Are you aware of the design decisions?

On how modeling should support design

Hristina Moneva, Roelof Hamberg, Teade Punter
Embedded Systems Institute (Eindhoven, The Netherlands), e-mail: [firstname.lastname]@esi.nl

***Abstract*—Complex systems design faces lack of seamless integration of model-based methods and techniques. In this paper we propose a framework that enables designers and architects to track design decisions throughout the design process. It provides a mechanism to deduct the impact of each design decision on the system under design. Furthermore, it supports model and result management while analyzing and experimenting with models throughout the entire design cycle.**

Keywords: model-based engineering; modeling formalisms; multidisciplinary design; integration framework;

I. INTRODUCTION

Model Driven Engineering is often advocated to handle the problems that complex systems design is facing, e.g., late integration and productivity [1, 7]. The importance of modeling was supported by the outcome of a survey that we did in December 2009 during the ESI Symposium 2009 in Eindhoven, the Netherlands. The survey was amongst more than 100 industrial and academics people that were interested in designing complex systems. 59 out of 63 responders answered "yes" on the statement: "Do you believe that modeling is essential for designing complex systems?" and only 3 people chose the option: "don't know".

In practice, often multiple formalisms are used to model the desired system or parts of it. The need for multiple modeling techniques stems from their ability to answer different questions. Complex systems do often require the involvement of multiple disciplines and specialists. These require different modeling approaches too. A second important aspect is concurrent engineering. Specialists from diverse teams are designing the same system simultaneously and the communication between them is crucial for early error detection, which is usually hampered by implicit assumptions and failing communication of taken design decisions. Lastly, we observe that systems are not modeled completely in the industry. Often, only the critical parts are modeled [Boderc2006]. The reason is that it is not cost effective to model the complete product.

These observations are relevant and were strongly supported in the survey introduced before, which results are depicted in Figure 1. All these aspects of reality shape the need of support for the design process in a new way.

In summary, designers are in need of a solution that is formalism-independent, providing connections between the different models to be able to track decisions, thereby increasing the level of understanding of the impact of each decision taken in the course of designing, and allowing for incomplete system modeling.

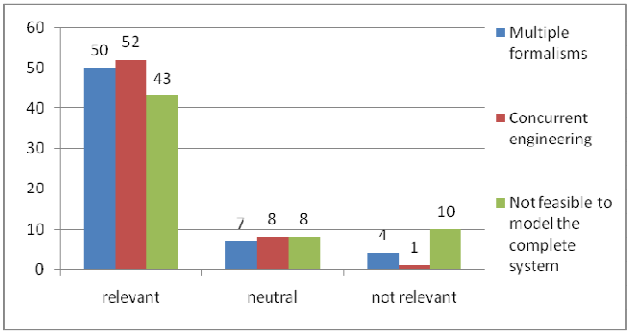

Figure 1. Are these 'aspects of reality' of designing complex systems relevant? – survey results

II. APPROACH

In this section, we present the reasoning that led to the proposed approach, the model that is targeting the support of a model-based design process as we believe it should be, i.e., with partial modeling, as well as showing how it relates to current industrial practice.

A. *Starting points*

In the Boderc project [Boderc2006] a design methodology for high-tech systems was proposed [Heemels2006]. It discusses the design process that should be used when having constraints like project duration and available man power. It proposes an approach how to focus the designing effort and time by two means – focusing in-depth analysis (via modeling) on the most critical issues and using simple models to create insight in a design decision. The "high-level method" can be given as a collection of three steps – preparation of the design, selection of critical design aspects, and their evaluation.

In the preparation phase, the key drivers and requirements are identified as well as their realization concerns, and core domain knowledge is made explicit. The selection of critical design aspects covers activities related with identifying tensions and conflicts in a qualitative manner as well as gathering facts and identifying uncertainties to qualify the tensions and conflicts. The evaluation of design aspects comprises of building of small models and performing related measurements. These steps are to be used iteratively, so

that progressive knowledge can be accumulated and applied.

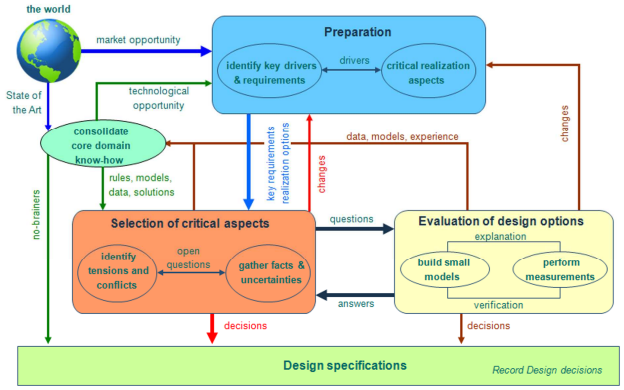

Figure 2. Dynamic flow of information in the Boderc method

The dynamic flow of information and the generic context of taking design decisions are shown in Figure 2. The iterative nature of the method can be followed through the steps as described above. For instance, once the evaluation gives conclusive answers on a particular issue coming from an identified realization of a concern via the selection of critical design aspects, a design decision can be taken. The iteration will proceed to a next issue of concern and all important information obtained during in-depth study of previous issues should eventually be consolidated in the core domain knowledge.

Another important aspect is the set of viewpoints that should be incorporated in the overall design of complex high-tech systems [ISO/IEC 42010]. The architect is interested in an overall view on the problem, where all these viewpoints (different domain specialists with their concerns) are present simultaneously [Muller2007]. The limitations of the human brain force the architect to create an overall view by quickly alternating the individual viewpoints. The order in which the viewpoints are alternated is chaotic: problems or opportunities in one viewpoint trigger the switch to a related viewpoint.

Finally, threads-of-reasoning as a means of integrating the set of views and the system qualities should be considered [Muller2007]. They provide a way to observe and reason about specific system qualities or concerns by connecting customer needs, specifications, and design issues, and connecting them to the overall design and present core domain knowledge in the organization. That can be seen as a multi-view cause-effect or trade-off analysis, or just as an exploration of a particular system quality/concern.

B. The model

This section presents a model that supports the design methodology as presented in the previous section.

1) Overview via types of design activities

One way of classifying the variety of activities in a design process is according to their level of formality. On the one hand, more formal activities exist, which cover aspects like creating, manipulating, and analyzing concrete models. On the other hand, the more informal activities describe all the development steps that are made and the design decisions that are taken. These steps and decisions put all models in the context of the desired system. The informal activities can also be further distinguished: they are either activities related to the concrete design process or activities related to the design itself.

In order to support all design activities in a design process, a generic integration framework model is proposed (see Figure 3). The integration framework consists of three abstract levels or layers: design flow, design views, and models. The framework is designed to be generic and to be able to fit any existent design process or development life cycle, any discipline or specific view, and to enable the usage of any formalism required in the working process. Due to the genericity of the approach, multiple features can be added to support the designer solely based on the data, which is already present in the model. The ability to tailor this framework to the needs of a specific environment is considered an important usability issue.

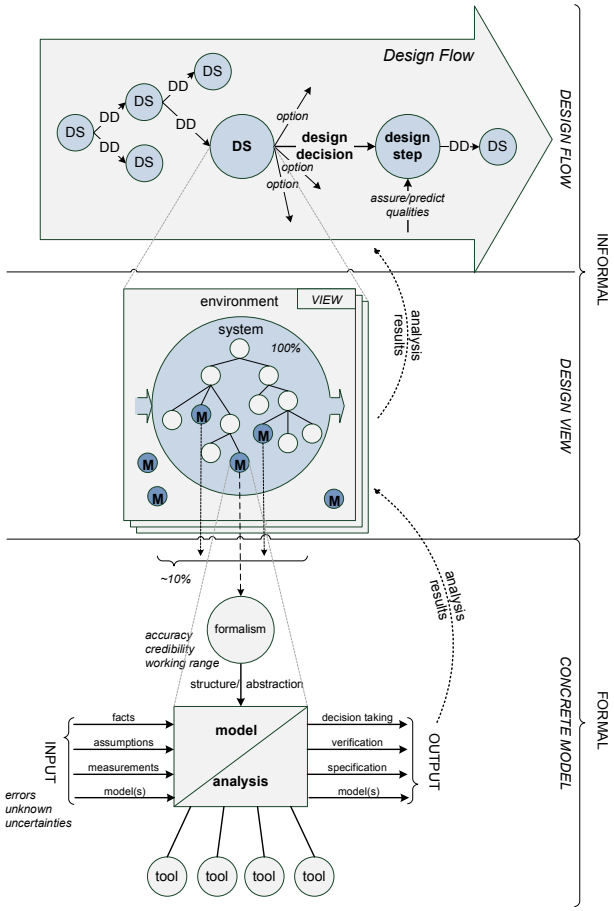

Figure 3. Integration Framework model

2) Design flow

Designing as an activity can be described by two main ingredients – refining the designed system until a moment when a set of options is present and a decision has to be taken. If we try to represent this process, it will be tree-like and it will comprise of a number of nodes (design steps)

and edges (design decisions or options). Often, some design activities do not lead to the desired system (dead branches, dead ends), some possibilities are not explored further yet (alternatives), or the designer even follows a few options simultaneously in order to gain better in-depth insight in particular aspects of each option.

3) Design view

The design view may be considered as a specific representation of the system. It might represent a discipline, e.g., software, hardware, mechatronics, electronics, and material flow, or a specific aspect, e.g., performance, safety. Every view bears a unique decomposition of the system under design. Empty structural basic blocks are used for the decomposition. They form the skeleton of the view. These blocks may contain or may be contained in other blocks.

The basic blocks are used as a container of the detailed models. The blocks are formalism-independent and consider the model from a black box perspective. One block may contain multiple models, each of which is developed in order to analyze a specific concern or quality of the system or parts of it. If multiple models are present in one block, they are not considered as redundant, because they may model different aspects of the same system block. The necessity of being able to accommodate multiple models is also based on the need of answering multiple design questions by analyses and the inability of any specific formalism to cover all required aspects.

Every block is characterized with a set of parameters. Each parameter might have a range of values and a unit, and may have dependencies to other parameters. The parameters are used as a mechanism to couple blocks either within one or between multiple views. The interoperability of multiple formalisms and models is also enabled through these parameters. The basic block and the parameters that characterize it are used as an abstraction of the real detailed models. Due to the parameter dependencies, conflict detection between concrete parameters and their values – inputs or outputs of various models and their analysis results – can be used.

4) Model

The need of modeling is based on the need of gaining in-depth knowledge of the system under design or its parts. Each model is expressed in its own formalism, which is suitable for analysis of specific aspects. The model usually has a number of inputs and is based on a set of assumptions, measurements, or even other models due to model transformations. The detailed model as a function of its formalism and inputs with their errors, unknowns, and uncertainties also has its own accuracy, credibility, and working range [Muller2007]. These are relevant attributes that should be known to the designer.

In general, multiple experiments can and will be performed on one model. The type of experiment is limited by the set of tools that support the formalism and their abilities. The decision to store the results of a particular experiment is under control of the designer. With any experiment data should be stored about the tool that is used, its parameters, and the results. The results may be used for further decision taking, verification of assumptions or system qualities, or specification of some system parts. A specific result may be another model which in its turn is used for further analysis.

5) Features

A number of features can automatically be harvested to support design activities as a result of the genericity of the model and the incremental data that is provided by the user in the course of designing.

a) Conflict detection

Based on the data that is being stored in the framework during designing, a continuous conflict detection mechanism can be employed to support the user. The mechanism applies to parameters, their values, and to dependencies, and it will continuously check for incompatibilities. Suppose that an experiment is run on a model and the result is used as an input parameter to another model as well. When the first model's experiment yields a new result, the second model's results are already obsolete. The need of reminding the designer to rerun an experiment would be of great use in practice as well as supportive information about model versions.

b) Decision tree and impact of design decisions

An important feature is the ability to deduce the impact of each design decision on the system under design. That can be obtained by differencing two design steps and observing the changes in the design. This feature facilitates retrospection of the decisions taken and their exact impact on the system. Often, when taking a decision this does not imply that designers are aware of the changes required to the design in advance. In such cases retrospection may help to improve the understanding as well as to give space for improvements if necessary. This method can be also applied to a set of decisions.

c) Model and result management

At each design step, the state of modeling is preserved. This provides model management in the design process, while allowing to step backwards when the design does not result into the desired system. One can continue from an already existent state of modeling instead of starting from scratch. The result management allows the designer to preserve concrete experimental setups and results, which (s)he considers to be the valuable ones. The preserved results are attached to the exact model version they belong to. In practice that is often a chaotic process and needs special attention and support.

d) Representations

By representing the design process in different ways, we can improve learning about the work we do. For example, just ordering the design flow along a time line based on when design decisions are taken and when we have to reverse, we can learn about the type of mistakes that we made. Figure 4 shows a complete design flow that was restarted after some work was done already. Why was this done? What went wrong? It might be interesting to look at the decisions taken and learn from it.

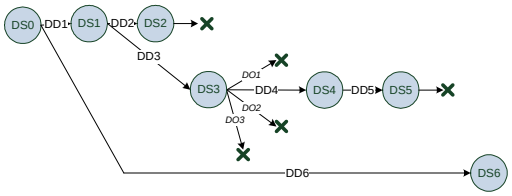

Figure 4. Time-line design flow representation

By having a statistical representation of how many times a system block was touched in the process of taking decisions, we can see whether or not our estimated time and effort for developing components matches to the actual figures. This allows further identification to conclude perhaps that critical system parts were underestimated (Figure 5).

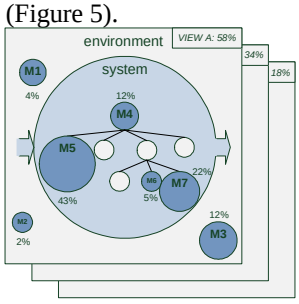

Figure 5. Statistical representation of effort spent per view and component

e) Reuse

We observe in industrial practice, that reuse of components or complete systems is common practice. However, reuse does not always fit the new requirements and constraints. In order to be faster in adapting reused components/subsystems to the new context, it would be easier to reason about the new solution if the design of the reused subsystem is present and it is easier to find the available variation points in the design (see Figure 6).

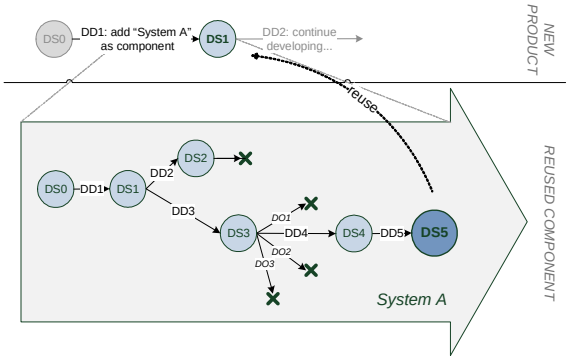

Figure 6. Transfer of knowledge in new design context

Moreover, frequently the knowledge of your organization is depleted when architects change their job or retire. To avoid such situations, a process which stores all implicit knowledge in a single-point-of-truth would be very valuable. It also allows new employees to learn faster about the system and not wasting valuable time of specialists in discussions (at least in early stages, when the reason is just transfer of information).

f) Merge and branch

When talking about concurrent engineering, issues as merging and branching are taken into account. Firstly, the simple case is just using a repository, which is in control of the distributed environment and a number of users in their specific roles.

Secondly, branch and merge might be needed in an organization where multiple teams are working concurrently and follow their own processes, but the results are integrated on regular bases (see Figure 7).

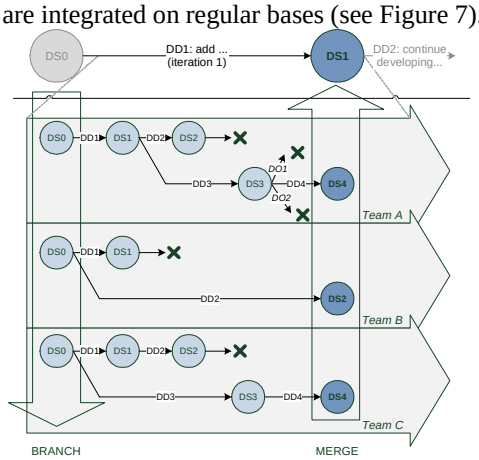

Figure 7. Concurrent engineering (branch and merge)

Suppose a process which is driven by iterations that will introduce new features, not based on design decisions. If a number of teams work on the same new feature it may be interesting to see the assumptions the other team makes while you are having your own understanding about the problem. In this context, the design flow introduces the concept of meta-flow, where multiple concrete flows can be part of one iteration/phase.

g) Domain tailoring

When introducing new methods and tools in the industry, issues such as the effort needed to change the way of working, suitability to familiar methods and/or tools, and usability should be taken explicitly into account. Therefore, we think that the added value of the proposed framework comes with adding the opportunity of tailoring the framework to specific organizational needs. There are a number of aspects which should be flexible w.r.t. changes in order to support the tailoring process, such as processes, views, specific formalisms, and tools.

The tailoring of process and views is seen as an extension of the current model. The idea of a meta-process can be used as an extension of the current model in two different ways – to support sub-flows in case of concurrent engineering or to support predefined iterations in a concrete product life cycle.

The extension of views can be based on domain specific languages with standard system decomposition instead of the proposed generic structural blocks as well as predefined set of system views. Therefore it is possible to start with a standard for one's own organization, set of

views and standard components instead of always starting with an empty project.

Another attractive opportunity is to support a specific set of formalisms and tools. In this way, the parameters characterizing the basic blocks can be automatically extracted and models can be generated on the basis of the system decomposition already present in the view. Special support for performing experiments, automatic invocation of tools and collecting of results can be added.

When talking about support to specific organizations or domains we can also consider issues such as support for organization-specific glossaries, taxonomies, and domain-specific phenomena (representation of standard relations and domain-specific laws). Depending on specific system views and their decompositions, support for design space exploration (e.g. Y-chart) or other tools that may support the design process (help for finding design options and of the decision taking) can be provided as well.

h) Exploration or cause-effect analysis

Relationship based exploration, as discussed in [Sinha, et al. 2006], is designed for source code exploration. It allows to expand the visualization interactively from any starting point (method in a class) and to continue the exploration in any direction (to and from) based on the provided information in the model relations (method calls) to other objects (classes). The method can be used in the framework by substituting the objects with system blocks and the calls with parameter dependencies. Following that basic idea, we can use it for showing and storing specific aspects of the system under design in cases where views reflect disciplines and the system qualities/concerns are spread among them (e.g. availability, performance). It also can be applied for exploring the system views and store exploration traces for later use/reuse. These exploration traces can be also seen as the threads of reasoning that we have introduced before.

In our survey, introduced in section I, we asked the respondees also for the relevance of the features that we propose in our framework. Figure 8 shows the outcome.

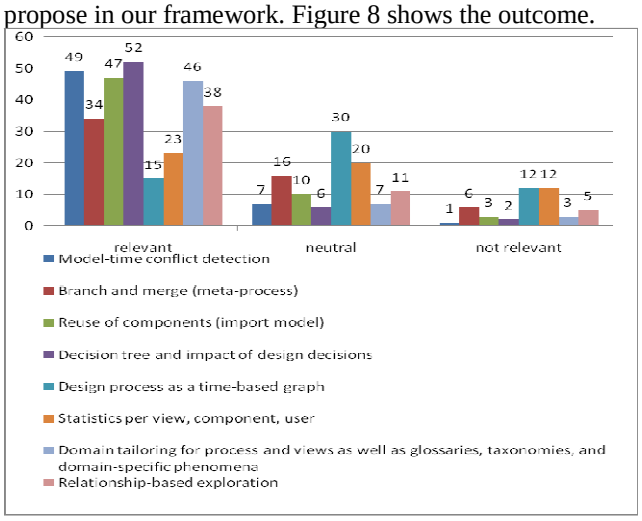

Figure 8. Are these 'features' for supporting design relevant? – survey results

C. Application

A reflection on the design process of a new picking concept within Vanderlande Industries (ACP distribution concept) in the period July-October 2009, was carried out to fill in the artifacts in the integration framework of the proposed approach. This reflection is probably neither complete nor entirely correct, but it provides a good insight in the main activities of the ACP project team in this period and was used to validate and improve the conceptual model as such (no prototype used).

First of all, the steps in the design process were considered. Due to the domain knowledge present in the team (what are the decisions to be taken and what is their impact on the system under design) the process was mainly risk-driven. For that reason, the design decisions taken in the design flow level of the framework, relate to specific risks, and the sequence in which they are dealt with is based on their impact on the concept, probability of occurrence, and uncertainty about the possible solutions.

Next to this, the team's progress in terms of design views was considered. One of the prevailing views is the physical view, which is used to synchronize between team members how the system will look like and how it will work. Coordination of work across different subsystems typically requires another view with a different decomposition. These views are refined and changed on a frequent basis. Next to the graphical representation of design views, documents are produced that describe sets of requirements or specifications of components, functions, or interfaces. These elements coincide with the elements that can be identified in the graphical representations. In this way, we see a good coverage of all available information by having multiple graphical views.

Finally, models are used at many levels in the team. At the system level, design choices taken from several alternatives are dealt with in a qualitative way. This is supported by simple, first order models in Excel that reflect static calculations of system costs and subsystem performance. In a number of cases, simulation models were available in order to study dynamic system behavior. One of the first simulation models was quite detailed and with a lot of simplifying assumptions in it. Apparently, it was constructed that way, because the applied modeling tool (known by the simulation engineer) required it. In general, the synchronization of all assumptions, prerequisites, input parameters, and model results across team members, was done through weekly team meetings and frequent bilateral consultations. It appeared difficult to do this in a watertight way.

Some preliminary observational conclusions that stem from this first realization step are:

- The conceptual model covers the design activities observed in this case.
- It should be natural for designers to relate design uncertainties with high impact (i.e., risks) with the design steps in the design flow.
- The design views need at least a functional and a physical component view. Further alignment with

architectural approaches such as Y-chart [Kienhuis1997] and “4+1 views" [Kruchten1995] can be beneficial.

- The conceptual model made us aware that some models seem to precede the questions they are trying to answer, probably because their formalism and approach is well known. This can be avoided by making the relation to design issues and the number of assumptions that have to be made much clearer.

III. Discussion

The Boderc design methodology for high-tech systems, which we have presented in the beginning of this paper, is an approach that has been applied in practice. The proposed integration framework aims at bridging the high-level method with existing design tools.

Many model-based and model-driven solutions that are applicable in design and development processes can be found. They can be mapped to each of the three levels of the proposed framework, i.e. being related to the design process, to the system structure and its interrelations, or to the detailed modeling of some system parts.

When considering the model-based tools used for detailed models, it can be observed that they focus on a concrete modeling formalism. This formalism is used as a basis for tools providing answers to concrete questions such as absence of dead-lock, performance, and schedulability. Examples of tools of this group are CIF, Uppaal, mCRL2, UML, PHAVer, gPROMS, Modelica, and Matlab, and many others. Some of them can be connected to other formalisms by means of model transformations, but as a whole, they are independent and not coupled. These tools have the tendency to lock designers in a mode of using a single tool, even when design problems require multiple means.

The model-based approaches at the system level usually combine a number of formalisms and tools by means of model transformations. These usually are able to work with limited set of formalisms and often are targeting one concrete domain, e.g. automotive. Examples are Ptolemy, SysML, SystemC, and Autosar. The problem is that it is difficult to extend these tools in order to support other domains or to extend the set of tools which are supported. Another important issue is the lack of support for incomplete modeling, which as earlier discussed is an important ingredient to fit the current industrial needs.

Another important group of tools involved in the design and development cycle of a product is positioned at process level. These are the data repositories, requirement tracking tools, architectural documents, etc. Examples are Lattix, Doors, Clearcase, and SVN. They play a central role in the complete process, but the problem is that they do not track the process as a time-line and do not show the progress of models, architecture, and decisions. Often they are also too specific: coupled to a specific development process and design framework.

The integration of all these tools and harvesting their specific benefits and advantages has been targeted by the proposed approach in order to enable the designer to choose which ones he wants to use and to provide a “single point of truth” view on the system.

IV. Conclusions

In this paper, we have presented a conceptual framework of a design methodology that was derived from our experiences in industrial practice. The main value of our approach is the integrated support of design activities, covering three related but different levels – process (design flow), system decomposition (views), and models and analyses. What especially distinguishes this approach from others is: 1) flexibility of using different tools and formalisms which enables you to deal with model heterogeneity, 2) design-time error detection by keeping the dependencies explicit between design decisions, system elements, and models, 3) support for incomplete modeling, and 4) the ability to show the impact of each design decision on the system under design.

Acknowledgment

This work has been performed as part of the “Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems (MULTIFORM) project, supported by the Seventh Research Framework Programme of the European Commission.

Grant agreement number: INFSO-ICT-224249.

References

[1] Boderc: Model-based design of high-tech systems, edited by W.P.M.H. Heemels and G.J. Muller, Boderc Symposium 2006, published by Embedded Systems Institute, Eindhoven, http://www.esi.nl/publications/bodercBook.pdf.

[2] ISO/IEC 42010, “Recommended Practice for Architectural Description of Software-intensive Systems”, 2007.

[3] W.P.M.H. Heemels, E.H. van de Waal and G.J. Muller. “A multi-disciplinary and model-based design methodology for high-tech systems”. Proceedings of the Conference on System Engineering Research (CSER), Los Angeles, CA, 2006.

[4] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 338–349. IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[5] P. Kruchten, Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), November 1995, pp. 42-50.

[6] G.J. Muller, “Coupling Enterprise and Technology by a Compact and Specific Architecture Overview”, INCOSE, San Diego, 2007.

[7] D. Schmidt, Cover feature – Model Driven Engineering. IEEE Computer, February 2006, pp.25-31.