Project Planning Support by Model Checking

Björn Axenath, Oliver Sudmann
Software Engineering Group, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany
axenath,oliversu@uni-paderborn.de

Abstract: Today's trend in software and system engineering is to utilize more specialized models. This model-based development approach makes a single engineering task more easy, as the engineer can focus on the particular aspect of the system, when working with one model. Though collaborations get more difficult, because more models have to be kept consistent. Unfortunately, the process support for model-driven development is still rather weak in today's development environments: static processes are supported, but this is insufficient for collaborations. We present a technique for project planning which utilizes relations between models and which uses a verification method to produce suggestion for the project plan, based on the current situation of the project.

1 Introduction

In the 90's, there was a research trend on process-centered development environments. Not much of the research results have found their way to practice: Today's favorite development environments do provide only very little process support. Even worse: often developers are of the opinion, that a process support would hinder their work.

We noticed, that there is still a gap between document-centric process models and today's model-driven engineering approach. A *document* is an item in a workflow, which defines a certain amount of information which can be processed in one solitary task. A *model* is a partial description of a system, in general an abstraction or an oversimplification. A process model describes the control flow, but it does not consider the dependencies among models explicitly.

Development environments, especially those for mechatronic products, use workflow engines to implement document-centric processes. But engineering tasks differ significantly from business process tasks, which are successfully supported by workflow engines. Engineerings tasks are characterized by making decisions on assumptions. How difficult this is, can be seen in Airbus' A400M [Flo09]. At its first release, it was 12 tons overweight, so that it had a shorted cruising range and lowered loading capacity. Also a flight maneuver, the steep approach, was impossible. Several components, like lowerable undercarriage, had to be excluded already, which had the drawback of a reinforcing element in the floor. To cope with these uncertain assumptions, changes are a daily occurrence.

A change consists of several steps. First of all, it has to be noticed that a change of a

model has an impact on other models. This sounds trivial, but it is the human nature to disregard these dependencies [Dör79]. Then, it has to be identified, which impact the change has. This impact depends on the current state of the project. Let us regard an example. Tests of mechatronic systems may take up to several month. Changes on an element which has already been tested would require a repetition of the test, which might be very costly or infeasible due to the delay. Finally, the change has to be integrated into the existing project plan, which takes the availability of developers and much more into account. All these steps are neither supported sufficiently nor integrated by today's development environments.

To identify the complexity in development processes of complex products, we analyze the characteristics of development process of mechatronic systems (Sect. 2). We will see that the model-driven development approaches complicates the collaboration among the engineers. Then, we identify use cases being more appropriate than today's process support. In Sect. 3, we describe our concepts which help to tackle the complexity of project planning. In Sect. 4, we shortly describe our prototype and the experience which we made. After we discussed related work in Sect. 5, we draw the conclusions of our work and give an outlook on the future work.[1]

2 Development Processes for Mechatronic Products

Mechatronic development, here as an example for a class of complex development processes, aims to create products which are a synergetic combination of the disciplines mechanical engineering, control engineering, and software engineering. In advanced mechatronic systems, like self-optimizing mechatronic products, the software connects several mechatronic components so that they are able to fulfill functions in a synergetic way. A mechatronic component is build of sensors, providing information from the physical work, actors, and a controller. Think about modern cars, in which the motor management, the steering mechanism and brakes control the stability altogether. Notice, mechatronic components are connected in two ways: on the one hand they are connected by physical effects; on the other they can be connected by networks.

As a matter of course, a complex system cannot be specified in one step. Several intermediate development goals have been introduced. Along the process, there are goals like principle solution, modularization, interface specifications of components, design specification, implementation specification, and many more. The order of these steps represents the overall development method. Former development process which have been taken from mechanical engineering started with the definition of the assembly structure; automation and software were added thereafter. More recent development processes, like the VDI 2206 [VDI04], treat the disciplines equally by having an active structure first, which covers all domains. Orthogonal to that, industrial standards, like the MDA method [BBI+04], propose to process the requirements of the target platform in an incremental

[1] This work was developed in the course of the Collaborative Research Center 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

manner. On the one hand side, a developer can now focus on the very specific topic, which the model has been developed for. This reduces the complexity of a single task, which has been part of a composed task before. On the other hand side, the more models are used, the more coordination among them is necessary. The complexity of the process resp. the collaboration rises. Nevertheless, the models are only loosely integrated and traceability among them is still on the research agenda [PBKS07].

The process of developing a mechatronic system will be illustrated by a brief example of a rail vehicle which can build a convoy with other vehicles as shown in Fig. 1. To master the complexity of the overall system, the system is modularized in subsystems. In our example the system structures is expressed by an active structure diagram [GFDK08a]. As depicted in Fig. 1, the active structure consists of two components, which are called system elements here. The Configuration Controller is responsible for the negotiations with other vehicles, if a convoy is build or not. The Distance Controller is part of the drive system and controls the systems position. The Distance Controller gets the current distance d_{cur} between two vehicles to keep the desired distance d^*. Thereafter, an interface specification has to be written for all system elements. The behavior of the Configuration Controller is modeled by an Hybrid State Chart as its behavior is discrete. The Distance Controller is modeled by a block diagram as its behavior is continuous. Here, both components have to refine the parameters d^*, which is modeled in the system structure on a conceptual level. For instance, the data type and the rate of change are not defined, but both have a significant effect on the system. Thus, the hybrid state chart and the block diagram are relate to each other by the parameter.

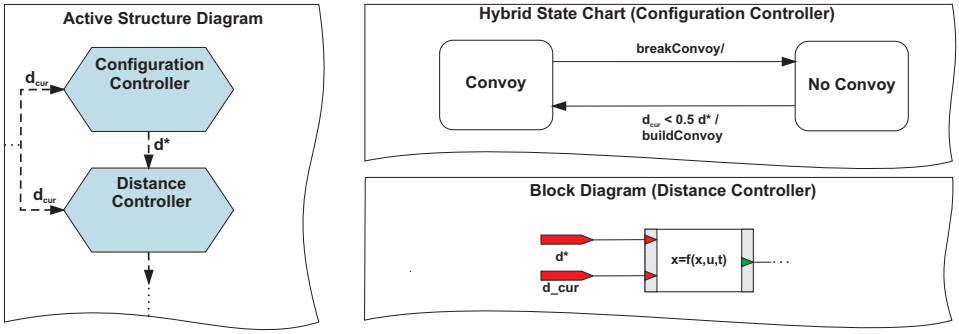


Figure 1: Fragment of a mechatronic systems structure and behavior specification

Furthermore, it has to be considered that engineers of different disciplines are involved, so that developers do not understand all dependencies of their model to models of other domains. Either specifications like the system structure which can be understood by all disciplines have to be used, or cross-discipline experts have to mediate. In our example, it would be not appropriate to keep consistency directly between the Hybrid State Chart and the Block Diagram.

Usually, mechatronic systems are modularized in several discipline-spanning components, which are developed by several teams working rather independent in parallel. Nevertheless, the models are not necessary developed at the same time. A change, for example in

the system specification, might result in different costs, depending on the number of dependent tasks which have already been finished. For mechatronic components some tasks, like quality assurance on a testbed, can become costly. A plain traceability analysis, which just shows the dependent documents of the change, is insufficient: The progress of the project has to be considered.

The result of a development process is influenced by the success of finding a solution with an acceptable cost-benefit ratio. The requirements and the final specification have to be fit to one another. For that, iterative processes can be applied. In iterative processes, the previously created artifacts are revised resp. changed, because of the experience from the previous iteration. The question is then: is it possible to do change now, or should it be done in the next iteration?

In summary, the solving of a development problem is split up into several, small modeling tasks, which are highly dependent. Though, developers cannot see or do not want to see these dependencies, and for project managers, it is hard to understand the dependencies due to a lack of knowledge. Thereby, the project manager is not interested in the dependencies, but in the resulting work.

We deduce the following two *use cases* with respect to process support:

1. *Impact Analysis:* A developer gets informed, if his work on a model has dependencies to other models or puts a deadline at risk due to dependencies.
2. *Process Synthesis:* As it happens in some domains, that developers do not follow the project plan, the project manager should get support to identify tasks to make the project consistent with the process model.

In these use cases, the development environment should identify all tasks which have to be done, and order them according to the process model. Thereby, it is sufficient identify a *possible impact* of an activity or to make *coarse suggestions* for project plans. The final project planning is left to the project manager as too many side constraints, like resources, social aspects, and so on, exist. Consequently, we are not interested to create a detailed and complex model of the process, which could consider the uncertainties in the execution of a task, for example.

3 Concept

To explain our concepts, we describe some preliminaries, mainly the formalization of the process model including the traceability links. Then, we explain the concept, which applies model checking to analyze the state space defined by the process model. Finally, we explain how this approach supports our use cases.

3.1 Preliminaries

We aim to perform a dependency analysis with respect to the process model. Before the analysis can be performed, the process model and its dependency to the data managed by the development environment have to be specified. Next, we describe the necessary steps in short. In doing so, we use a running example, which is a fragment of a development process of mechatronic systems [GFDK08b], and which was applied to create our introductory example.

First of all, the *document model* has to be defined. In general, the document types can be extracted from a reference model and have to be refined thereafter. A reference model of our example is shown in Fig. 2. In the first step, the system is decomposed. The decomposition creates a system structure. Then, for every component a sub process is created, of which the first step depicted. In this first step, the observable behavior of the components should be specified. Furthermore, the control structures which define relationships among the documents have to be identified, which we assume to be part of every development process model anyhow.

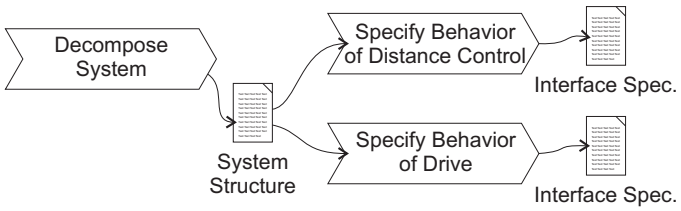


Figure 2: Process Fragment Sample

Secondly, the document types have to be refined. For every document, a model is defined which specifies the concepts which should be used to write the document. Notice, that the purpose of a document is to define the information which is necessary to perform a task. How this is done, depends on the product. In our example, the system is a mechatronic, self-optimizing system which is build up of mechatronic components. These components can be arranged in mechatronic function groups which are connected by information flow, material flow, and energy flow [Kal98]. The interface specification should specify the parameters which are described in the system structure on a conceptual level, e. g. physical dimensions and precision are not defined. Fig. 3 shows an excerpt of the project information model of our running example. The dashed box contains the documents of the process model. The Signal Checklist and the Signal Trace are specified in the next steps. Altogether, they are the *project information model*.

Thirdly, to enable traceability, the relations between elements of the project information model have to be defined. These relation also depend on the product. Nevertheless, domain reference models are available for several domains which define traceability relations (see e. g. [RJ01]). For our product we would like to enable traceability between the information flow which is defined in the system structure and the interface specification of mechatronic

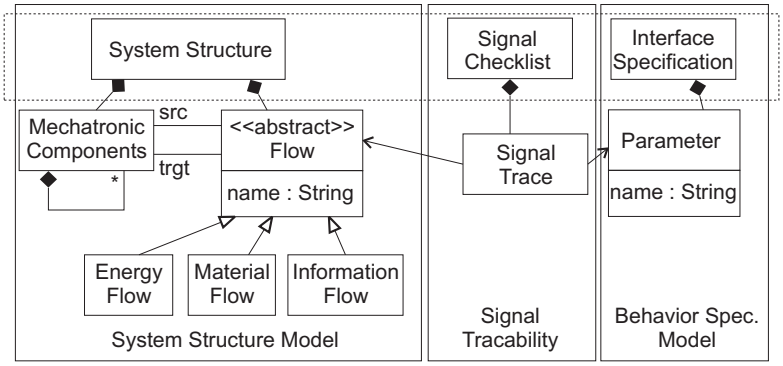


Figure 3: Project Information Model

systems. This is done by the *Signal Trace*[2]. For the process view, these links are also part of the process. So, they have to be part of the document model, and we define a new document type, the Signal Checklist. Often, they are used within documents from the quality assurance, because they do not specify the product itself, but describe its evolution.

In this model we take into account that several disciplines are working together. Traceability links should be defined only between concepts, which can be understood by the participating domains. In general, links between discipline-specific models are avoided and discipline-spanning models are used [GGS+07].

Fourthly, the project information model has to be implemented by some tools. In general, the available tools do not support the concepts modeled by the project information model, and the creation of tools which directly support the project information model is still costly. Consequently, more general tools are used. Examples are spreadsheet applications like Microsoft's Excel, generic UML tools for software components, and dataflow-oriented tools like Mathworks' Matlab with the Simulink Toolbox for signal processing. To integrate these tools, we use the ToolNet [ADS02] approach, which integrates the tools by adapters, that map the elements of the information model onto elements of *tool data model*. That is, the instance of the project information model is then materialized by the tool data. Traceability links which are not handled by any tool are stored in a link repository.

To make suggestions for the project planning, we need information from the previous project plan. A project plan schedules tasks and defines intermediate goals by milestones, which are measurable quality criteria on the system under development. A sufficient abstraction for these measures are processing states, like planned, finished, approved, or integrated.

[2] Actually, we define them by Triple Graph Grammars (TGG) to enable automatic model transformation and consistency checking [Sch95]. By knowing the artifacts, which have been checked in together and assuming that they are consistent, we can select an automatic repair action, which can be derived from a TGG. Despite this approach has some limitations, it is able to automate a significant amount of tasks.

3.2 Project Analysis

To implement the use cases, we have to identify certain paths in the process space. In both use cases, we have to search a path, which leads from the current situation to a given milestone. The impact analysis checks, if the next milestone is still reachable after a certain execution of a task. So, we have to analyze the complex state space which is defined by the processing states and its transitions given by classes of tasks. We apply model checking [CJGP99] for this analysis, because model checking provides adequate algorithms for this kind of analysis. Furthermore, we will see that temporal logics are an adequate formalism to define boundary conditions on the process.

The state space is build in the following way. We define for every document type its processing states. Transitions are given by tasks resp. their classes. In Fig. 4 we extended our introductory example by further quality assurance tasks on the documents: the system structure has to be checked; the interface specification has to be tested and checked according to some guidelines, thereafter it is integrated with other specifications. Then, for every document instance, an automaton is created. Finally, all are put in parallel. In doing so, dependencies of transitions have to taken into account. Transitions of the subsequent tasks are only allowed to fire, when the input documents have reached a certain state. When tasks process more than one document, the particular transitions have to fire synchronously. Additionally, we have transitions which result from change processes (drawn dashed). When a document is changed, then all depending documents are changed synchronously.

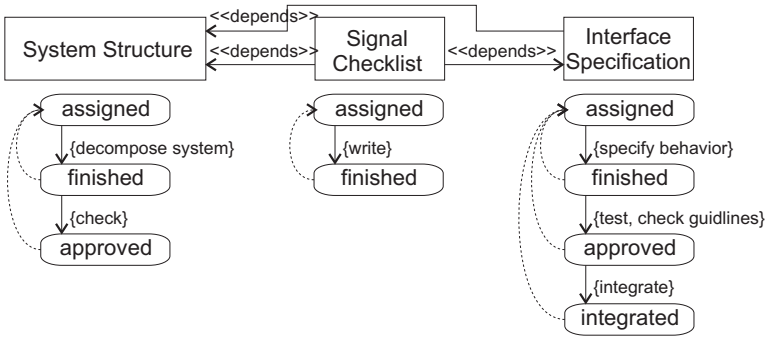


Figure 4: Definition of State Space

Model checking verifies a temporal formula on a labeled transition system M. In general, a system is checked weather a property is always fulfilled. If the property is not fulfilled, the model checker can produce a counterexample which describes a run of the system violating the formula. In contrast to this kind of analysis, we are interested in the reachability of a state, so that we need the path witness for a formula requiring the existence of a path. We check if the milestone, which is represented by propositions on a state, cannot be reached in future (F for future) starting from the current situation s:

$$M, s \models \neg F \,\mathrm{milestone}$$

To create suggestions for the project plan, the deadline and the duration of tasks are taken into account, which can be extracted from the project plan. When tasks have been iterated already, an estimation of next iterations duration can be determined by extrapolation of the previous durations. This requires timed model checking, which considers clocks. We use a global clock to measure the project time. For the processing times which are consumed by the tasks, we create a local clock on each automaton and define guards to keep the document in a state for the estimated time. The side effect of the usage of a deadline is, that the state space is reduced significantly.

The project management usually defines further boundary conditions on the project plan. For instance, certain documents should not be modified any more. This can only be modeled if we use a branching time logic, like CTL, as we search for one particular path: There exists (E for exists) a path on which the constrains are valid until (U for until) we reach the milestone. In formal terms, the *process synthesis* has to verify:

$$M, s \models E\,(\text{constraints}\,U\,(\text{milestone} \wedge t < \text{deadline}))$$

The *impact analysis* is formulated in the following way. Let s_1 be the state of a document after the intended task. It exists a path in which in the next (X for next) step s_1 is valid and from which a path to the milestone exists:

$$M, s \models EX(s_1 \wedge EF(\text{milestone} \wedge t < \text{deadline}))$$

The final concept of our approach is to control the process continuously, by monitoring the version management system of the documents. After every check-in an impact analysis is performed, if the next milestone can be reached.

Let us consider an exemplary situation in a project, which is illustrated in Fig. 5 as a Gantt Chart. At the deadline, work on Active Structure s, Hybrid State Chart $b1$, and Block Diagram $b2$ has to be finished. The process model of the example is defined as shown in Fig. 2. When at $t0$ a developer working on $b1$ creates an inconsistency with s, he gets the information that Task 1 is created. Additional work on $b2$ is not created, because work on $b1$ has not been started yet and even due to the delay of Task 1 it is sufficient time to finish before the deadline. Would this inconsistency occur at $t1$, the developer would be informed that, the impact of his work puts the deadline at risk, because according to the project plan firstly Task 2 and then Task 3 have been done.

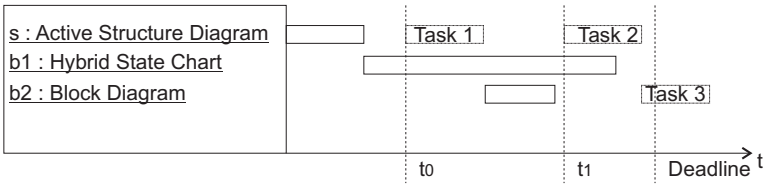


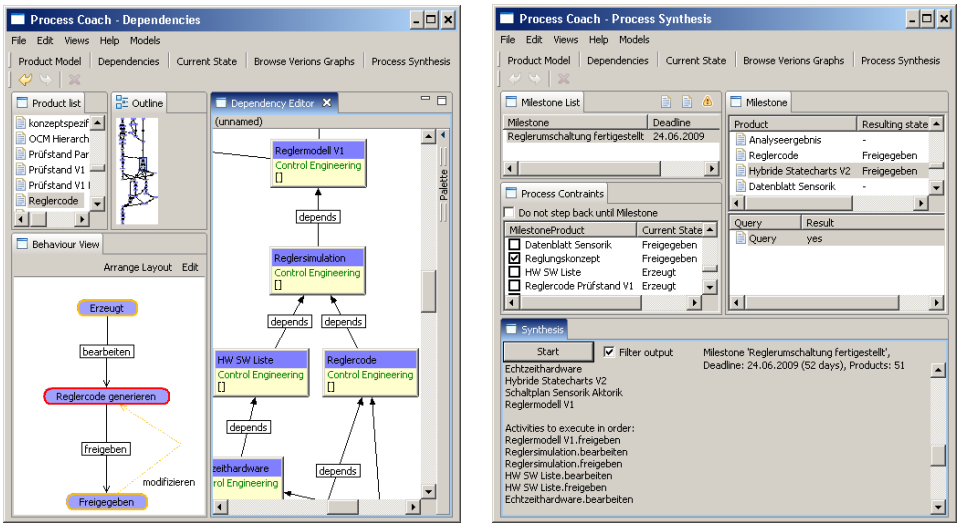
Figure 5: Gantt Chart

4 Implementation

We have implemented our concepts in a prototype called *ProcessCoach* which is integrated in the Eclipse IDE. The process model is defined by our own formalism, for which we build a graphical editor as shown in Fig. 6(a). The ProcessCoach uses the model checker UPPAAL (see www.uppaal.com) for the state space analysis. UPPAAL is able to search the shortest path, so that we are able to use it without any extension. The counterexample from UPPAAL is imported into the ProcessCoach and can be exported to Microsoft Project.

A critical issue for the application of model checking is that temporal logics can hardly be specified by users of a development environment. In the synthesis view depicted in Fig. 6(b), the project manager selects the documents from a table, defines their desired processing states at the milestone, defines a deadline, and further boundary conditions on the process. Here, he decides that the Regelungskonzept should not be set back in its processing state. The ProcessCoach creates the temporal formula internally, starts UPPAAL and shows the result at the bottom .

As a case study, we modeled a real-live development process of mechatronic system having more than 50 documents. All tests we made did not run into a state space explosion of the model checker. The results had been calculated within a few seconds on standard computers and notebooks.



(a) Document Model

(b) Process Synthesis

Figure 6: Screenshots of ProcessCoach

5 Related Work

Product data management system, like IBM's Enovia, provide workflow support and recently also integration of project planning tools. But they are only able to define and control activities in the planning tool which are then executed by their workflow engine. The process model is not taken into account, so that no planning support wrt. the necessary activities is provided.

Due to the complexity of the planning processes, mainly algorithm are applied which use heuristics, so that not all solutions are considered. Planning by model checking, which is able to analyze the complete state space, has been under research during the last ten years and a few of them also considered real-time model checkers. Most of these approaches aim to synthesize controllers, which are more complex results than sequences of actions for a project plan. Furthermore, we aim to identify the optimal sequence of activities to the particular milestone and repeat this procedure — a so-called interleaving planning. An early application of a real-time checker for planning was presented by Goldman et. al. [GMP00]. Their approach of domain modeling is similar to ours, but as they aim to use the model checker for a controller synthesis, their overall method, including the goal definition, is very different. Dierks [Die05] applies the real time model checker UPPAAL CORA to the general planning language PDDL [MC98]. UPPAAL CORA uses priced timed automata, so that it is possible to use further variables for optimization [BLR05]. We only use time to restrict the state space so that we are able to omit the pricing functions. In addition, we suppose that we are able to check bigger models, as we can use optimizations based on our domain model as we do not use the general planning language PDDL.

Furthermore, it seems to be an interesting idea to map graph grammars to action planning as it has been suggested by Edelkamp et. al. [EJL05]. This would enable us also to analyze dynamic document models, but due to the complexity, we suppose that we would not be able to check realistic examples.

6 Conclusion and Future Work

We have systematically analyzed which advanced use cases of development environments are necessary and feasible to support development processes of one class of complex, technical products. We improved the traceability analysis, because we took the process model into account, so that the impact of changes can be estimated by tasks instead of just listing depended artifacts. By our use cases, it is more easy to motivate developers to generate traceability data, because they benefit from it now. The explicit specification of the project information model starting with the document model bridges the gap between document-oriented processes and model-based development.

The benefit of this method is hard to determine. On the one hand side, the presented approach requires some effort for to the process modeling and for the tracking of changes. For some product classes, especially for complex or safety critical systems, these tasks are already done. Unfortunately, the particular tools are not integrated, so that also additional

effort on the integration becomes necessary. On the other hand side, the major advantage of the impact analysis and the process synthesis are an acceleration of the process. The value of the acceleration is hard to be estimated, as it depends on various factors — up to the assumed release dates of competitors. So this method might be suited for companies which have mature processes and use a highly integrated development environment.

We have applied model checking to implement the use cases. By using this technique, we achieved an improved quality of the process suggestions, because we are able to search the complete state space and do not use any heuristics. However, the user of the development environment is not annoyed with the formalism, although we have introduced an interface to the model checker based on temporal logics.

Our case study uses real-life processes, but the tool has not been tested during a project by the developers. To provide a comprehensive prove of concept, we have to analyze not only the technical details, like scalability, but also the usability. We expect, that we can also derive from a usability report further use cases for process support in model-driven engineering.

Up to know, our document model is static. In the next step, we will apply infinite state model checking methods, to consider that further documents are created. Moreover, our future work is motivated by the idea of a development environment with a *self-optimizing process*. That is, we want the development environment to be able to adapt the process' goals depending on the development situation and accordingly change the process model. Before being able to correctly change the process model, we need to be able to rate the current situation of the development. Therefore, we have to extend our approach by considering measurements about the models' completeness related to a processing state. For instance, we want to use test results or the completeness of traceability links for that.

References

[ADS02] Frank Altheide, Dr. Heiko Dörr, and Andy Schürr. Requirements to a Framework for sustainable Integration of System Development Tools. In *Proc. of the 3rd European Systems Engineering Conference (EuSEC)*, pages 53–57, 2002.

[BBI+04] Grady Booch, Alan Brown, Sridhar Iyengar, James Rumbaugh, and Bran Selic. An MDA Manifesto. *MDA Journal*, pages 2–9, May 2004.

[BLR05] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal Scheduling using Priced Timed Automata. In *Proceedings of ICAPS'05 Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, pages 1–8, 2005.

[CJGP99] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[Die05] Henning Dierks. Finding Optimal Plans for Domains with Restricted Continuous Effects with UPPAAL CORA. In *Proceedings of ICAPS'05 Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, pages 1–10, 2005.

[Dör79] Dietrich Dörner. *Problemlösen als Informationsverarbeitung*. Kohlhammer, Stuttgart, 1979.

[EJL05] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch Lafuente. Action Planning for Graph Transition Systems. In *Proceedings of ICAPS'05 Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, pages 1–9, 2005.

[Flo09] J. Flottau. Mission Impossible. *Süddeutsche Zeitung*, 20.1.2009.

[GFDK08a] J. Gausemeier, U. Frank, J. Donoth, and S. Kahl. Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus – Teil 1. *Konstruktion*, Juli/August, Heft 7/8:59–66, 2008. Springer-VDI-Verlag, Düsseldorf.

[GFDK08b] Gausemeier, J., Frank, U., Donoth, J., and Kahl, S. Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus – Teil 2. *Konstruktion*, September, Heft 9:91–108, 2008. Springer-VDI-Verlag, Düsseldorf.

[GGS+07] Jürgen Gausemeier, Holger Giese, Wilhelm Schäfer, Björn Axenath, Ursula Frank, Stefan Henkler, Sebastian Pook, and Matthias Tichy. Towards the Design of Self-Optimizing Mechatronic Systems: Consistency between Domain-Spanning and Domain-Specific Models. In *Proc. of the 16th International Conference on Engineering Design (ICED)*, pages 1–12, Paris, France, 8 2007.

[GMP00] Robert P. Goldman, David J. Musliner, and Michael J. Pelican. Using Model Checking to Plan Hard Real-Time Controllers. In *In Proc. AIPS Workshop on Model-Theoretic Approaches to Planning*, 2000.

[Kal98] Ferdinand Kallmeyer. *Eine Methode zur Modellierung prinzipieller Lösungen mechatronischer Systeme*. PhD thesis, Universität Paderborn, 1998.

[MC98] D. McDermott and AIPS-98 Planning Competition Committee. PDDL - The Planning Domain Definition Language. Technical report, 1998.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71, Washington, DC, USA, 2007. IEEE Computer Society.

[RJ01] Balasubramaniam Ramesh and Matthias Jarke. Toward Reference Models for Requirements Traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001.

[Sch95] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *WG '94: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163, London, UK, 1995. Springer-Verlag.

[VDI04] *Entwicklungsmethodik für mechatronische Systeme*. Beuth-Verlag, VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, 2004.