Putting Chaos under Control: on how Modeling should Support Design

Hristina Moneva, Roelof Hamberg, Teade Punter
Embedded Systems Institute
Den Dolech 2, Laplace Building 0.10
5612 AZ Eindhoven, The Netherlands
[firstname.lastname]@esi.nl

John Vissers

Vanderlande Industries B.V.
Vanderlandelaan 2
5466 RB Veghel, The Netherlands
john.vissers@vanderlande.com

Abstract. Complex systems design faces the lack of seamless integration of model-based methods and techniques. In this paper we propose a framework that enables designers and architects to track the design decisions throughout the design process. It provides a mechanism to deduce 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.

Introduction

This paper addresses the design of high-tech systems. High-tech systems are by definition constructed using cutting edge technology, and consequently embedded system technology plays a major and often even decisive role in such systems [Brinksma and Hooman 2008]. A large variety of high-tech systems exist, ranging from medical devices, such as a Magnetic Resonance Imaging, wafer scanners for IC lithography, to mobile phones, HDTVs, cars and airplanes.

The design process of high-tech systems is often characterized as being *complex* process resulting in complex products. These products are complex because getting the system overview is hard. The complexity originates from the increased number of product functions, as well as the increased difficulty of these functions, as well as the increasing number of people involved and the growing size of the international collaborations they work in, see, e.g., [Borches and Bonnema 2009]. The complexity of the design process is often detailed by other qualifications. Although we might not be complete we think system design faces three main challenges, namely: multidisciplinary development, concurrent engineering, and multiple formalisms.

Multi-disciplinarity is a challenge because multiple disciplines and specialists, such as mechanics, hardware and software, are simultaneously involved in system design, which requires interactive communication between the stakeholders in the disciplines with the aim of integrating them. High-tech systems are often developed according to a *concurrent engineering* approach, rather than along a linear process. Development phases are (partly) done in parallel to shorten development time, to increase quality and to reduce the development cost. This dynamic and flexible approach is needed to deal with the continuous change in requirements: the ‘Tsunami’ of functionality [Punter 2008]. Another driver for the need of concurrent engineering is that timing of new products is based on new technologies. For example, in the low-end TV market, TV-suppliers shop around each half year to find appropriate (quality versus cost) components for their TVs. The implication of concurrent engineering for the design is that specialists from a variety of 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 design decisions. Yet another challenge for the design process is working with *multiple modeling formalisms*. The main reason for modeling is rooted in the need to get answers to design questions about a system. A formalism is the syntax and semantics (the language) to describe a model covering those system parts that are relevant for, e.g., a performance analysis [Bosch, et al. 2007]. The instances of the formalisms are called models, e.g., a performance model. We observe that multiple formalisms are applied in industrial practice. The need for multiple modeling formalisms stems from their ability to answer different questions. Another observation that we made is that high-tech systems are hardly ever modeled in a complete way [Heemels, et al. 2006]. Rather only the critical parts are modeled. This is due to the fact that it is not cost effective to model the complete product.

This paper addresses the complexity of design processes, while in particular addressing the three challenges as stated before. It does not propose a new method or approach, but combines existing approaches and recent developments, like the approach proposed in [Heemels, et al. 2006], and presents a framework for system design instead. The result is a conceptual framework with a high-level method that encompasses the main constructs to reason when designing High-tech System Design. This framework is needed to position formalisms, models, tooling in a high-tech systems' design flow

Concepts for design

In the Boderc project [ESI 2006] a design methodology for high-tech systems was proposed [Heemels, et al. 2006]. 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 – focused 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.

The dynamic flow of information and the generic context of taking design decisions are shown in Figure 1. The iterative nature of the method can be followed through the steps as described above. For instance, once an 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.

A further important aspect is the set of viewpoints that should be incorporated in the overall design of complex high-tech systems [ISO/IEC-42010 2007]. The architect is interested in an overall view on the problem, where all these viewpoints (different domain specialists with their concerns) are present simultaneously [Muller 2007]. 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.

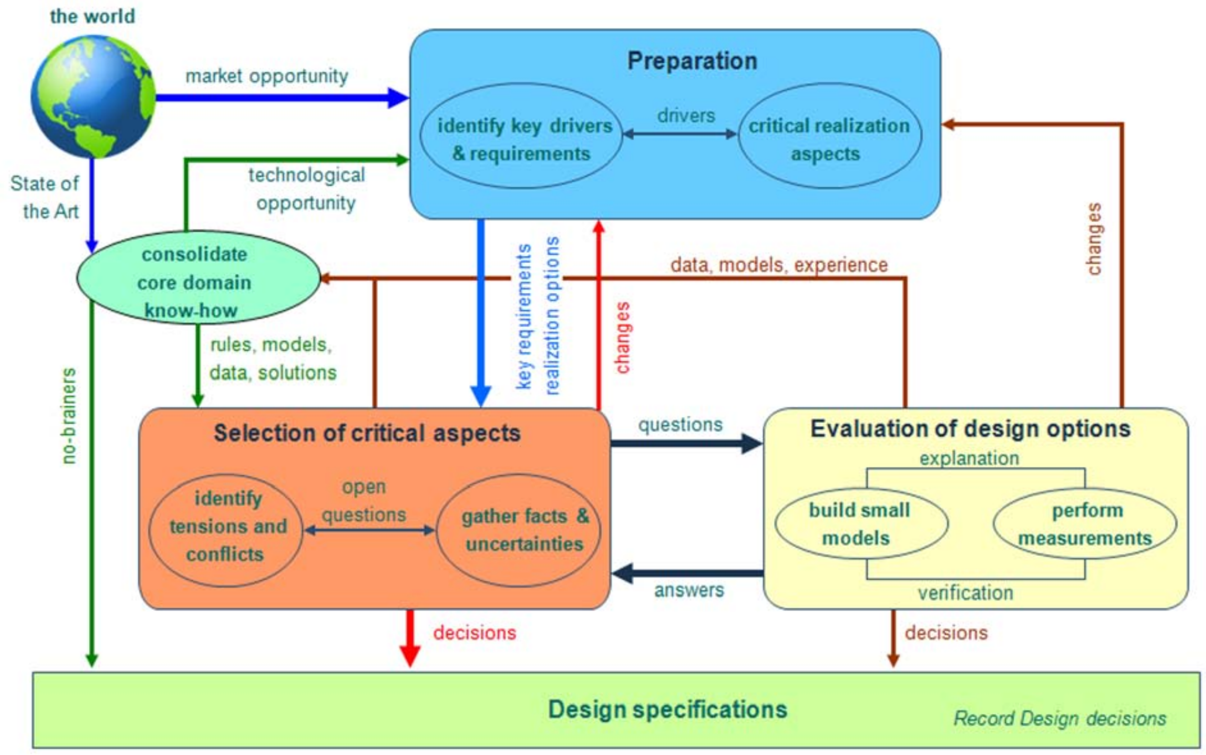


Figure 1. Dynamic flow of information in the Boderc method

Finally, threads-of-reasoning as a means of integrating the set of views and the system qualities should be considered [Muller 2007]. They provide a way to observe and reason about specific system qualities or concerns by connecting customer needs, specifications, and design issues, and to connect 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.

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 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 each of them focuses 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 Uppaal [Uppaal 2009], mCRL2 [mCRL2 2009], UML [OMG 2009(2)], Modelica [Otter, et al. 2007], and Matlab [Taylor and Kebede 1996], 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 the mode of using a single tool, even when design problems require other 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 a limited set of formalisms and often are targeting one concrete domain, e.g. automotive [Schmidt 2006]. Examples are Ptolemy [Ptolemy 2009], SysML [OMG 2009(1)], AADL [Lewis and Feiler 2006], SystemC [Panda 2001], and Autosar [Autosar 2009]. 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 actual industrial needs.

Another important group of methods and tools involved in the design and development cycle of a product is positioned at process level. These are the data repositories, requirement tracking tools, etc. Examples are Doors [IBM 2009(2)], ClearCase [IBM 2009(1)], and SVN [Collins-Sussman 2002]. 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 framework 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.

The conceptual framework setup

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 framework model is proposed (see Figure 2). The 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 framework, 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.

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.

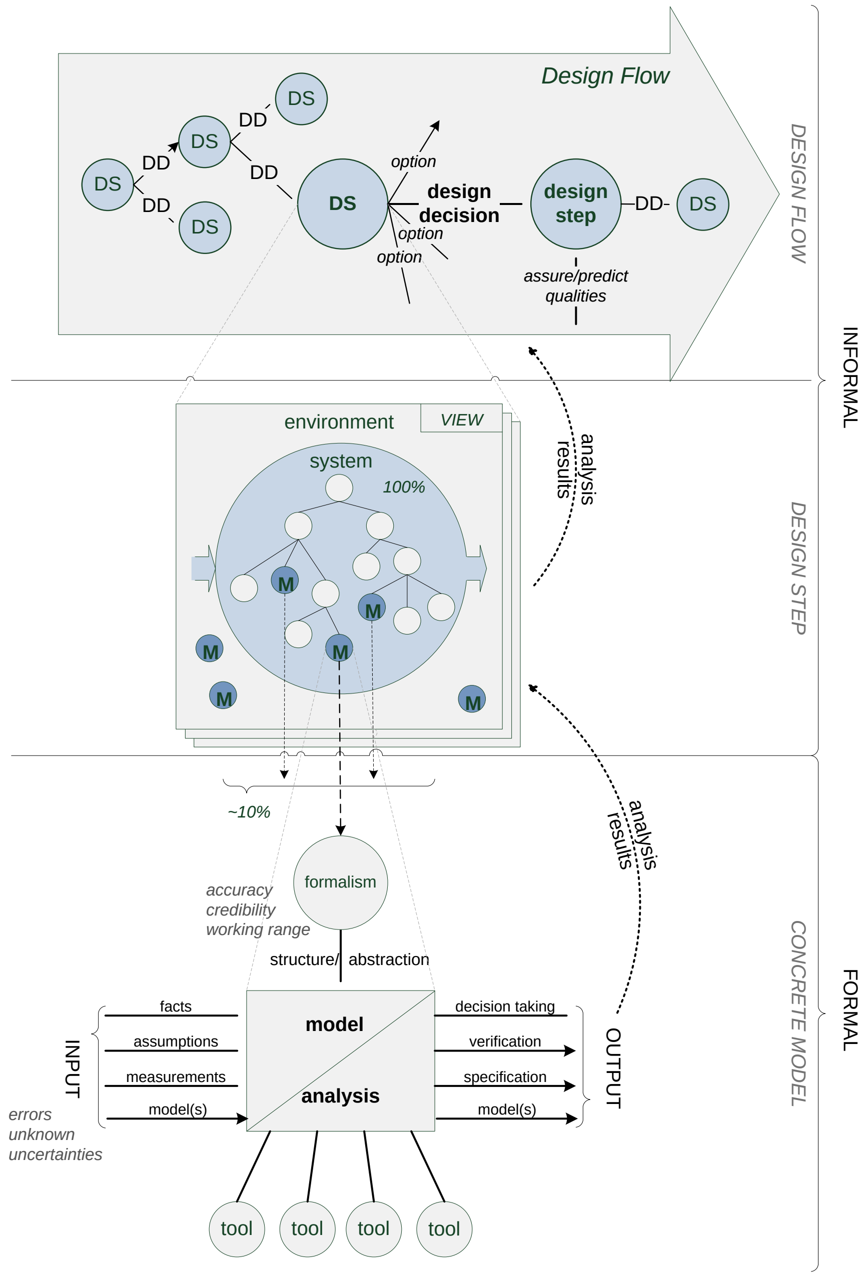


Figure 2. Framework model

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. For the decomposition empty structural basic blocks are used which 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 value, a unit, a range, 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.

Model

The model part is the most formal activity within the design process. 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 [Muller 2007].

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

Features

A number of features can automatically be provided to support these activities due to the genericity of the model and the data that is provided by the user in the course of designing.

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. It is based on the parameters, their values, and dependencies, and it continuously checks for incompatibilities. One important aspect of this is that if an experiment was run on a model and the result is used as an input parameter to another model as well, then in the moment 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 the versions of the models and their state.

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 steps and observing the changes in the design. This method can be also applied to a set of decisions and seeing their impact, but that can be applied only if they are in sequence with each other in the design process.

Model and result management. At each design step, the concrete state of modeling is preserved. That provides model management in the design process, allowing to step backwards when the design does not lead to the desired system and to 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.

Representations. By representing the design process in different ways, we can learn better about the work we do. For example just ordering the design flow along time line based on when design decision was taken and when we had to go back, we can learn about the type of mistakes we make. On Figure 3, can be see that the complete design process was restarted after some amount of work already done. Why? What went wrong? Look at the decisions taken and you may learn from it.

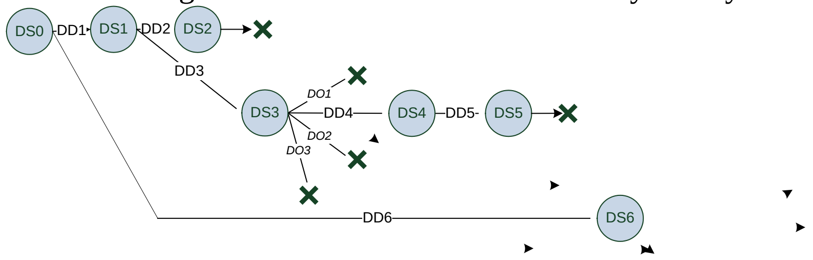


Figure 3. Time-line design flow representation

By having statistical representation of how many times a system block was touched in the process of taking decisions, we can see whether or not we really spend our time and effort on the components that were identified as critical or we underestimated some system parts (see Figure 4).

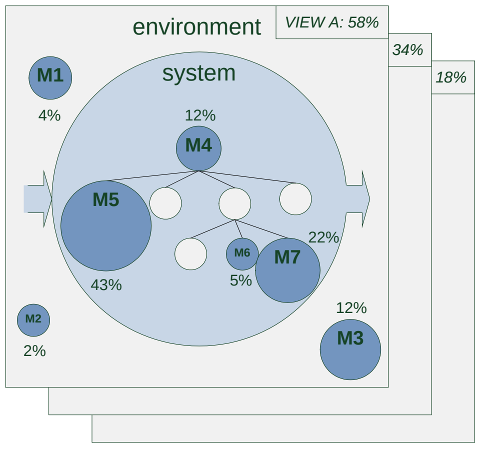


Figure 4. Statistical representation of effort spent per view and component

Reuse. Reuse of components or complete systems is common practice. But 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 5).

Moreover, how often the knowledge of your organization walks out the door with the architect who decides to change his job. 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).

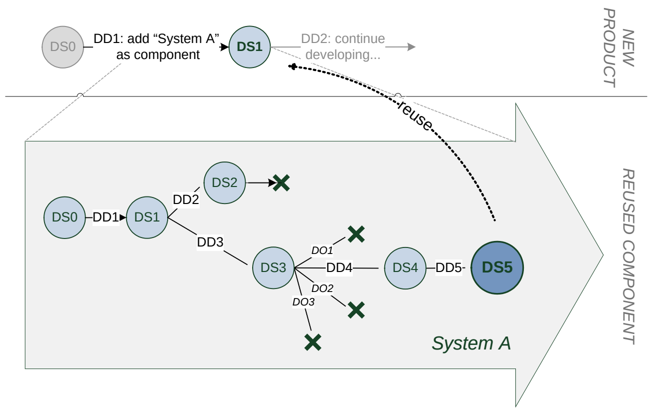


Figure 5. Transfer of knowledge in new design context

Merge and branch. When talking about concurrent engineering, issues as merging and branching have to be 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 can 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 6).

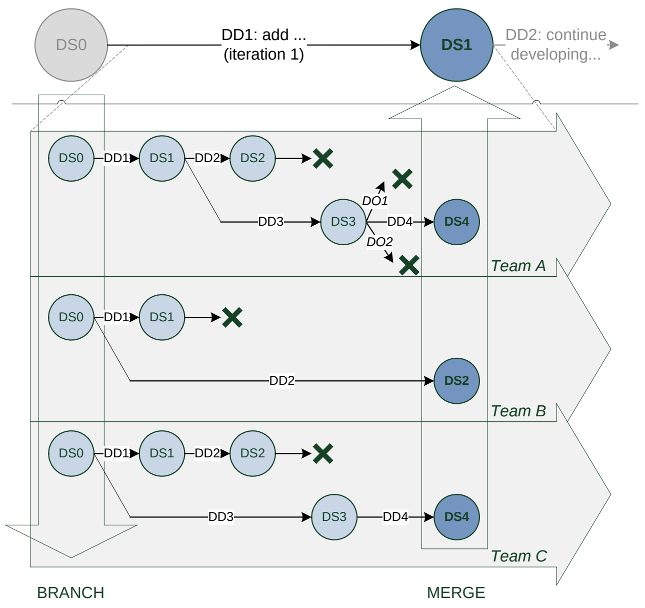


Figure 6. Concurrent engineering (branch and merge)

Similarly, if a number of teams work on the same new feature (in the context of a process which is driven by iterations introducing new features, not based on design decisions), it may be interesting to see the assumptions the other team makes while you are having your own understanding of 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.

Domain tailoring. When introducing new methods and tools in the industry, issues such as the effort to change the way of working, suitability to familiar methods and/or tools, usability should be taken into explicitly account. For that reason, we strongly believe 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 to changes in order to support the tailoring process, such as process, view, 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. In this way, you can start with standard for your 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 view and its decomposition, support for design space exploration (e.g. Y-chart) or any other tools that may support the design process (help for finding design options and of the decision taking) can be provided as well.

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 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. security, 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 introduced earlier.

Case study

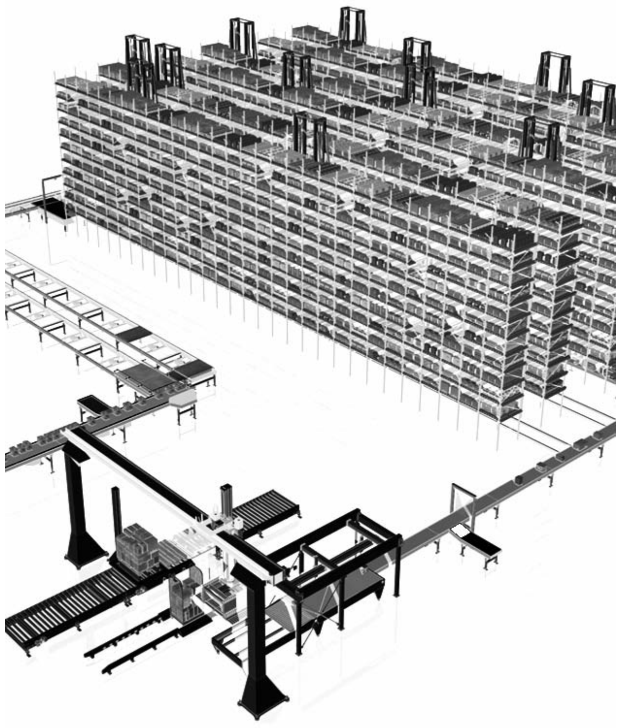
In this section we present a case study that illustrates the framework that we presented before. The case is a new picking concept for warehouses that is developed in 2009 by Vanderlande Industries in Veghel, the Netherlands (http://www.vanderlande.com/).

The goal of the case study is to learn about the framework in a real environment, to show that the three levels of our framework are valid and that they are supportive to structure a design process in practice. Therefore we followed a part of the design process of a new picking concept. The reflection of this process is neither complete nor entirely correct, because it is a running project. Nevertheless, it is quite relevant to obtain early feedback on the framework in order to illustrate the basic idea underlying the concept. The sequence of design options and decisions was closely followed. We analyzed the main views that played a role during design, as well as collected the models that were made and analyzed by the development team.

The modular picking concept involves the integration of a large number of existing and new components, most of which are developed by partners. The development team consists of members who have expertise that cover the most relevant activities that must be carried out during development. This is spread across economical and technical feasibility of specific components as well as system integration factors that influence the feasibility of the picking concept as a whole.

Design flow

The design flow, i.e., the sequence of design steps that are related by more or less explicit design decisions, is not trivial even for a relatively small team like the picking development team. The flow is driven by reduction of risks in order of relevance. The risks are estimated according to their probability of occurrence, their impact on the concept, and the uncertainty that exists about their possible solutions.



In almost all cases, the risk can be phrased as a tension between a number of (non-) functional requirements, such as system performance and cost. In general, many solutions exist to provide the required system functionality. A number of basic choices are asserted in order to make progress. Some of the very early choices have been largely indisputable, whereas most of the risk mitigation strategies involved relatively small changes or evolutions to the basic concept. Only in one case a significant redefinition of the basic module was made. At that point in time, two independent system performance models of the own concept as well as of a competitor's concept gave comparable evidence for severe deficiencies of the actual concept. In a time-line representation this will show as a long branch, sprouting from an early design step, indicating the reconsideration of multiple decisions that were taken since that time.

Design views

The applied design views that could be discerned were mainly functional and physical in nature. In an early phase of concept development this is not strange, as the functional view provides a refinement of the system requirements and the physical view provides the team members a tangible representation that they can reason about. Moreover, in logistics it is often the case that the mapping of functionalities on task executors (also called resources, to be identified with the physical components from the physical view) bears little redundancy, i.e., a single function is provided through a number of identical resources. This facilitates life when it comes to reasoning about different views on the system.

Discipline-specific views (like electrical or software engineering views) were not observed, but this would be rare in an early phase anyhow. First of all, the top level concept has to be detailed in smaller functional blocks in order to be able to think of possible solutions.

Towards the end of the studied period a part of the high level control of the total system has become candidate for a redesign. In this area the interaction with the proposed new software architecture is large: the currently available functional views of this part will be matched with software building blocks in the near future. These can be viewed as physical blocks as well, although many people would refer to that view as the software which is mapped onto the hardware that runs that software.

Models

A lot of models have been made in the studied period. These models range from very basic Excel files to calculate averages and weighted sums of series of numbers to quite advanced simulations in Automod and SHESim/POOSL. We have studied the following aspects of the models and the modeling activities: the developer's competence level, sustainability, interrelationships, and applied abstraction level.

Developer's competence level. The effect that a formalism as well as the tool that implements it, should both be known by a developer in order to apply it, is trivially understood, but severely underestimated in terms of impact. The examples are abundant: many Vanderlande developers are knowledgeable in Excel and it is used frequently, the simulation engineers at Vanderlande are trained in Automod and they have a preference to apply that tool in analyzing simulation models, and SHESim/POOSL was applied by an already experienced user of that tool. Only one exception to this heuristic rule was observed: one developer learned to apply new knowledge about availability relationships between system and component levels.

Sustainability. We have also observed that some models are much longer lived than others, which means that the models do exist for a longer period of time, are regularly updated, and are used in a repetitive way throughout the design process. A good example of a sustainable model is the cost model that supports all system cost estimations in all phases of the development so far. It is a modular model in the sense that it connects to other standardized sources of information available in Vanderlande. The system cost estimations are made regularly in the project to benchmark the economical foresight of the concept. Many other models are made in a short period of time, analyzed, and only remembered in terms of their final results. An example is a model used to analyze the influence of keeping tasks in sequence on the overall system performance. The design was adapted as a result of this model, which originally showed a severe impact. The model results as such are remembered as a curve (a visual relationship) where the relative performance drops as a function of the task aggregation level under certain specific conditions.

Interrelationships. This brings us at the next observation when it comes to models, their inputs, and their results. The number of interrelations between models is limited, but quite crucial. Moreover, people tend to forget about one or two factors. Generally speaking, in logistics it is difficult to neglect the influence of the specific customer details. As a rule, types and quantifications of the customer's business processes have a significant influence on the logistic system. Now, especially for models that are analyzed by simulating them, these dependencies cannot be transferred in a transparent way from the input to the output. Therefore, the results of such models are explicitly dependent on specific customer data input, which is easily forgotten when conclusions are built on top of their results only.

Levels of abstraction. Logistic systems are difficult to model when it comes to their dynamic behavior. They consist of many interacting parts, but not that many that macroscopic, fluid-like models work sufficiently well. The difficulty of the models is reflected in the struggle of developers to make strong assumptions or abstract from a lot of details. On the other hand, the visualization of models that enables team members to understand the system's behavior and reason about it, calls for specification of details that are not yet relevant at the time the visualization itself is needed. Such temporary model inputs require the developers to stay alert on the goals of models, their factual inputs and the necessary assumptions that were made in order to make the models work.

Summary

Considering the Vanderlande case as described above, one can infer a number of conclusions that serve as feedback to our presented framework. First of all, in the design flow we have observed that risks, unknowns, and uncertainties are strong drivers for how the detailing of system design is being done. This can be considered to be independent from the design steps and design decisions that we introduced earlier - it rather adds to it. Secondly, a number of different design views frequently pop up. Of course, this observation is strengthened by the fact that there are several architectural description methods that use a multi-view approach. Introducing a number of standard views into the framework has advantages and disadvantages, which will be weighted through further studies and discussions. Lastly, the model part appears to fit seamlessly in the framework approach: the interdependencies, input-output relations, assumptions, measurements, and model relationships were all observed in a relatively short and limited development process.

The presence of domain knowledge through experience of developers, past projects, known (heuristic) laws, or already present models in the framework would add tremendously to its usability. Therefore, we want to extend the framework – this is discussed in the next section.

Outlook

In this section we are extending our framework based on the lessons learned in the case study. This extension is about the core domain knowledge (see Figure 1) and it consists of the design decisions that have to be taken, the risks related to them with attached impact, uncertainty, and probability. Often this domain knowledge also contains possible architectural alternatives, the dependencies between these options, and the impact on the system qualities.

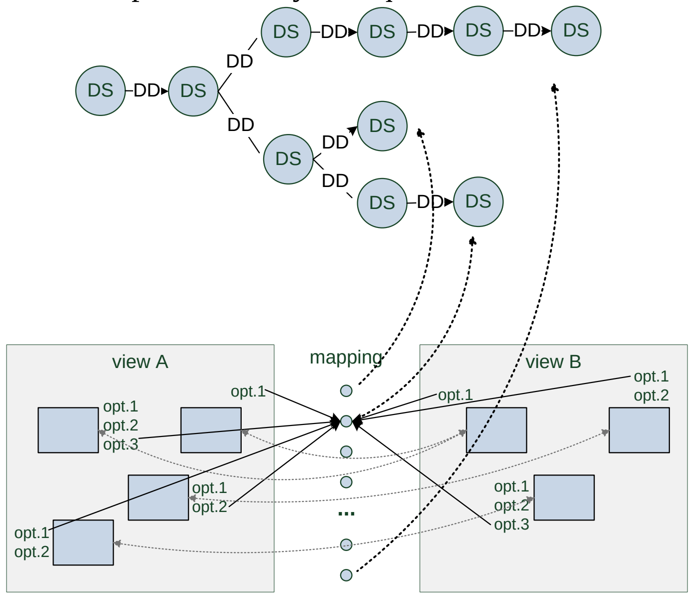


Figure 7. Model of the core domain knowledge

Modeling core domain knowledge requires a representation. Several representation forms are available: Y-chart [Kienhuis, et al. 1997], “4+1” views [Kruchten 1995], FBS/PBS (functional/physical breakdown structure) [Liberti, et al. 2009], GOMS (goals, operators, methods, and selection rules) [Kieras and Knudsen 2006], GTA (groupware task analysis) [Veer, et al. 1996], etc. The framework should support a generic representation, which will allow to be tailored to a specific one. The commonality between them is that the system is represented by number of views, each of the views has its unique structure, and it may have formal, semiformal, or informal mapping between elements and views. Some of the architectural approaches allow storing alternatives for system parts. The alternatives may have dependencies, e.g. choosing a specific option may filter out/restrict the options present in other components or possible mappings to other views.

We further discuss the relationship between the framework and the core domain model. A branch from the design flow represents only one possible mapping of system views and elements (see Figure 7). Often in early stages of the design, more than one branch (design decision) is followed, respectively multiple mappings are explored, in order to gain more knowledge about the system under design. It is interesting to see that different usages can be envisioned. For example, when designing a system without much prior domain knowledge, it is possible to explore one or more design options in order to gain it. But once the design domain and its dependencies become more familiar, the design process may be driven by the accumulated knowledge as in the Vanderlande Industries case the process is driven by the risks associated with the design options. This process may be assisted by optimization or domain space analysis/exploration methods if multiple alternatives are present.

Conclusions

In this paper, we have presented a conceptual framework of a design methodology that has been applied in practice. The main value of our framework 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 framework 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

AutoMod. 2009. `http://www.automod.com/` (accessed November 8, 2009).

Autosar. 2009. `http://www.autosar.org/` (accessed November 6, 2009).

Borches, P.D. and M. Bonnema. 2009. Coping with system evolution, In Proceedings of the Nineteenth Annual International Symposium of the International Council on Systems Engineering. Singapore: INCOSE.

Bosch, P. v.d., M. Verhoef, G. Muller, and O. Florescu. 2007. Modeling of hardware software performance of high-tech systems, In Proceedings of the Seventeenth Annual International Symposium of the International Council on Systems Engineering (San Diego, CA). Seattle:

INCOSE.

Brinksma, E. and J. Hooman. 2008. Dependability for high-tech systems: an industry-as-laboratory approach. In: Design, Automation and Test in Europe. DATE '08, Munich.

Collins-Sussman, B. 2002. The subversion project: buiding a better CVS. Linux J. 2002, 94, pp.3.

ESI. 2006. 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.

IBM. 2009(1). Rational ClearCase: complete software configuration management. http://www-01.ibm.com/software/awdtools/clearcase/.

IBM. 2009(2). Rational DOORS: A requirements management tool for systems and advanced IT applications. http://www-01.ibm.com/software/awdtools/doors/.

ISO/IEC-42010. 2007. Recommended Practice for Architectural Description of Software-intensive Systems.

Hamberg R. and J. Vissers. 2009. Automatic Case Picking development and modeling – A reflection on process, concept refinement, and model support. 2nd Falcon Research Fair.

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

Kienhuis, B., E. Deprettere, K. Vissers, and P. van der Wolf. 1997. 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.

Kieras, D. and K. Knudsen. 2006. Comprehensive Computational GOMS Modeling with GLEAN. Proceedings of BRIMS-2006.

Kruchten, P. 1995. Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

Lewis B. and P. Feiler. 2006. Multi-Dimensional Model BasedEngineering for Performance Critical Computer Systemsusing the AADL. ERTS 2006, Toulouse, France.

Liberti, L., D. Krob, F. Marinelli, and O. de Weck. 2009. A General Framework for Combined Module- and Scale-based Product Platform Design, Second International Symposium on Engineering Systems. MIT, Cambridge, Massachusetts.

mCRL2. 2009. http://www.mcrl2.org/ (accessed October 7, 2009).

Muller, G.J. 2007. Coupling Enterprise and Technology by a Compact and Specific Architecture Overview. In Proceedings of the Seventeenth Annual International Symposium of the International Council on Systems Engineering (San Diego, CA). Seattle: INCOSE.

OMG. 2009(1). SysML. http://uml.org/ (accessed October 21, 2009).

OMG. 2009(2). UML. http://uml.org/ (accessed October 23, 2009).

Otter M., Elmqvist H., and Mattsson S. E. 2007. Multidomain Modeling with Modelica. Handbook of Dynamic System Modelling. editor Paul A. Fishwick, Chapman & Hall/CRC, chapter 36, pp. 36.1 - 36.27.

Panda, P. R. 2001. SystemC: a modeling platform supporting multiple design abstractions. In Proceedings of the 14th international Symposium on Systems Synthesis, Montréal, P.Q., Canada.

Ptolemy. 2009. http://ptolemy.eecs.berkeley.edu/ (accessed October 3, 2009).

Punter, T. 2008. High-tech systems in their environment. Report on INCOSE 2008 STT2.

INCOSE Insight, September 2008.

SHE/POOSL. 2009. `http://www.es.ele.tue.nl/she/` (accessed November 6, 2009).

Sinha, V., D. Karger, and R. Miller. 2006. Relo: Helping Users Manage Context during Interactive Exploratory Visualization of Large Codebases. Visual Languages and Human-Centric Computing (VL/HCC 2006). Sep. 4-8, 2006, Brighton, United Kingdom.

Schmidt, D. 2006. Cover feature – Model Driven Engineering. IEEE Computer 39 (2), pp.25-31.

Taylor J.H. and D. Kebede. 1996. Modeling and simulation of hybrid systems in MATLAB. Proc. ifac World Congress, pp.275—280.

Uppaal. 2009. `http://www.uppaal.com/` (accessed November 6, 2009).

Veer, G.C. van der, B.F. Lenting, and B.A.J. Bergevoet. 1996. GTA:Groupware Task Analysis - Modeling Complexity. Acta Psychologica, 91, pp. 297-322

About the authors

Hristina Moneva received her MSc/eng degree in Computer Systems and Technologies (1997-2001) at the Technical University – Varna, Bulgaria and continued her education at Stan Ackermans Institute with the Software Technology program (2006-2008), where she received her PDEng degree. She is currently working as a Research Assistant at Embedded Systems Institute. Her main interest is in finding the way to combine the academic innovations with the industrial needs and she definitely likes challenging tasks.

Roelof Hamberg received his MSc and PhD degrees in Theoretical Physics from the Universities of Utrecht and Leiden, respectively. He joined Philips Research in 1992 to work on perceptual image quality modeling and evaluation methods. In 1998 he joined Océ as a developer of in-product control software, shifting his role via digital system architect to department manager. In 2006, he joined ESI as research fellow. His research areas of interest are easy specification, exploration, simulation, and formal reasoning of system behavior, and systems architecting in general.

Teade Punter is Knowledge Manager at the Embedded Systems Institute (ESI), the Netherlands. He received a M.Sc (Ir.) from University of Twente in 1991 and a Ph.D. from Eindhoven University of Technology in 2001. He worked at the Open University of the Netherlands, Kema Nederland B.V., Fraunhofer IESE and Eindhoven University of Technology. He has dealt with a variety of topics in software and systems engineering, e.g., software measurement and assessment. Teade's interests are in model driven engineering, integration & test and technology transfer.

John Vissers received his MSc degree in Mechanical Engineering at the Technical University of Eindhoven. He joined Vanderlande Industries in 2005 to work within the Research and Development department, where he shifted from mechanical engineer towards research engineer who has the responsibility to determine the feasibility of new product concepts. He is currently working as product engineer at the Systems Distribution department. His main interests are in defining, understanding, and analyzing system and product concepts.