# Multidisciplinary Modeling

**Current status and expectations in the Dutch TWINS consortium**

**Frank P. M. Stappers, Lou J. A. M. Somers, and Michel A. Reniers**

Department of  and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513
5600 MB Eindhoven
The Netherlands
{F.P.M.Stappers, L.J.A.M.Somers, M.A.Reniers}@tue.nl

**Abstract:** The TWINS project addresses industrial hardware-software co-design product development. In search for a multidisciplinary modeling solution, knowledge is required about the current practice of multidisciplinary development in industry. By analyzing user requirements, use cases, and applied tools and techniques of six industrial partners, this paper presents the current status of multidisciplinary development in the Dutch consortium. The analysis is used to improve a successful collaboration between the industrial partners, and it also identifies the challenging topics that need to be explored within a multidisciplinary environment.

**Keywords:** Multidisciplinary modeling, current practice, overview, system development, co-design, industry, tools, methods, techniques

## 1. INTRODUCTION

Many systems developed today are a combination of hardware and software (for example printer machinery, folding machinery, in-line manufacturing systems, or consumer electronics). To gain new customers and satisfy current customers, the reliability of these systems needs to be as high as possible. Meanwhile, features are added for future releases and the development time has to decrease. As a result, the complexity of a system increases, which creates a challenge for both hardware and software development, especially in the development trajectory where the different hardware, software, and electronic parts are merged. To tackle these challenges, companies need to apply techniques in which engineers from different disciplines can cooperate.

Unfortunately, the techniques used nowadays have a monodisciplinary focus. As a rule, each discipline (electrical, software, mechanical, optics, etc) has a separate development trajectory. When the components meet their finalization, they are integrated into a highly complex system. However, during development there is hardly any formal information which can be exchanged between the different disciplines. This often results in unforeseen integration problems, functional requirements that cannot be satisfied, or in no system at all.

In industry, the feeling resides that it is now time to address the separate monodisciplinary development trajectories and convert them into a single intensive multidisciplinary development trajectory. With the help of a framework supported by various disciplines, we aim for a better development trajectory that can be used for simulation, verification, generation, and the exchange of information.

To gain insight in the current development practices, we have performed an industrial inventory as part of the ITEA2 [1] supported TWINS project [2]. The focus of this project is “Optimizing the HW-SW Co-design flow for software intensive system development” with different partners located throughout Europe. In this project, the Dutch partners have created a shared vision in which they want to cooperate in “Service and Software Creation” and “System Engineering”. With the help of multidisciplinary models, they want to develop modeling, simulation and verification techniques for the successful integration of independently developed electronic, mechanical and software components.

This paper presents the outcome of a study among the different partners in the Dutch consortium. The paper describes the common needs of the different industrial partners and the solutions to fulfill them. It also describes the methods that have been used to obtain the information. Moreover, an overview is given about current practices, methods and tools used by the industrial partners.

## 2. PARTNERS

The Dutch consortium is composed of six industrial partners and one knowledge partner. The industrial partners consist of three multinational enterprises and three small and medium sized enterprises:

- LogicaCMG is a venture specialized in developing value-added solutions and delivering custom-made systems.
- Neopost is a provider of mailing and shipping solutions.
- NBG provides services in the field of electronics, embedded software and PC-software for industrial and medical applications.
- Océ develops products like printers and delivers services related to the effective exchange of information.
- OTB Display is a supplier of in-line organic light-emitting diode (OLED) production equipment for mass production volumes.
- Sioux Embedded Systems is a company specialized in the development and support of software for embedded systems.

The knowledge partner is the Technical University of Eindhoven. The partner is formed by a joined collaboration between the departments of “Mechanical Engineering” and “Mathematics and Computer Science” in cooperation with the universities Laboratory of Quality Software (LaQuSo).

## 3. COMMON GOALS

A common denominator in the goals of the different partners is the use of models to improve the current development methods, thereby reducing time-to-market, lowering costs, and increasing product quality. A *model* is a description of one or more *concerns* of the system at some (deliberately chosen) level of abstraction. The models can be used for amongst others documentation, analysis, simulation, testing, or verification. Concerns that are traditionally described in models are often functionality, external or timed behavior. Presently, each discipline involved in the development trajectory of a system uses its private modeling techniques. Some shortcomings of monodisciplinary modeling for embedded systems are:

- the lack of understanding other disciplines,
- different incompatible models and non-exchangeable model formats,
- the difficulty of having correct and verifiable models over the disciplines.

As a (partial) solution to the problems mentioned above, literature mentions *multidisciplinary modeling* [3,4]. In multidisciplinary modeling models are used in such a way that they

- cross-cut the concerns that are native to disciplines in isolation, or
- are native to a discipline but are kept consistent with other models.

In academia, tools related to modeling are often only practiced on small toy-based problems. The problems in industry are much more complex. Unfortunately, when scaling the toy-based problems to industrial problems, the models cannot be used due to various complications. The desire is to make the academic tools and techniques for modeling applicable in industry.

To maintain close links with the various disciplines, information exchange must be clear, easy, and unambiguous. To fulfill this need, the multidisciplinary models need to be understandable and easy to create. Most partners in the consortium use a specific tool or set of tools to support their system development. Some industrial partners also support clients in their system development. Given this information, they want a framework which is consistent with the multidisciplinary model and the tools used.

Using a *Model Driven Architecture (MDA)* [5] and *Model Driven Engineering* [6] it is possible to construct embedded control software by means of models in a model driven development environment. *Component Modeling* [7] also creates a reference architecture for product lines and facilitates release management.

Often, the physics within a system are not completely understood. As a result, the integration of hard- and software components consumes a lot of time. With the help of models it is easier to understand the physical behavior of a system and therefore less time needs to be spent on the integration of components. Formal models may allow the design of a controller by using control synthesis [8]. This creates consistency between the behavior models, the mechanical models, the electrical models, the software design models, and the implementation.

The industrial partners also opt for a *Model Driven Test Environment* [9,10]. In current practice, tests are performed on a (partially) implemented system. With the use of a model driven approach it is possible to describe and execute the behavior of a system, prior to build. A virtual test bench (a real-time interactive framework containing a set of virtual hardware devices on which tests and experiments can be conducted) allows the observance of the behavior of a system without the chance of damaging it. Therefore, current techniques for simulation and testing need to be enhanced to encompass a larger part of the system functionality, to provide a better visualization for the results gained while testing, and generation of tests for hardware and software components.

## 4. USER REQUIREMENTS AND USE CASE ANALYSIS

To address the goals in TWINS, all partners have created a set of user requirements associated with a set of use cases. From the requirements and use cases we have tried to determine the cohesion between the Dutch partners and address the topics on which cooperation is possible.

A *user requirement* describes a method, process or tool that an industrial partner wants to improve or wants to have, during the TWINS project. A *use case* is a description of a sequence of events that needs to be performed and documented to fulfill a user requirement. For the analysis the following user requirement attributes are relevant:

- The **description** gives a summary of the purpose of the requirement. It contributes to a better understanding of the goal of the user requirement.
- The **source** describes the origin of the requirement, provides a link to the information on which it is based and helps to understand the existence of the requirement. The source information can also indicate the importance of a requirement.
- The **rationale** describes the justification of the requirement. It describes the value for the stakeholders and the business goals which should be satisfied. The rationale is the link between the problem and the solution domain.
- The **priority** describes the importance of the requirement. The prioritization is taken into account when determining the cohesion.

When analyzing the use cases the following attributes are relevant:

- The **name** of a use case often hints at the goal that a partner wants to satisfy.
- The **user requirements** that are fulfilled when carrying out the use case. If shared requirements are fulfilled in different types of use cases, they will not contribute to a better cohesion.
- The **goal description** summarizes the goal satisfied when carrying out the use case.

The analysis of the use cases and user requirements yields five relevant topics:

**Testing** is a process of technical investigation, performed on behalf of stakeholders, that intends to reveal and measure quality related information about a product with respect to the context in which to operate. The analysis shows that the topics are constrained to the correctness, completeness, and security of a system.
To gain confidence in developed products, two partners intend to develop a test environment architecture in which they can virtually test their controller by a modeled environment. Via uniform test procedures and executable models, they want to test their systems for functional properties.

**Verification** is the process of technical investigation that intends to prove the correctness of a design or a system conform a given specification. At least one partner is interested to verify the interfaces of components at a semantic and syntactic level. By performing consistency checks it is deemed possible to achieve a decomposition of a system, in which the correctness can be assured. With the help of formal methods and models, they want to create a test case generation technique, to generate tests for individual components.

**Simulation** is the process of mimicking the functional behavior of a physical or abstract system by means of a (mathematical) model. With the help of simulations it is possible to reveal errors w.r.t. the used model. Prior to construction, it allows a user to study selected aspects, explore hazardous behavior, or it can contribute to a better understanding of the behavior of a complex multidisciplinary system.

Several partners are interested in visualization and simulation techniques based on model-based designs. With such visualization techniques, it should be possible to virtually simulate and observe the behavior of products or production systems. In this way they want to better understand the behavior of a system and improve engineering choices in the early stages of development. Another technique incorporates Hardware-In-the-Loop / Software-In-the-Loop simulations (HIL / SIL) [11,12], by which the partners want to eliminate problems related to the late realization of hardware (early integration) or the limited number of hardware resources (a virtual hardware environment).

**Model Driven Development (MDD)** [13] is a paradigm to generate code and realize designs effectively based on a model. With the reuse of models and the derivation of code, it is possible to reduce cost and shorten the development time. A couple of partners have requirements that ask for a *Model-Driven Development paradigm*. They want to create a language or architecture, for which it is possible to generate code and derive documentation based on a model. By using this technique, they believe that it is possible to save time during development, decrease the time to market, and improve the quality of their systems.

**Multidisciplinary development language** (MDDL) is a language in which various monodisciplinary modeling languages are combined into a universal language, for which it is possible to easily exchange and share information between disciplines by using the proper view on the MDDL.
One industrial partner wants to focus on the development of a MDDL. Without a MDDL, inefficiency in the system model activity is introduced, because various modeling approaches need to be translated into the modeling approaches of other disciplines.

## 5. CURRENT INDUSTRIAL PRACTICES

To determine the compatibility between partners, we tried to collect the methods, formalisms, and tools which the industrial partners currently use in their multidisciplinary system development. The questionnaire (and follow-ups in plenary sessions) we used focuses on three different phases: the system design, integration, and verification phase (discussed in more detail in the next pages). The partners believe that these phases encompass the most important multidisciplinary developments.

A good design of a highly complex multidisciplinary system increases its chance of success. When multiple disciplines develop a system, it is very important that each discipline can fully express its requirements and knowledge in a design, while other disciplines are still able to understand the design. Usually, each discipline creates its own designs and by means of (relatively short) meetings information about these designs is shared with other disciplines. As a result, problems may occur in the development, because mistakes and mismatches are easily overlooked.

A problem during the integration of subsystems originates from having multiple different disciplines. Already when integrating different components from a single discipline, difficulties occur which are related to the misunderstanding of concepts and the lack of non-formal designs. When having multiple disciplines it becomes even harder, because the misunderstanding between the disciplines also needs to be taken into account.

Another problematic issue is the verification of the correctness (of the design) of a multidisciplinary system. For rather small concurrent systems it is already hard and sometimes impossible to prove the functional correctness. When different disciplines are involved, the verification becomes even more complex. We have not only to deal with discrete events; we also have to deal with other aspects like physics.

### 5.1 Design phase

Without a good design it is merely impossible to develop a highly complex multidisciplinary system. Each design focuses on a different aspect of the development and contains information at different levels of detail about the development. We can distinguish the product development process, the system design, and the decomposition.

**Product development process.** The design of the product development process describes the road map for the development of a multidisciplinary system. The questionnaire shows that the partners use two different flavors, based on the classical V-Model [14] and the CAFCR [15] method.

The partners who use a V-model extend their designs with separate tracks running in parallel that concern the design and creation of tests, quality assurance, and configuration management. Although these extensions are not mentioned in the classical V-model, in practice these tracks exist tacitly. CAFCR is a method for embedded systems architects who use a multi-view architectural method, aimed at reconciling the necessary high level and detailed design aspects. CAFCR creates views for each of the architectural decompositions and these views are concurrently developed.

An explicit model of the product development process contributes to the harmonization between disciplines, due to easy accessibility and readability. It helps to manage specific field related knowledge and can be used to track and trace the development of a system. However, when inconsistency between the product development design and the actual execution is encountered, it is hard to remove it. Furthermore, the absence of visualization makes it sometimes difficult to understand the exact progress of the development.

**System design.** The system design describes the specification of a system that should comply with the requirements. The design is used to determine and clarify the complete behavior of a system. It serves as a communication aid towards stakeholders, assists in project management, and is used for risk reduction. To ensure a correct design, it is reviewed by the stakeholders, which are mainly the engineers involved. If a system is build for external usage, the external stakeholders are included as well.

The requirements can be split into functional and non-functional properties. A functional property refers to the overall physical behavior or performance of a system, and reflects on the various interactions that the system can perform and/or is influenced by. A non-functional property captures the constraints on the behavior and describes the overall attributes of a system [16]. Based on the questionnaire, we conclude that the most important non-functional properties can be put into three classes: performance, reliability and safety.

**System design decomposition.** The system design decomposition describes the decomposition of designs and their mutual dependencies. Usually the coordinating design is decomposed into five smaller pieces: the system requirements, a design for mechanical behavior, a design for static physical properties, a software design, and an electronic design (Figure 1). Almost all of the industrial designs have no formal commonality, which makes them vulnerable to flaws and unforeseen incompatibilities.

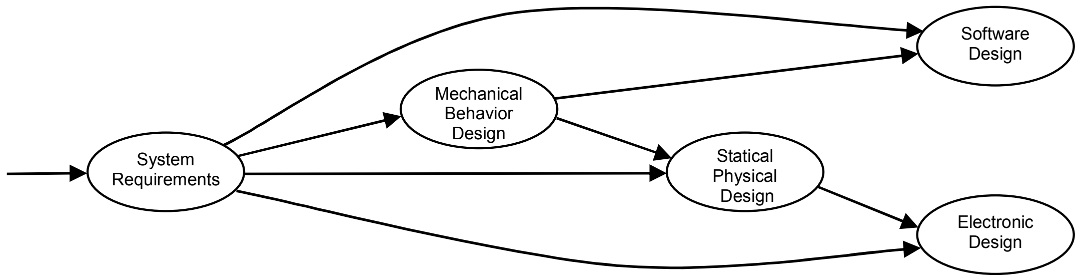

**Figure 1**: The relationship between the various designs.

*System requirements* describe the requirements to which the system must comply. They are acquired by talking to the stakeholders and identifying their needs. When the system requirements are defined, each system requirement is refined until it becomes discipline specific. The questionnaire shows that one partner applies a strategy to convert a problem into a set of requirements by using *problem frames* [17,18]. This method is based on analyzing the problem and split it into smaller parts. A part (including the solution) is called a frame. If the sub-problem relates to a frame, a similar solving strategy is applied. If a frame does not match an existing frame, it is either split into smaller frames or if the frame is small enough a new solution is applied.

*Mechanical behavior designs* describe the physical behavior of a system. The essential behavior (after abstraction) can be classified into three different types: systems in which discrete events are relevant, systems where only continuous changes are relevant, and hybrid systems where both continuous changes and discrete events are relevant. In TWINS, the industrial partners focus on discrete event and hybrid systems. Models for discrete event systems can be specified in discrete event specification languages (such as [19,20]). We see, however, that partners only use formal languages and modeling techniques when they are really needed, e.g. to get a certification that indicates (high) reliability level. A hybrid system requires, next to a discrete event specification language, an algebraic environment to describe continuous physical phenomena. With tools like MatLab/Simulink [21] it is possible to model, simulate, and analyze dynamic systems. Especially in the fields of control theory and digital processing these tools are used. The disadvantage of these designs is that they become obsolete at the point when they are translated to software and hardware.

*Software designs* describe the functional properties for software. The questionnaire shows that for software design *Unified Modeling Language* (UML) [22] techniques are used. Because of its graphical notation the language is accessible, which results in swiftly acquired designs. However, UML models are hard to reuse if they are not complete. The partners also point out that it is impossible to assess the software performance. Another disadvantage is absence of a formal principle. This makes it is hard to perform formal simulation and formal verification on the models to predict or guarantee certain behavior.

*Static physical designs* describe the physical non-functional properties a system. These are often mechanical designs, constructed with the help of Computer-Aided Design (CAD) [23] tools. Depending on the tools used, they provide functionality in the range from 2D vector based drafting systems to 3D solid and surface modelers. The designs provide a clear view about the mechanical construction and are directly usable for implementation or subcontracting. It is, however, difficult to see the dynamic properties, like the stiffness or the fragility of a material. To view these properties it is possible to extend CAD designs with *finite element method*s (FEM) [24]. To determine these properties, experts are consulted, which make it an expensive task. CAD tooling can also be used to design circuits [25], however this method is not very popular in the field of electronic engineering. CAD designs are primarily used to determine the constraints for the electrical engineers.

*Electronic designs* describe the electronic components that are used, their positions and the relationship to other components. To realize the designs, special *Printed Circuit Board* (PCB) tools are used. These tools include rules for checking the design and provide functionalities for board fabrication in an assembly process. Often they also provide functionality for auto routing and trace optimization, which shortens the layout time. With the help of other tools it is possible verify the design of a circuit, by means of simulating signals of analogue circuits [26]. Some tools also provide the possibility to verify that a circuit meets the non-functional properties, given a VHDL description [27,28]. In practice, these techniques are heavily dependant on the tools and a great deal of expertise is needed to master them.

**5.2 Integration phase**
For the integration phase, the industrial partners pointed out four difficulties.

Firstly, each system is different from other (predecessor) systems, because each system uses different components or the relationship between components is different.
Secondly, the semantics and the definition of the interfaces need to be accurate and complete. Even if components of the same discipline are integrated with each other, it already leads to complications. If components of different disciplines are integrated, an additional factor occurs: the gap between the different disciplines.
Thirdly, components can be connected to other components at different levels. Each discipline creates a hierarchy in its components. Therefore, if the system design has not been well constructed, it is possible that the levels of a component connect to different levels of another component, which creates a dependency web (Figure 2).
Fourthly, many of the occurring problems are the result of practical unforeseen incidents during the realization of a system.

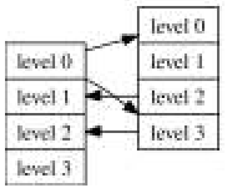

**Figure 2**: Example of a hierarchical mismatch between different components

For a smooth integration phase, the partners use two different approaches, namely an incremental sequential development process and a parallel development process with an iterative integration.
In the incremental sequential development (Figure 3), components of the different fields are developed sequentially, whereas components of the same discipline are developed in parallel. When the development of a discipline reaches a certain phase, the information in this discipline is transferred to the depending discipline. This creates a mutual dependency. Here, the software is dependent on the electronic components and the

electronic components are dependent on the mechanical components. New mechanical improvements appear after the software integration has been completed.

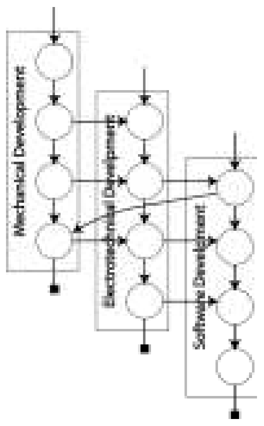

**Figure 3**: Incremental sequential development

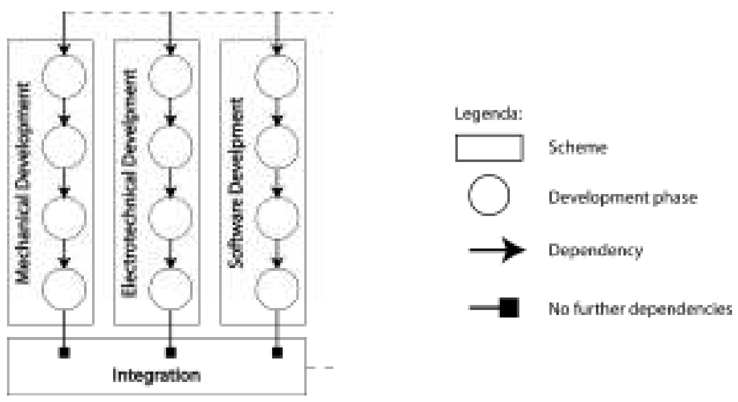

**Figure 4:** Parallel development

In the parallel development process all components are developed in parallel independently from other disciplines. Each discipline develops its components according to a fixed scheme. Figure 4 shows that when all components are finished, they are integrated into a prototype. By means of iterations the prototype is converted to a system.

In both approaches, components are tested thoroughly and separately from each other. If a component passes a test, it is integrated into the (prototype) system. If a component fails to pass a test, it will be changed in order to pass. The designs for the fixed or changed components are adapted in an ad-hoc process afterwards.

**5.3 System verification phase**
The system analysis or verification phase is the phase in which the system is tested thoroughly. In TWINS, the partners are primarily interested in the functional correctness of a system. To minimize the number of faults or to guarantee the correctness of a system, the industrial partners use techniques that can be categorized into three different types: model design level, integration level and realization level.

**System design level.** All industrial partners verify their system designs with informal techniques. They perform manual checks for consistency or perform (visual) simulation, to see if the design behaves as expected or apply falsification to find incomplete requirements.

In contrast to informal verification techniques, formal verification techniques are applied rarely. Often such techniques are not very accessible (it takes a lot of time to master methods like [29,30]) and the tools have limitations (e.g. state-space explosion when model checking). In spite of the disadvantages, the industrial partners admit the importance. The ones that have applied formal techniques could resolve flaws in their designs.

**Integration level.** At the integration level, simulations and test benches are used to decrease the development time, to reduce the chance of failure and to test whether components can cooperate with each other. Simulation is used for various purposes. Some use it to confirm presumptions on the systems behavior, whereas others use it to find errors and explore hazardous situations. The environments are not generic. This means that each time a simulator is needed, it is created from scratch. Often HIL/SIL simulators are created in a programming language like C++ [31] or are embedded on a FPGA.

**Realization level.** At the realization level, tests are executed with scenarios derived from the use cases. These tests have an informal character. A scenario consists of a set of conditions or variables and a set of tasks that must satisfy a requirement when being executed. Often, observing the results from a scenario is a complex task. In that case, special equipment is used to log and analyze the data. If a scenario does not reveal errors in a system, it contributes to the faith of having a system without flaws.
To gain a higher degree of confidence, some partners use conformance testing [32,33] to positively verify the user requirements. In contrast to scenario based testing, this method has a formal character. By performing conformance testing is possible to prove the correctness of a system w.r.t. the given specification. Because it is performed on a real system, the execution of a single test can take a considerable amount of time. When the

system is large and complex, it can even become unmanageable. Furthermore, it is also possible that the hardware gets damaged, when the system gets into an unsafe state.

## 6. TECHNOLOGICAL CONTRIBUTION

Based on the analysis of the use cases, the user requirements, and the questionnaire, the technology partner proposes to deploy the following activities:

**Exchange of formalisms.** Among the industrial partners there exists a need to model and analyze components or systems using different tools, techniques and formalisms over different disciplines. In a multidisciplinary formalism it is important that monodisciplinary models can be shared, exchanged and used, by creating methods that allow the systematic exchange of information between disciplines. With this starting point, the knowledge partners want to investigate ways to exchange (or combine) information from one formalism to another directly or indirectly via an interchange format like [34].

**Formal verification for multidisciplinary systems.** To ensure correct behavior on a multidisciplinary system, automated formal verification techniques are needed. The knowledge partners want to investigate methods at the modeling level and the implementation level. At the modeling level, one can use model checking, where techniques need to be developed that can cope with larger (timed) discrete event system or even hybrid systems. The idea is to accomplish this via extensions, adaptations and development of tools for the mCRL2-formalism [35]. At the implementation level, it is possible to conduct tests to see if a formal specification meets the actual implementation of a component or system [36]. Based on formal methods interacting with an actual implementation of hardware, it should be possible to verify the correctness of certain properties, with the help of automated forms of testing.

**Controller synthesis for multidisciplinary systems.** Supervisory control theory of Wonham-Ramadge [37] is a controller synthesis methodology for discrete event systems, whereby the controller is automatically synthesized from the formal specification of the control objectives. The obtained controller is correct by construction, i.e. it satisfies the control objectives. The advantage of the method is that no further verification of the correctness of the closed loop system is required. It is planned to investigate the possibility of applying supervisory control methodology to hybrid models and the integration of the methodology with existing formal hybrid modeling tools.

**Visualization of multidisciplinary system models.** Multidisciplinary systems describe complex concurrent behavior. A visualization often contributes to a better insight in the behavior of a system. Therefore, the technology partner proposes to investigate techniques that can lead to a better understanding of the behavior of a system. A visualization technique, based on formally defined models, should be compatible with the tools used in industry and open new techniques for analysis.

## 7. RESULTS

From the use cases and user requirements we have derived a matrix, which shows the cohesion between the industrial partners (Table 1). The lines of attentions show that every industrial partner can cooperate and share knowledge with another industrial partner.

| | Partner 1 | Partner 2 | Partner 3 | Partner 4 | Partner 5 | Partner 6 |
|---|---|---|---|---|---|---|
| Test Environment | X | | | X | | |
| Verification | | | X | | | |
| Visualization | | X | X | X | X | |
| HIL/SIL | X | X | | X | X | |
| MDD | | X | | | | X |
| MDDL | | | | | | X |

**Table 1**: Commonalities among industrial partners based on the most important user requirements and use cases. The industrial partners have been placed in random order.

The questionnaire shows that the industrial partners use a broad range of tools for the multidisciplinary system development. Many of these tools strongly depend on the industrial field in which the company operates.

When examining the tools, we see that most of the tools are used to create monodisciplinary designs or to support the implementation process. Although a variety of tools are used, we see that similar methods are applied to create the designs for the different disciplines. The designs support engineers in their development choices, but are hardly used for the generation of interfaces and the behavior of components.

Many of industrial partners use discipline specific tools which are vendor locked. This means that the partners are dependent on a vendor for a service or a product. When developing a multidisciplinary framework for the industrial partners, these locks could cause a problem.

The creation of a design or model for verification purposes requires specific knowledge (purpose, modeling techniques, etc) and time. To create a design, often experts are consulted, which frequently has an informal character. Therefore it is hard to verify the correctness. Also the reuse of the designs is hard, because knowledge is lost (e.g. the expert has left) or the purpose has changed. It often happens that if the goals change, the entire design needs to be reconstructed, because the design does not have the required modularity level. We see that the absence of formal multidisciplinary design creates a challenge when integrating the components of the disciplines.

To improve the integration phase, a lot of time is spent on testing and simulation. These techniques are also applied to see if a multidisciplinary system meets the system requirements. With a multidisciplinary design or model it is imaginable that effort is reduced and development is shortened.

Based on the analysis of the questionnaire, we can conclude that there exists cohesion at the level of the applied methods and less on tools. Especially in the field of software development, similar methods are applied. Based on the outcome of the entire analysis, we do not see insuperable incompatibility between the industrial partners that might prevent a successful collaboration in search for a multidisciplinary modeling framework.

## REFERENCES

[1] ITEA: http://www.itea2.org/ (2008) Visited: 14-January-2008

[2] TWINS: http://www.twins-itea.org/ (2008) Visited: 14-January-2008

[3] Macmillan, S., Steele, J., Kirby, P., Spence, R., and Austin, S.: Mapping the design process during the conceptual phase of building projects. Eng. Construction and Architectural Mgmt **9** (2001), pp. 174-180

[4] Scholten, H., Kassahun, A., Refsgaard, J.C., Kargas, T., Gavardinas, C., and Beulens, A.J.M.: A methodology to support multidisciplinary model-based water management. Environ. Model. Softw. **22**(5) (2007), pp. 743-759

[5] Pahl, C.: Semantic model-driven architecting of service-based software systems. Inf. Softw. Technol. **49**(8) (2007), pp. 838-850

[6] Kent, S.: Model driven engineering. In Butler, M.J., Petre, L., Sere, K., eds.: Integrated Formal Methods, Third Int. Conf., IFM 2002, Turku, Finland, May 15-18, 2002. LNCS 2335, Springer (2002), pp. 286-298

[7] Pavel, S., Noyé, J., Poizat, P., and Royer, J.C.: A Java implementation of a component model with explicit symbolic protocols. In Gschwind, T., Aßmann, U., Nierstrasz, O., eds.: Software Composition, 4th Int. Workshop, SC 2005, Edinburgh, UK, April 9, 2005, Revised Selected Papers. LNCS 3628, Springer (2005), pp. 115–124

[8] Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. Siam J. Control and Optimization **25**(1) (1987)

[9] Calame, J., Dai, Z.R., Ioustinova, N., v.d. Pol, J., and Swinkels, R.: Definition of methods for automated test generation for TTCN3 based test systems. Technical report, TT-MEDAL (2005)

[10] Ruuska, P. and Kärki, M.: Ttcn-3 language characteristics in producing reusable test software. In Software Reuse: Methods, Techniques and Tools: 8th Int. Conf., ICSR 2004, Madrid, Spain, July 5-9, 2009. LNCS 3107, Springer (2004), pp. 49–58

[11] Kwon, W.H. and Choi, S.G.: Real-time distributed software-in-the-loop simulation for distributed control systems. Proc. IEEE Int. Symp. on Computer Aided Control System Design (1999), pp. 115-119

[12] Hanselmann, H.: Hardware-in-the-loop simulation testing and its integration into a cacsd toolset, Proc. IEEE Int. Symp. on Computer Aided Control System Design (1996), pp. 152-156

[13] France, R. and Rumpe, B.: Model-driven development of complex software: A research roadmap. In FOSE ’07: 2007 Future of Software Engineering, Washington, DC, USA, IEEE Computer Society (2007), pp. 37-54

[14] Forsberg, K., Mooz, H., and Cotterman, H., eds.: Visualizing Project Management: Models and Frameworks for Mastering Complex Systems. 3rd edition. Wiley (2005)

[15] Muller, G.: CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity. PhD thesis, Technische Universiteit Delft (2006)

[16] Chung, L.: Representation and utilization of non-functional requirements for information system design. In Anderson, R., Bubenko, J.J.A., and Solvberg, A., eds.: Proc. 3rd Int. Conf. Advanced Information Systems Engineering, CAiSE, Springer (1991), pp. 5-30

[17] Jackson, M.: Problem frames: analyzing and structuring software development problems. Addison-Wesley, Boston (2001)
[18] Jackson, M.: Software requirements & specifications: a lexicon of practice, principles and prejudices. ACM Press/Addison-Wesley, New York (1995)
[19] Reisig, W.: Petri nets: an introduction. Springer, New York (1985)
[20] Baeten, J.C.M.: A brief history of process algebra. Theor. Comput. Sci. **335**(2-3) (2005), pp. 131-146
[21] Higham, D.J. and Higham, N.J.: MATLAB Guide. Society for Industrial and Applied Mathematics, Philadelphia (2000)
[22] Evans, A., France, R.B., Lano, K., Rumpe, B.: The UML as a formal modeling notation. In Bézivin, J., Muller, P.A., eds.: The Unified Modeling Language, UML'98, First Int. Workshop, Mulhouse, France, June 3-4, 1998, Selected Papers. LNCS 1618, Springer (1999), pp. 336-348
[23] Encarnação, J.L., ed.: Computer Aided Design: Modelling, Systems Engineering, CAD-Systems - CREST Advanced Course, Darmstadt, Germany, September 8-19, 1980. LNCS 89, Springer (1980)
[24] Laouafa, F. and Royis, P.: An iterative algorithm for finite element analysis. J. Comput. Appl. Math. **164-165**(1) (2004), pp. 469-491
[25] Keller, K.: An Electronic Circuit CAD Framework. PhD thesis, EECS Department, University of California, Berkeley (1984)
[26] Kelessoglou, T. and Pederson, D.: A framework for analog circuit verification. IEEE Int. Symp. on Circuits and Systems, 1990. (1-3 May 1990), pp. 33-36, vol.1
[27] Cohen, B.: Real Chip Design and Verification Using Verilog and VHDL. VhdlCohen Publishing (2001)
[28] Bergeron, J.: Writing Testbenches: Functional Verification of HDL Models. $2^{nd}$ edn. Kluwer (2003)
[29] Spivey, J.M.: Understanding Z: a specification language and its formal semantics. Cambridge University Press, New York (1988)
[30] Holzmann, G.J.: Design and validation of computer protocols. Prentice-Hall, Upper Saddle River (1991)
[31] Stroustrup, B.: The C++ Programming Language. Addison-Wesley, Boston (2000)
[32] Bi, J., Wu, J.: A formal approach to conformance testing of distributed routing protocols. In Wu, J., Chanson, S.T., Gao, Q., eds.: Formal Methods for Protocol Engineering and Distributed Systems, Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China. IFIP Conf. Proc. 156, Kluwer (1999), pp. 151-163
[33] Tretmans, G.J.: A Formal Approach to Conformance Testing. PhD thesis, University of Twente, Enschede (1992)
[34] van Beek, D.A., Reniers, M.A., Schiffelers, R.R.H., and Rooda, J.E.: Foundations of a compositional interchange format for hybrid systems. In Bemporad, A., Bicchi, A., and Buttazzo, G.C., eds.: Hybrid Systems: Computation and Control, 10th Int. Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, LNCS 4416, Springer (2007) pp. 587-600
[35] Groote, J., Mathijssen, A., Reniers, M., Usenko, Y., and van Weerdenburg, M.: The formal specification language mCRL2. In Brinksma, E., Harel, D., Mader, A., Stevens, P., and Wieringa, R., eds.: Methods for Modelling Software Systems (MMOSS). Dagstuhl Seminar Proc. 6351, IBFI, Schloss Dagstuhl, Germany (2007)
[36] Reussner, R., Mayer, J., Stafford, J.A., Overhage, S., Becker, S., and Schroeder, P.J., eds.: Quality of Software Architectures and Software Quality, First Int. Conf. on the Quality of Software Architectures, QoSA 2005 and Second Int. Workshop on Software Quality, SOQUA 2005, Erfurt, Germany, September 20-22, 2005. LNCS 3712, Springer (2005)
[37] Ramadge, P.J. and Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1) (1987), pp. 206-230