_i as the time needed for the transmission of signal i, and T_i as the transmission cycle of signal i. With given function and communication mapping, only the width of signals in bit and an estimation of their cyclic appearance are needed to compute an approximated average load on known communication hardware (e.g. CAN with 125.000 bit/s transmission rate).

For evaluation as a mean of quality assurance, more and more precise details of the communication may be available. For example, see the performance model which is based on the system architecture model in Wandeler et al. [13]. The Modular Performance Analysis (MPA) expects event stream and resource models as input. Event stream models contain detailed information on the communication behavior, i.e. the occurrences of events respectively signals. Periodic (maybe with jitter or bursts) and sporadic occurrences can be included in the evaluation now. The more conservative approach above has taken only straight periodic occurrences into account because of a lack of concrete scheduling information. Furthermore, a resource model is needed by the MPA describing the resources available in the controller network. In addition to the approach above, MPA takes execution times as reaction of event arrival into account. Thus, not only communication performance but computation performance is analyzed and therefore evaluated as well. The amount of required input data reveals that this approach is most appropriate to limited sets of architecture variants. This keeps the effort of building the input models for each of the variants as low as possible. Based on the MPA, an example how to perform architecture evaluation in the QADAG is given in Section VII. Scenarios and constraints are associated to quality attributes containing an evaluation technique in most cases. In performance analysis for architecture evaluation, a concrete example of an event stream can be considered as a load scenario.

B. Interpretations and Data Types

The hierarchical structure of quality attributes, the QADAG, is based on the composite pattern. Thus, it has leaf and composite quality attributes. Leaf quality attributes contain an evaluation technique describing how to evaluate the architecture regarding the quality attribute. Composite quality attributes contain a joining technique describing how to combine several evaluation results to one. In both cases, the result may be represented as raw value (a data type instance) of a unit predefined by the evaluation technique or the quality attribute, e.g. resource utilization in % as percentage or costs per unit.

Additionally, a quality rate expression is useful, which is a data type instance. A quality rate is an interpretation of the raw value to a scale from 0 to 100 %. It denotes the architecture variant's meeting of a requirement represented by a quality attribute. While raw values are less expressive for people in an architecture development process, who are not close to a particular aspect of architecture, the interpretation to a quality rate can be seen as a common communication basis for architectural decisions. The view graph in Figure 4 is an example for the hardware costs interpretation to the quality rate. A higher quality rate means better meeting of the requirements. Thus, higher costs are interpreted to lower quality regarding the hardware costs quality attribute taking into account the

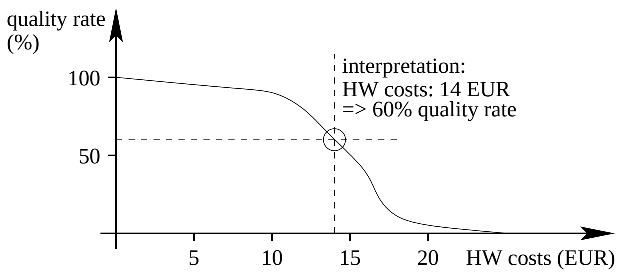

Figure 4. Interpretation of hardware costs

overall hardware costs of one unit of the system to be built. Some quality attributes of the hardware itself may be increased with rising costs. However, this is not part of this interpretation but of e.g. a performance quality attribute.

C. Evaluation Hierarchy and Joining Techniques

The QADAG is not restricted to be a tree because multiple occurrences of the same quality attribute as subattribute and additional restriction for other quality attributes should be allowed. For example, some performance quality attribute can be demanded to be of certain quality to support a modifiability quality attribute. Lacking quality would lead to a rejection of the architecture variant by the modifiability attribute. Modifiability needs performance reserves in case of adding new functionality. Thus, the quality of the performance attribute may influence the modifiability.

A composite quality attribute retrieves its quality result from its subattributes. It has no evaluation technique but a joining technique attached. This joining technique describes how to retrieve the quality result from the results of the subattributes. In most cases, a weighted and normalized summation will be applied to get a result based on several quality rates. Thus, a weight has to be assigned for each subattribute to the superordinate quality attribute. These weights are implicitly contained in the joining technique instance in Figure 3. We do not restrict the sum of the subattribute weights to be 1. This is not necessary because the weighted sum of the quality results will be normalized.

In some cases, the joining of non interpreted raw values may make sense. Hardware costs are one example, because additional expenses in one part of the architecture may be compensated by savings in other parts. Obviously, costs are substitutable along architecture elements, technical properties like RAM or ROM capacities are not. Our experiences in industrial projects have shown that the interpretation of raw values for joining should be considered as late as possible, although an interpretation can be useful for documentation anyway. But an early interpretation means early loss of result details because of the generalizing character of the interpretation. Thus, the coherence of results which may be substitutable, like costs, is lost and may lead to a disqualification of architecture variants, that taken as a whole are promising.

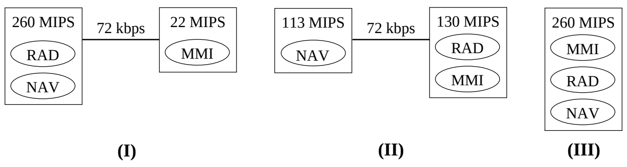

Figure 6. In-Car Radio Navigation System architectures (cf. [13])

VII. Evaluation Case Study

The focus of this case study lies on performance evaluation. Therefore, an example is given (see Figure 5) on how to structure performance quality attributes. Actually, the forming of the QADAG depends on the application domain and partially (for refinement) on the progress of the architecture development. Costs, performance, and modifiability are first class quality attributes to show the benefits of this approach. The rectangles with a double outline above the separator represent composite quality attributes, the singly outlined rectangles below the separator represent leaf quality attributes which have directly associated evaluation techniques not shown in the figure. The meaning of the separator is addressed in Section VIII.

Although the performance of computation hardware (devices) and of communication hardware are related, they can separately be considered. Thus, a more detailed analysis can be applied on the architecture based on more specific evaluation results. The simple scheduling algorithm of Liu and Layland [22] (s.a.) can be used to assess bus utilization as well as RAM and ROM utilization, although RAM and ROM are utilized nearly statically in the automotive domain. The CPU usage is quite a bit more difficult to evaluate. Especially in early design phases, only few details of resource requirements and scheduling are available. Thus, the CPU usage evaluation is mainly based on an expert's know-how. With growing amount of detailed information, approaches like the MPA can be applied in the evaluation process. Figure 6 contains architecture Variants I, II, and III of the In-Car Radio Navigation System introduced and analyzed in Wandeler et al. [13] (actually, I is C, II is D, III is E in the original case study).

A. Evaluating Costs

As can be seen in Figure 6, the architecture variants differ in the underlying hardware architecture. The implementation of the functions is assumed to be invariant regarding the controllers on which they are mapped. Thus, software costs are considered without details. Variants I and II are based on a controller network which leads to additional costs in contrast to Variant III based on a single controller. Furthermore, the computation power will serve as basis for controller costs. Actually, the prices are contrived which does not effect the presentation of the possibilities of this approach. Moreover, prices will change over time anyway. Thus, the documentation of them is important to understand back-dated architectural decisions.

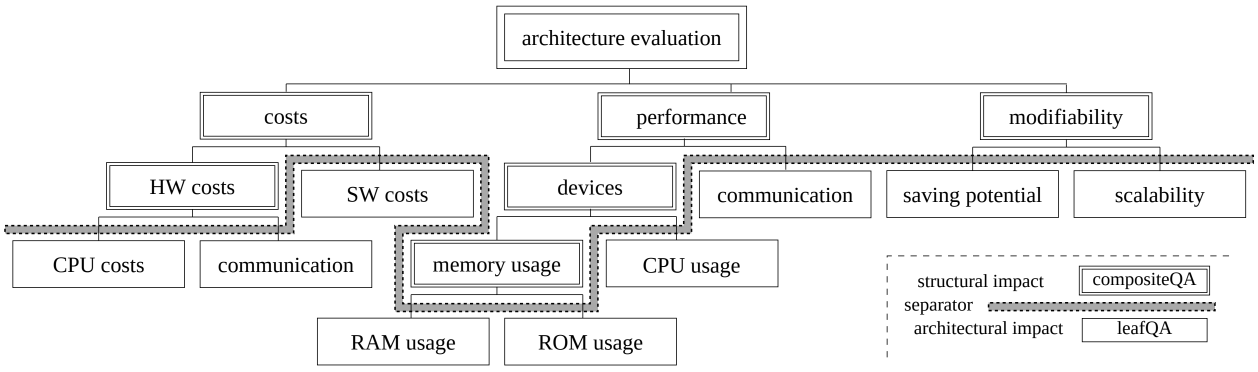

Figure 5. Case study instance of a QADAG

The variants prices are :
Variant I: *€16 (€10 + €1 + €5 (network))*
Variant II: *€14 (€4 + €5 + €5 (network))*
Variant III: *€10*
The interpretation of hardware costs is depicted in Figure 4. The €10 variant is most affordable, €16 the least affordable. Thus, €10 will be interpreted to 90 % quality rate. Actually, €16 is still affordable as the interpretation will be 40 % quality rate. €14 is interpreted to a quality rate of 60 %.

B. Evaluating Performance

MPA is based on the communication behavior of the system. While multimedia mass data is not considered in the evaluated architectures (there are additional communication lines for such data), the delay of communication and the following computations are most interesting. Delay in the communication will lead to delay of the computation start. Furthermore, the computation itself will take time, too. The TMC (Traffic Message Channel) may cause computation-intensive reactions of the system, i.e. recalculation of the route. The driver will tolerate some delay for such calculations, but in case of a change of the radio volume, the reaction should be immediately noticeable to save the drivers patience. Thus, the system performance can be measured by the reaction latency. MPA works on use case scenarios (for communication behavior) represented by message sequence charts with detailed information on time and computation power consumption of the messages and following computations. In combination with knowledge about the hardware resources, an MPA model (performance model) can be created for each architecture variant. This model allows for analyzing (evaluating) the performance of the respective variant. MPA results state that the requirements are met by all architectures. Thus, a 100 % quality rate is assigned for each variant (see Wandeler et al. [13]). The benefits of integrating the MPA are presented in Section IX, in which the results are considered in detail.

C. Evaluating Modifiability

Sufficient performance for all variants and the costs quality attribute in mind qualify the least expensive variant to be realized. This is no surprise and represents most business strategies. The evaluation of the architecture modifiability shows the benefits of taking several quality attributes into account. A growth scenario attached to the modifiability quality attribute defines that the architecture should be reusable in product lines targeted at lower budget. Thus, the navigation system can be left out of the architecture while radio and MMI (Man Machine Interface) are still necessary. Let Variants II and III be favorites because of their low costs. If the navigation system is left out, the hardware architecture of Variant II will be scaled down to a single controller architecture. Variant III already contains only one controller. The performance will not be affected negatively because the computation-intensive navigation system is no longer part of the system (scalability: `ok`). Variant II can get rid of the unused capacities and therefore costs (saving potential: €9), Variant III can not (saving potential: €0). Thus, Variant II without navigation system is only €5 which is half the price of Variant III.

D. Result Representation

Next to specific diagrams representing e.g. technical evaluation results, a summation of the results can be represented with respect to the QADAG. Tables I, II, and III show the essential information to trace the overall results: name of the quality attribute (gray), quality rate (interpreted value), raw value (where available), and weights of subattributes in case of a composite quality attribute with weighted summation as joining technique.

An explicit and detailed documentation of the evaluation is not covered by this representation of course. But for understanding the reasons why some variant got qualified, the table representation is quite efficient.

Furthermore, the tables are the basis for discussion regarding the importance of quality attributes, i.e. their weight. As seen above, the most cost-efficient Variant III is ruled out by the more expensive Variant II. The weights reflect the requirement for at least partial reuse of the architecture in other products or product lines with the background of saving expenses for additional system developments. Omitting the modifiability, Variant III would be the most promising one.

Variant I							
80 %							
-							
50			*100*			*50*	
costs			**performance**			**modifiability**	
70 %			100 %			50 %	
-			-			-	
100		*100*	*200*		*100*	*100*	*100*
HW costs		**SW costs**	**devices**		**com**	**sav.pot.**	**scalab.**
40 %		100 %	100 %		100 %	0 %	100 %
€16		-	-		MPA: ok	€0	ok
100	*100*		*50*	*150*			
devices	**com**		**mem**	**CPU**			
-	-		100 %	100 %			
€11	€5		-	MPA: ok			

(weights)
QA name
quality rate
result

TABLE I.
ARCHITECTURE EVALUATION VARIANT I

Variant II							
95 %							
-							
50			*100*			*50*	
costs			**performance**			**modifiability**	
80 %			100 %			100 %	
-			-			-	
100		*100*	*200*		*100*	*100*	*100*
HW costs		**SW costs**	**devices**		**com**	**sav.pot.**	**scalab.**
60 %		100 %	100 %		100 %	100 %	100 %
€14		-	-		MPA: ok	€9	ok
100	*100*		*50*	*150*			
devices	**com**		**mem**	**CPU**			
-	-		100 %	100 %			
€9	€5		-	MPA: ok			

(weights)
QA name
quality rate
result

TABLE II.
ARCHITECTURE EVALUATION VARIANT II

Variant III							
86.25 %							
-							
50			*100*			*50*	
costs			**performance**			**modifiability**	
95 %			100 %			50 %	
-			-			-	
100		*100*	*200*		*100*	*100*	*100*
HW costs		**SW costs**	**devices**		**com**	**sav.pot.**	**scalab.**
90 %		100 %	100 %		100 %	0 %	100 %
€10		-	-		MPA: ok	€0	ok
100	*100*		*50*	*150*			
devices	**com**		**mem**	**CPU**			
-	-		100 %	100 %			
€10	€0		-	MPA: ok			

(weights)
QA name
quality rate
result

TABLE III.
ARCHITECTURE EVALUATION VARIANT III

Weights and Impact							
25 %			**50 %**			**25 %**	
50			*100*			*50*	
costs			**performance**			**modifiability**	
12.5 %		**12.5 %**	**33.3 %**		**16.6 %**	**12.5 %**	**12.5 %**
100		*100*	*200*		*100*	*100*	*100*
HW costs		**SW costs**	**devices**		**com**	**sav.pot.**	**scalab.**
6.25 %	**6.25 %**		**8.3 %**	**25.0 %**			
100	*100*		*50*	*150*			
devices	**com**		**mem**	**CPU**			

impact
weights
QA name

TABLE IV.
WEIGHTS AND IMPACT

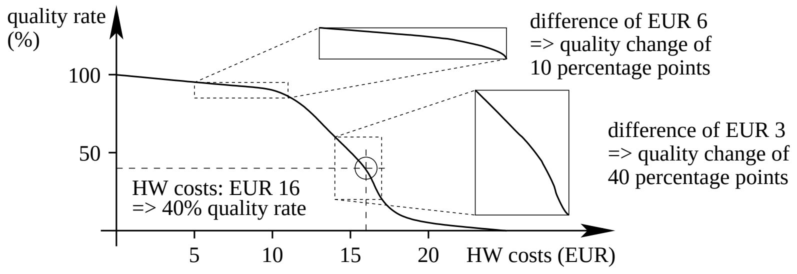

Figure 7. Interpretation of costs

VIII. Evaluation Sensitivity and Challenge

In this section, the sensitivity of architecture evaluation is discussed in detail. We will highlight the difference between structural impact and architectural impact on the evaluation results. The structural impact describes the impact of the evaluation structure, i.e. the hierarchy of quality attributes. Whereas the architectural impact describes the sensitivity regarding architecture artifacts. Figure 5 draws a line between composite quality attributes, which build the hierarchy and are accountable for the structural impact, and leaf quality attributes. The latter describe how to evaluate an architecture and are accountable for the architectural impact. To give a concrete statement about the evaluation sensitivity, both impacts have to be taken into account in combination. Because of competing impacts, which actually lead to tradeoffs, the overall quality requirements may not be reached. To describe a reachable level of quality, which we call the *challenge* of an evaluation, those tradeoffs have to be analyzed and made explicit. The challenge is addressed at the end of this section.

A. Structural Impact

To make it short, the structural impact can be considered as the absolute weight of a quality attribute if the joining of partial results is proportional. (Otherwise, the structural impact can be quite difficult to express.)

Table IV presents the absolute weights based on the relative importance of the quality attribute. We mention this structural impact not just because it is necessary to align different sensitivities. Building a QADAG means building the hierarchy of quality attributes. With growing width, the share of a single quality attribute's impact becomes smaller. With growing depth, the share of impact has to be shared again by the respective subattributes. These facts should be considered for building a QADAG. Our experiences have shown, that disregarding these facts by building a purely organizational arrangement of quality attributes, the absolute weight may not represent their actual importance. Thus, the weights have to be determined in order to represent the importance of a quality attribute like mentioned in Section VI-C. Otherwise, the evaluation results may be corrupted.

Nevertheless, the weights are usually determined in a stakeholder meeting. And their designation can be quite abstract. To avoid this, the Analytic Hierarchy Process (AHP) supports determining the weights based on intuitive statements about the relative importance between two quality attributes. For more information on the AHP see Zhu et al. [23].

B. Architectural Impact

Sensitivity of architecture evaluation describes the fact that the evaluation, or rather its result, is dependent on architecture artifacts. The description of the sensitivities can be quite difficult, not just because of the big amount of sensitivities or resulting tradeoffs. The most challenging and interesting fact is that sensitivities are not just existent or inexistent (see Clements et al. [16]). Dependent on the actual architecture variant, the evaluation result may be influenced in different ways by changing the architecture, e.g. by touching sensitive parts of the architecture.

The description of sensitivity can be quantified in the embedded domain. For example, changing an architecture variant, which is inured to minor changes, may have less effect than changing one in a more critical area regarding a stakeholder's needs. This means the sensitivity depends on the actual architecture variant. Figure 7 illustrates this issue based on a costs interpretation. Because of the non-linear dependency between costs and quality, the actual value of the variant is a matter of particular interest. The sensitivity, or rather its architectural impact, strongly depends on the actual variant and may be considerably different for other variants.

C. Challenge

According to the impact of architectural decisions and the structural and architectural impact of changes on the evaluation results, it is useful to describe how challenging it may be to fulfill the overall quality requirement. A stand-alone quantified quality result alone misses expressiveness. Quantified results are expressive only if (1) comparative results of other architectures are available or (2) a reference value respectively a scale, which represents the reachable quality result, can be given. Actually, to reach the quality of this reference is the challenge of an evaluation and to provide this reference is one of the biggest challenges for sensitivity analysis. Tradeoffs are the key for analyzing the challenge because many strong tradeoffs can make it quite difficult to reach a certain level of quality. Low quality results may be caused by very challenging requirements and not necessarily by bad architecture decisions. The challenge is a very important means of communication between stakeholders. It can be used to explain why some system may not be realizable on an overall satisfactory level.

IX. Architecture Potential Analysis

Architecture potential analysis is motivated by the need for knowledge of the insides of the evaluation, i.e. dependencies of the architecture quality. With this knowledge, design space exploration can be performed more efficiently. Without this knowledge, the evaluation of a new architecture variant has to be performed to check its quality, which can be quite expensive. Known dependencies can guide the development and changes of an architecture and save development resources. Furthermore, they help to explain and document architectural decisions. Documentation is quite important because dependencies may change over time. For example, the dependency between costs and performance depends on the market and available technology. Over the years, the costs for performance (i.e. powerful hardware) will drop as well as the willingness to pay will do. But the need for performance will rise in order to provide additional functionality. This is just a small example for the complex and changing dependencies. To actually analyze architecture and its evaluation regarding its potential, i.e. the possibility of increasing the overall quality of an architecture, the dependencies have to be taken into account seriously.

In this section, we present how to identify important, i.e. to be analyzed, dependencies. The In-Car Radio Navigation System of Wandeler et al. [13] is taken as case study for the architecture potential analysis. After interpreting the results, we discuss how to benefit from the uncovered architecture potential.

A. Identifying Important Dependencies

Architecture quality is represented by quality attributes and eventually by the quality rate which, after all, is the interpretation of evaluation results. Thus, interpretation instances as dependencies of quality rates on evaluation results need to be taken into account first. Either a promising tradeoff is already known or tradeoff analysis (e.g. the Architecture Tradeoff Analysis Method, ATAM, see Bass et al. [5] and Clements et al. [16]) has to be performed. For the case study, the tradeoff between costs and performance has been chosen. The legitimate question arises how to correlate dependencies without common reference. In case of costs, the interpretation in Figure 4 is based on the raw value of the evaluation result which is costs in €. Performance in terms of user noticeable quality is expressed in system reaction delay. Because of various use cases, or in this case rather meaning user interactions, the interpretation has to be given with respect to the particular timing requirements of a particular use case. Figure 9 depicts the common interpretation of delay based on the expectations of the user. The tolerance limit, which represents 100 % of the timing requirements, is applied as point of reference for all use cases.

Although the existence of a costs performance tradeoff is well known, information on how to correlate those quality interpretations is quite rare. Especially a relation between expected delay with respect to various use cases and costs is not available without further analysis. Hence, performance analysis is just a first step of getting a link in a chain of dependencies needed for correlation. Furthermore, the performance analysis results themselves determine additional dependencies to be investigated. As we already know without performance analysis, the MPA provides results describing dependencies in the form of delay of particular use cases in milliseconds over processor speed. While delays regarding various use cases can easily be aligned at the user's tolerance limit (see above), the dependency of processor speed on costs still needs to be investigated. Actually, this can be done by inquiring a business department and sifting through some price lists. In the following section, the steps to correlate the quality interpretations via a chain of dependencies are performed on the case study.

B. Performing the Analysis

The MPA has shown that there is no need to improve the performance quality of the architecture variants. All variants already meet the requirements. Sensitivity analysis is needed to uncover sensitivities of the evaluation results regarding changes of an architecture variant. Because sensitive points (cf. Clements et al. [16]) and tradeoffs are already known, the actual dependencies between quality attributes have to be taken into account as will be shown in the subsequent paragraphs.

Again, the MPA is applied for analyzing the performance. One of the results is shown in Figure 8. This result is taken from Wandeler et al. [13] and will be input for further considerations. The x- and z-axes represent the processor speed available in Variant II. The y-axis represents the delay of handling a TMC (Traffic Message Channel) use case, which describes the system reactions on the reception of a TMC message. Actually, the delay is a (mathematical) function over the processor speed.

The evaluation is based on costs and performance. While performance is sufficient in all variants, costs reduction is the center of interest. But, costs and performance quality attributes are reciprocally influenced by changes in the architecture. MPA provides analysis results for the favorite Variant II as shown in Figure 8. The delay will grow if processor speed is decreased. More delay means worse interpretation, i.e. lower quality rate. In general, higher performance of a CPU is the equivalent to higher costs. Thus, in case of higher performance, the results of the cost quality attribute becomes worse. These facts are combined to uncover architecture potential. Figures 9 to 14 represent input, intermediate steps, and output of the analysis for architecture potential. Following, the rationale and meaning of each coordinate system is explained in detail.

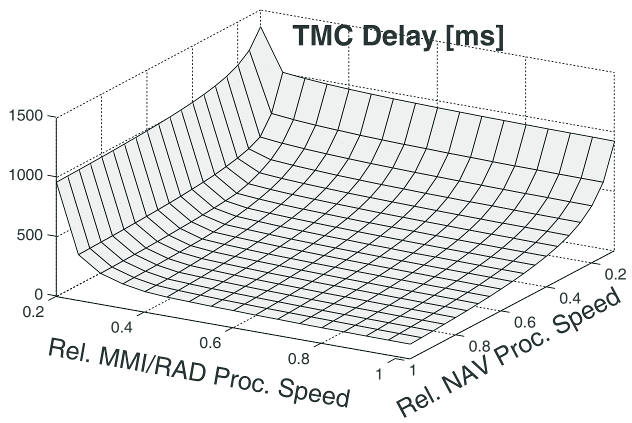

Figure 8. MPA result (Variant II), see [13]

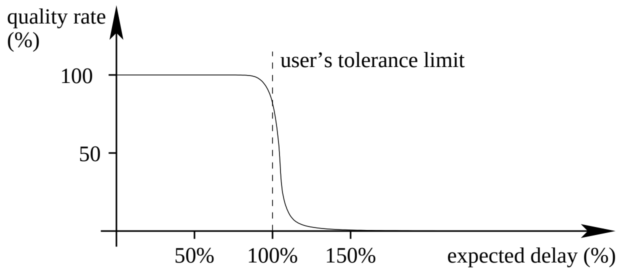

Figure 9. Interpretation of expected delay with user's tolerance limit as reference point

Figure 9: Interpretation of expected delay with user's tolerance limit: The quality rate interpretation of the expected delay and the user's tolerance limit are shown in this coordinate system. The delay is given as percentage because the absolute delay depends on the complexity of the task to be solved. For example, the user will be more patient at complex navigation tasks than at simple volume changes. Thus, the actual requirements depend on the task's complexity. As a common basis, the requirements are taken into account with respect to the respective complexity. The interpretation of hardware costs is already given in Figure 4.

Figure 10: MPA result as delays of MMI/RAD processor speed in various scenarios (Variant II): This figure contains the MPA result for architecture Variant II. It represents the dependency between the MMI/RAD processor speed and the delays of some scenarios, i.e. use cases. The dependencies are nearly uniform, which is true for most delays observed by MPA in the case study. Thus, this result is taken as representative to save analysis effort. For more precise results, this analysis has to be performed for each of the architecture variants, each of the scenarios, and each of the processors deployed in a variant.

Additionally, the user's delay tolerance limit is given explicitly in the coordinate system. This is an important piece of information in order to be able to map costs to delay via the processor speed. Because the quality rate interpretation for delay is given in percent of expected delay and the dependency on processor speed is given as absolute value in milliseconds.

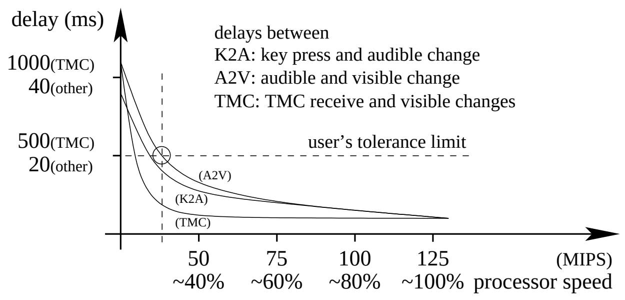

Figure 10. MPA result as delays of MMI/RAD processor speed in various scenarios (Variant II)

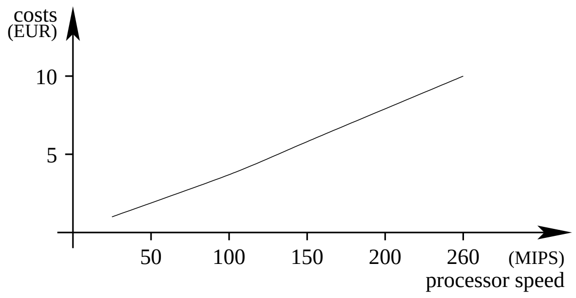

Figure 11. Costs for processor speed (basis for costs-delay dependency)

Please note that the requirements given in [13] are raised to design the analysis more expressive.

Figure 11: Costs for processor speed: The dependency of processor speed on costs is depicted in this system. Such information has to be obtained from the business/purchasing department and may change over time. Thus, it is quite important to document such information for later reconstruction of architectural decisions.

This dependency is necessary in architecture potential analysis because the costs interpretation needs to be correlated with the performance interpretation via the MPA results. Hence, a dependency between costs and the MPA results has to be identified. While the delay will be correlated with respect to the user's tolerance limit, the processor speed needs to be put into relation with costs. This is the rationale of the costs performance tradeoff. If there was no dependency between costs and processor speed, i.e. the hardware performance, this tradeoff would not exist.

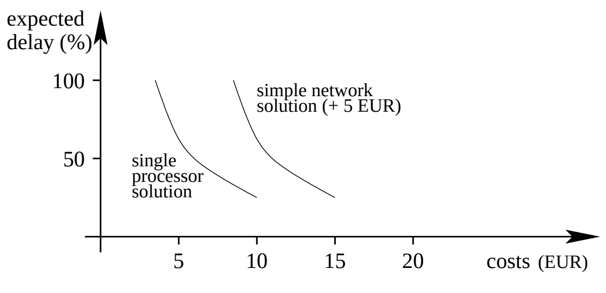

Figure 12. Costs-delay dependencies left: without network, right: with network

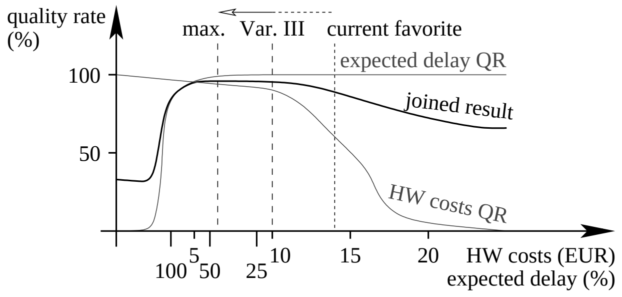

Figure 13. Architecture potential based on costs-delay dependencies without network

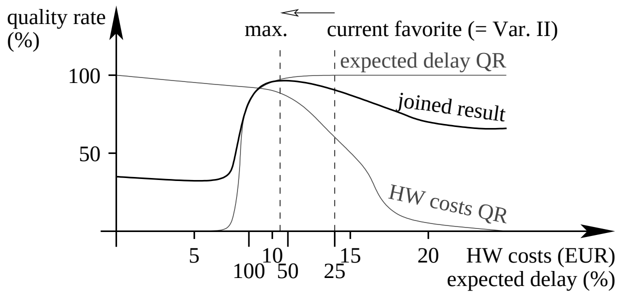

Figure 14. Architecture potential based on costs-delay dependencies with network

Figure 12: Costs-delay dependencies, left hand side: without network, right hand side: with network: The dependency between costs and delay can be derived via the processor speed, because the dependency *of* processor speed on costs and the dependency of delay *on* processor speed are already known. The graphs show the costs of the hardware (without and with network) that keep the delay in the scope of the user's expectations. Until now, only processor costs have been taken into account as hardware costs. Therefore, a second dependency containing the network costs is shown in the coordinate system. This is necessary, because the interpretation of costs takes network costs into account. Although they are invariant regarding a changes of the processor performance, they are not to be neglected. Thus, the additional dependency is given.

The represented dependencies are necessary to correlate hardware costs and (expected) delay on the x-axis of the coordinate systems in Figures 13 and 14. In these coordinate systems, the architecture potential, which to uncover and express was the main intention of the analysis, is depicted.

Figure 13 and 14: Architecture potential based on costs-delay dependency: Both systems contain the same type of analysis result. Because of differences in Variants II and III (with and without network), the analysis results have to be presented separately. Thus, the position of the expected delay quality rate is shifted to the right in respect to the hardware costs quality rate in the second system (Variant II). This is necessary to regard network costs. The third graph in both systems represents the partial quality result based on the weighted summation of both quality rates. The distance to the maximum of this graph (see the arrow) presents the architecture potential for the variants. The results are discussed in detail below.

C. Exploiting Architecture Potential

The analysis of architecture potential uncovers not just potential regarding particular quality attributes. The documentation of the analysis results helps to express architectural knowledge. Besides the directly contained information, it can be justified to derive indirect results on their basis.

First, the costs performance tradeoff can be considered based on the complete sensitivities instead of just sensitive points. Hence, not just the tradeoff itself but the impact of an architectural change is known. It can be predicted quite precisely by taking the intensity of the change into account. Furthermore, the tradeoff can be discussed regarding particular architecture variants for which its effects can be quite different.

Second, an extended view on the architecture rationale, represented by the quality attributes, is provided. In the coordinate systems in Figures 13 and 14 is shown that the overall architecture potential regarding saving expenses is significantly lower for the network based solution. The single controller solution provides more potential on this score. The reason for this are the costs for the network, which are not considered in a processor downscaling of the currently available architecture variants. Actually, these costs push the modifiability result because the system can be modified during design time and even life time without directly affecting the controllers. Thus, a low-price version without navigation system can be easily realized. Moreover, the system can be offered with an optional navigation system (which can be refit later).

While more architecture potential is predicted for the single processor solution, the network solution already made it in the evaluation. To use the identified architecture potential, further considerations should be done. The identified potential is directed to saving expenses while keeping the performance up. Modifiability is not yet included. Thus, the predicted improvements by downsizing one controller are carried forward to the evaluation to consider modifiability. Actually, this procedure is much simpler than extending the analysis for architecture potential. If the feedback reveals ambiguous results, the analysis can still be extended.

The result of a changed Variant II will be raised to approximately 97.8 %. The result of an even more changed Variant III will be raised to approximately 86.7 %. This seems to be surprising at first, because Variant III may be less expensive. But, the original evaluation favored Variant II already because of its higher modifiability, which has not been considered in the analysis for architecture potential. Furthermore, device costs have significantly less structural impact than the CPU performance in this case study (6.25 % vs. 25.0 %, see Table IV) and the architectural impact of costs is limited for Variant III as exemplarily depicted in Figure 7. Thus, although

Variant III has more potential, this is not sufficient to make up the advantage of Variant II.

The nearly optimal fulfillment of the requirements by a changed Variant II do not motivate building additional variants or making further attempts of improvement. At most down-scaling of the remaining controller of Variant II may be taken into account in this special case to achieve even more improvement. Variant III will not be regarded any more.

In a less definite case study, to know about the challenge of the evaluation (see Section VIII) could be helpful to determine whether further attempts are promising.

X. Conclusion

Concise architecture and architecture evaluation models are necessary to perform analysis of architecture potential. Expressing analysis results provides a closer look inside the architecture evaluation which is useful for documenting architectural decisions and knowledge as well as for design space exploration. This closer look supports the quick, cost-efficient, and traceable development of system architectures for dependable embedded systems.

To show applicability of this approach in several domains, a special performance analysis called MPA has been included. The results have been the input for architecture evaluation and for analysis of architecture potential. Furthermore, the results of analysis methods like the ATAM (see Bass et al. [5]) can be used to identify sensitive points and tradeoffs to be explored in detail.

We have shown, that simple architecture evaluation may not be sufficient for developing high quality architectures. But in combination with sensitivity analysis, the potential of an architecture can be analyzed, presented, and integrated in the development. Thus, unused potential can be uncovered, promising changes can be highlighted, and unprofitable development effort can be avoided.

Several approaches—originally dealing with performance analysis—are extended to support design space exploration (see Wandeler et al. [13], Henia et al. [24]). Moreover, free resources and capacities to handle unexpected changes and even imprecise requirements can be detected with such approaches (see Hamann and Ernst [25]). The analysis of architecture potential can be considered as design space exploration regarding architectural decisions introduced in Section IV. An integration with approaches like [13] and [24] has turned out to be quite promising as the results of this work and further attempts have already shown. Furthermore, the integration of hardware problems like cable harness layout (see Gemmerich et al. [26]) for decreasing costs, weight, and installation effort is an interesting field of ongoing work.

Besides the integration of approaches for particular subdomains like performance and layout, an automated architecture improvement support is desirable. It is still a long way to automated generation of architectures, however, the reuse of components, subarchitectures, and architecture knowledge is highly interesting. Especially with availability of component libraries, design space exploration and automated decision support become realistic mid-term objectives.

References

[1] B. Florentz, *Systemarchitekturevaluation: Integration unterschiedlicher Kriterien*, ser. Haus der Technik Fachbuch. expert verlag, 2006, ch. 2.1, pp. 49–65.

[2] H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Mate, K. Nishikawam, and T. Scharnhorst, “AUTomotive Open System ARchitecture - an industry-wide initiative to manage the complexity of emerging automotive E/E-architectures,” in *Convergence 2004, International Congress on Transportation Electronics*, October 2004.

[3] V. Debruyne, F. Simonot-Lion, and Y. Trinquet, “EAST-ADL an Architecture Description Language, Validation and Verification Aspects,” in *Workshop on Architecture Description Languages*, Toulouse, France, August 2004.

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[6] M. A. Babar and I. Gorton, “Comparison of Scenario-Based Software Architecture Evaluation Methods,” in *APSEC*, 2004, pp. 600–607.

[7] M. A. Babar, L. Zhu, and D. R. Jeffery, “A Framework for Classifying and Comparing Software Architecture Evaluation Methods,” in *ASWEC*, 2004, pp. 309–319.

[8] K. Bergner, A. Rausch, M. Sihling, and T. Ternit, “DoSAM - Domain-Specific Software Architecture Comparison Model,” in *QoSA-SOQUA*, ser. LNCS, vol. 3712, 2005, pp. 4–20.

[9] L. Dobrica and E. Niemelä, “A Survey on Software Architecture Analysis Methods,” *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[10] M. T. Ionita, D. K. Hammer, and H. Obbink, “Scenario-Based Software Architecture Evaluation Methods: An Overview,” in *ICSE/SARA*, 2002.

[11] L. Grunske, “Early quality prediction of component-based systems - A generic framework,” *Journal of Systems and Software*, vol. 80, no. 5, pp. 678–686, 2007.

[12] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski, “Recommended Best Industrial Practice for Software Architecture Evaluation,” Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-96-TR-025, 1996.

[13] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, “System architecture evaluation using modular performance analysis: a case study,” *International Journal on Software Tools for Technology Transfer (STTT)*, pp. 1–19, 2006.

[14] V. R. Basili, “Applying the Goal/Question/Metric Paradigm in the Experience Factory,” in *Proceedings of the Tenth Annual CSR (Centre for Software Reliability) Workshop, Application of Software Metrics and Quality Assurance in Industry*, September 1993.

[15] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in Software Quality,” Rome Air Development Center, Tech. Rep. RADC-TR-83-175, Volume II, July 1983.

[16] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures. Methods and Case Studies*. Addison-Wesley, 2001.

[17] B. Florentz, “Inside Architecture Evaluation: Analysis and Representation of Optimization Potential,” in *Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA07)*, January 2007.

[18] B. Florentz and M. Huhn, “Embedded Systems Architecture: Evaluation and Analysis,” in *Quality of Software Architectures and Software Quality (QoSA’06)*, ser. Lecture Notes in Computer Science. Springer, 2006.

[19] G. Bontempi and W. Kruijtzer, “The use of intelligent data analysis techniques for system-level design: a software estimation example,” *Soft Comput.*, vol. 8, no. 7, pp. 477–490, 2004.

[20] R. Kazman, G. D. Abowd, L. J. Bass, and P. Clements, “Scenario-Based Analysis of Software Architecture,” *IEEE Software*, vol. 13, no. 6, pp. 47–55, 1996.

[21] R. Kazman, M. H. Klein, M. R. Barbacci, T. A. Longstaff, H. F. Lipson, and S. J. Carriere, “The Architecture Tradeoff Analysis Method,” in *ICECCS*, 1998, pp. 68–78.

[22] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[23] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery, “Tradeoff and Sensitivity Analysis in Software Architecture Evaluation Using Analytic Hierarchy Process,” *Software Quality Journal*, vol. 13, no. 4, pp. 357–375, 2005.

[24] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System Level Performance Analysis - the SymTA/S Approach,” *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, March 2005.

[25] A. Hamann and R. Ernst, “Efficient Priority Optimization in Complex Distributed Embedded Systems through Search Space Adaptation,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, January 2007.

[26] R. Gemmerich, S. Semmelrodt, A. Zündorf, C. Reckord, J. Leohold, J. Trippler, L. Brabetz, D. Müller, U. Schrey, and H.-G. Weil, “Ein ganzheitlicher Ansatz zur Generierung und Optimierung von Fahrzeugbordnetzen,” in *VDI Berichte Nr. 1907 (12th International Conference and Exhibition Electronic Systems for Vehicles Baden-Baden)*, October 2005, pp. 597–608.

Bastian Florentz is currently a Ph.D. candidate at the Technical University at Brunswick, Germany. He has received his Diploma and MSc degrees in computer science from the Technical University at Brunswick, Germany, in 2003, both.

His research interests include software and system architecture, architecture evaluation, sensitivity analysis as well as behavior and performance modeling.

Michaela Huhn has received a Ph.D. from the University at Hildesheim, Germany, in 1997. She has worked on formal methods for testing and verification in several industrial and academic projects.

Her main interests are constructive and analytic methods to assure the quality of embedded and safety-critical systems. After a period of work on behavioral models, architecture as a major issue for software quality came to her focus.