



# Integrating viewpoints in the development of mechatronic products



Martin Törngren\*, Ahsan Qamar, Matthias Biehl, Frederic Loiret, Jad El-khoury

Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden

## ARTICLE INFO

### Article history:

Received 3 May 2013

Accepted 30 November 2013

Available online 28 December 2013

### Keywords:

Mechatronics design  
Multiview modeling  
Model-based design  
Viewpoint integration  
Viewpoint interrelations  
Tool integration

## ABSTRACT

The development of mechatronic products involves multiple stakeholders which have different viewpoints and therefore use different concepts, models and tools to deal with their concerns of interest. This paper argues that an increased emphasis needs to be placed on the relations between viewpoints to be able to deal with the evolving scope and requirements on mechatronic products. We study relations between viewpoints at the levels of people, models and tools, and present solutions that are used to formally and explicitly capture such relations. Viewpoint contracts are used to define the vocabulary, assumptions and constraints required for ensuring smooth communication between stakeholders (people level). Dependency models capture relations between product properties belonging to different viewpoints, and how such dependencies relate to predictions and decisions (model level). Tool integration models describe the relations between tools in terms of traceability, data exchange, invocation and notifications (tool level). A major contribution of this paper is a unification approach, elaborating how these solutions can be used synergetically to integrate viewpoints. An industrial robot case study is utilized to illustrate the challenges and solutions with respect to relations between viewpoints, including the unification approach.

© 2013 Elsevier Ltd. All rights reserved.

## 1. Introduction

The development of mechatronic products involves multiple stakeholders which have different but nevertheless related viewpoints, [1], and therefore use different concepts, product data and tools to deal with their concerns of interest. Viewpoints will also relate to the frameworks and theories of tradition available to the respective stakeholders. In this paper we will be referring to product data in terms of well-formed models but most of the discussion is applicable to other representations of product data.

Examples of typical viewpoints include mechanics design, controller design, software design and safety analysis. According to [1], establishing a viewpoint means defining guidelines and conventions such as recommended types of languages, design rules, modeling methods and analysis techniques. This for example implies that choices of modeling languages should be driven by the context of the design task at hand, including the stakeholder concerns. Also according to [1], the term “view” corresponds to a model, or groupings of models, created using these languages, e.g. a CAD model, where a corresponding viewpoint establishes the conventions for the models.

Mechatronic systems are characterized by a tight integration of multiple technologies. This implies that the different stakeholders, their processes, models and tools will also be tightly affiliated

through the relationships between product parts and properties. It is clearly desirable to facilitate communication between stakeholders, reasoning about the impact of changes, and to ensure overall efficient concurrent engineering.

The overall thesis of this paper is as follows: *Efficient development of Mechatronic products necessitates support for dealing with the intricate relations between viewpoints, at the levels of people, models and tools.*

We hereafter refer to relations between viewpoints as *interrelations* as opposed to relations within viewpoints. In this paper we take a Model-Based Systems Engineering [2] approach to define viewpoint interrelations, and provide models that make interrelations at the three levels explicit; we refer to these as *support models*, which are provided by the solutions presented in this paper:

- *Viewpoint contracts* are used to define the vocabulary, assumptions and constraints required for ensuring smooth communication between stakeholders.
- *Dependency models* capture relations between *engineering models*, such as CAD and Simulink models, belonging to different viewpoints, and can be used to investigate how such dependencies relate to predictions and decisions.
- *Tool integration models* describe the interrelations between tools in terms of their services and data, making it possible to express tool interrelations such as data exchange, traceability, invocation and notifications.

\* Corresponding author. Tel.: +46 8 790 6307.

E-mail address: [martint@kth.se](mailto:martint@kth.se) (M. Törngren).

A major contribution of this paper is a unification approach, elaborating how these solutions complement each-other and can be used synergetically to integrate viewpoints. This is the first time that these three solutions are presented together. A related new contribution is the industrial robot case study, which all three solutions are applied to and which is used to illustrate the unification approach. For the individual solutions, this paper presents viewpoint contracts as a new technique (evolved from so called design contracts, [3,4]). The dependency and tool integration modeling solutions have previously been presented separately, see [5,6]. As a further contribution, we provide an overview of state of the art techniques that in some way address interrelations (such as co-simulation, model transformations, and suggestions for integration specific views), and relate them to our approach.

Dealing with interrelations has clearly always been relevant for mechatronic products, but is becoming even more relevant given the evolving scope and requirements of mechatronic products. One example of this evolution is given by automotive systems which are provided with capabilities such as fleet management, active (and autonomous) braking and steering for improved safety, and platooning (vehicle convoying, for increased energy and transport efficiency). The example of evolving vehicles illustrates common trends for mechatronic products, i.e. increasing connectivity, provision of new functionalities and high quality expectations, all of which imply not only more viewpoints but also an increased reliance on proper management of their interrelations. This evolution leads to an increasing complexity of mechatronic products.

With complexity, we refer to systems which exhibit heterogeneity in facets and relations, see e.g. the survey by Sussman on definitions of complexity, [7], thus posing a problem for humans in dealing with such systems. We will also use a more refined terminology to differentiate between those aspects of a system that are regarded as complex but can be simplified (accidental complexity) and those that are complex in their essence and cannot be simplified (essential complexity). See [8] for a discussion of essential and accidental complexity.

In an extensive literature review of mechatronics research by [9], it is found that the most commonly reported sets of challenges are primarily related to the way a product concept can be described, and how information linked to the product concept can be shared across engineering disciplines. Moreover, many problems in systems and safety engineering are known to stem from misalignment of assumptions and concepts within the organization, typically among different stakeholders, see e.g. [10]. Without proper management of interrelations, it will be difficult to develop systems on time, and fulfill the desired criteria such as cost and performance. The need for integration among viewpoints is also emphasized by for example [11–14].

The overall scope of this paper is illustrated in Fig. 1. Any particular development setting will involve multiple people (teams), models and tools – assigned to perform and support the tasks of the product development processes. The development responsibilities will be partitioned and handled by specific stakeholders, for example in terms of mechanical design, control design, and software design. We use Fig. 1 to illustrate interrelations between a pair of viewpoints at the three levels introduced previously. In order to develop a common understanding of the overall design problem, there is a strong need for communication between stakeholders. For example mechanical and control engineers need to be able to clearly exchange ideas and make sure their decisions are compatible. This communication can be sought informally, such as through meetings. However only relying on such informal means is error-prone. A possible solution to avoid error-prone communication is to turn towards more formal communication means, such as through *models*, [15]. While communication based

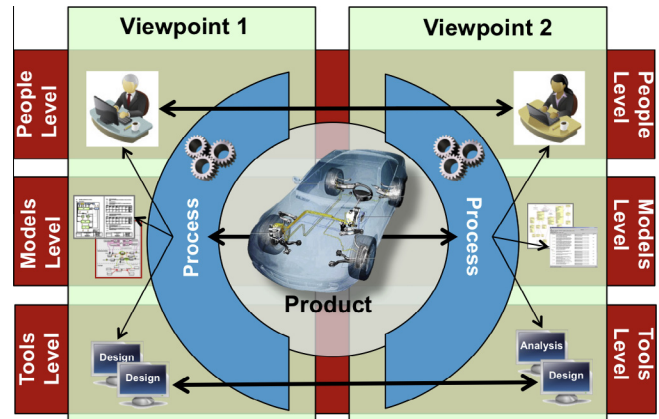


Fig. 1. People, models and tools involved in design of a robot.

on models can bring benefits if performed appropriately, see e.g. [16], it comes with challenges. First of all, models developed through a diverse tool set carry different meaning and their full understanding by other stakeholders cannot be taken for granted. Moreover, it is common that there are dependencies between multiple models which are only implicitly known, and there is a need to understand how changes across the models propagate. Hence communication through models also needs to address the management of such dependencies. Finally, to automate such propagation, there is a need to make the corresponding tools interoperable. A systematic approach for dealing with these challenges (that is, the proper handling of interrelations at the people, models and tools level) is the focus of this paper.

This paper is structured as follows: In Section 2, we introduce a robotics example which is used throughout the paper to illustrate challenges as well as solutions. In Sections 3 and 4, we address interrelations for each of the three levels; people, models and tools, where Section 3 discusses and exemplifies each of the three levels and the challenge of dealing with them, whereas Section 4 provides solutions that addresses them. Section 5 then brings the three solutions together and discusses how they can support each-other. In order to provide an overview of the state of the art and how it relates to our approach, we divide related work into the following coarse grained areas – described in the following subsections: integration specific views (6.1), processes and multi-view frameworks (6.2), model-driven engineering (6.3), co-simulation and optimization (6.4), tool integration (6.5), and multiview consistency (6.6). We discuss implications of our solutions with respect to industrial adoption and the generality of the solutions in Section 7. Finally, we provide conclusions with an outlook towards future work in Section 8.

## 2. A robotics example

The purpose of this section is to present a mechatronic design example. The problem at hand is to pick and place an object in a two dimensional environment with known obstacle locations. The design problem is formulated as follows:

*Design a pick-and-place revolute robot under the following constraints.*

1. The robot should cover a workspace (WS) of 4 m<sup>2</sup>.
2. The End to End Response Time (EERT) of the robot should not be more than 0.5 s.
3. The Closed loop Position Accuracy (CPA) should be at least 5 mm.

The above requirements are taken into account specifically for the chosen example. There will almost certainly be additional requirements such as on *Load* and *Cost*, however (for simplicity) they are not considered in this paper.

As per [17], the robot design process begins by synthesizing the *basic structures*, one of which is chosen to synthesize *quantified structures*. For this problem, we will not focus on the initial design synthesis phase (to search for solutions) or on assessment of alternative solutions, which are also significant challenges in mechatronic design [9]. We assume that a design solution has been found through product synthesis – consisting of a two Degrees Of Freedom (DOF) revolute robot. In Fig. 2, the quantified structure of the selected solution is shown, along with the workspace, the pick and place point, and the obstacle location.

For this example, three viewpoints are considered, each viewpoint addresses the needs and preferences of a development stakeholder, which are explained as follows:

1. **Mechanical designer:** Designs the mechanics of a two arm robot with a workspace of  $4\text{m}^2$ . Further design variables to be considered are: link length ( $L_A, L_B$ ); link width ( $W_A, W_B$ ); material density ( $\rho$ ); range of joint motion ( $\theta_A, \theta_B$ ); point of origin in the workspace (O); torque of motor ( $M_A, M_B$ ); resolution of sensor ( $S_A, S_B$ ); inertia of arm ( $I_A, I_B$ ) and maximum distance (PE) between pick and place point. It is assumed that the robot can grip objects of any size; hence no gripping movement is considered in this example.
2. **Control designer:** Designs a control system with a closed loop position accuracy of 5 mm, and EERT of maximum 0.5 s. Further design characteristics considered are: plant model ( $G(s)$ ), choice of controller approach and structured (abbreviated as CS, such as feedforward/feedback, adaptation), controller modes (*Modes*), state estimation (based on which sensors are used), controller gains (CG), admissible range of sampling frequencies ( $[f_s^{\min} - f_s^{\max}]$ ), and time delay, ( $\Delta T$ ). The Controller Performance Metrics (CPM) is defined based on the closed loop system Rise Time ( $T_{\text{Rise}}$ ), Settling Time ( $T_s$ ), *bandwidth* and the Steady State Error ( $e_{ss}$ ).

3. **Hardware/Software (Hw/Sw) designer:** Designs and implements a centralized or a distributed controller for the robot, satisfying the requirements for the I/O interface and processing power. The designer also makes a decision on whether to use the existing Electronic Control Unit (ECU) or follows up the development based on a new ECU. Further Hw/Sw characteristics considered are: Processor type (CPU), Clock frequency ( $f_{\text{clock}}$ ), Resolution for A/D converter ( $\text{Res}_{A/D}$ ), Resolution for D/A converter ( $\text{Res}_{D/A}$ ), Error Tolerance for A/D converter ( $\Delta_{A/D}$ ), Error Tolerance for D/A converter ( $\Delta_{D/A}$ ), Controller Code ( $\text{Code}_C$ ), I/O code  $\text{Code}_{I/O}$ , Worst Case Execution Time for the controller ( $\text{WCET}_C$ ), interference due to resource sharing ( $I$ ), Response time ( $T_{\text{Response}}$ ) and processor utilization ( $U$ ). It is assumed that the controller code shares CPU resources with other tasks including: (1) Diagnostics; (2) Emergency shutdown; (3) Initialization. The execution time and interference from higher priority tasks is used to determine the response time for a particular task using the equation:  $T_{\text{Response}} = \text{WCET} + I$  (where blocking ( $B$ ) is neglected).  $\text{WCET} = \text{WCET}_i$  refers to execution time (for four tasks including the controller task), and  $I$  is the interference. Requirements on CPU utilization  $U$  could also be present to provide room for future extensions, however for simplicity, they are left out in this example.

In order to proceed with the robot design, the mechanical designer has to ensure that the design requirements are met by the mechanical structure. The control designer has to come up with a controller which should satisfy the control design objectives. The HW/SW designer has to ensure that the embedded platform delivers the required performance especially concerning  $\text{WCET}_C$  and  $T_{\text{Response}}$ , and that the measurement resolution for all the sensors is satisfactory to ensure CPA. In short, the embedded implementation of the controller needs to satisfy both the control design and HW/SW design objectives.

Each of the three stakeholders – based on the design specifications for each viewpoint – develops models focussing on different aspects of the robot. In doing so, they utilize different design and analysis tools, such as a CAD tool (e.g., Solid Edge [18]) for

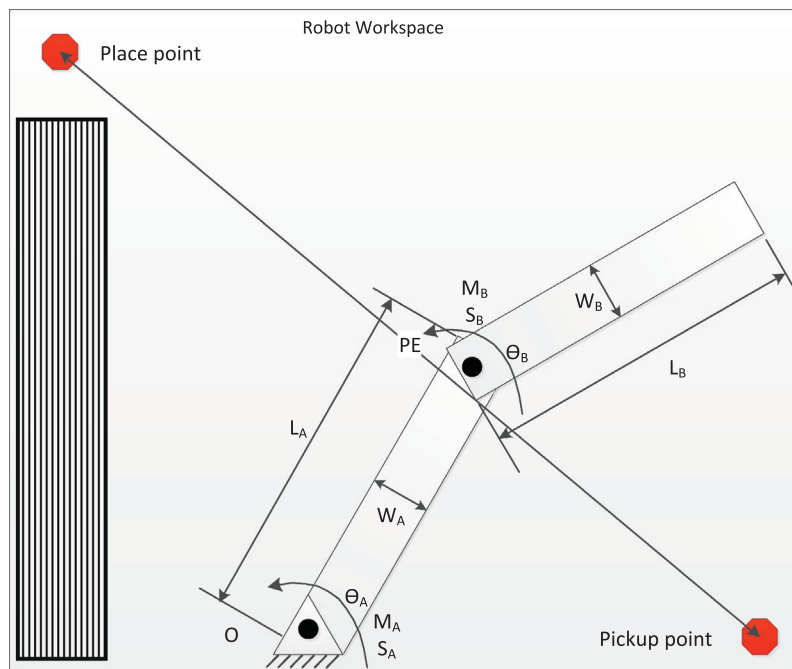


Fig. 2. Design concept for a robot.

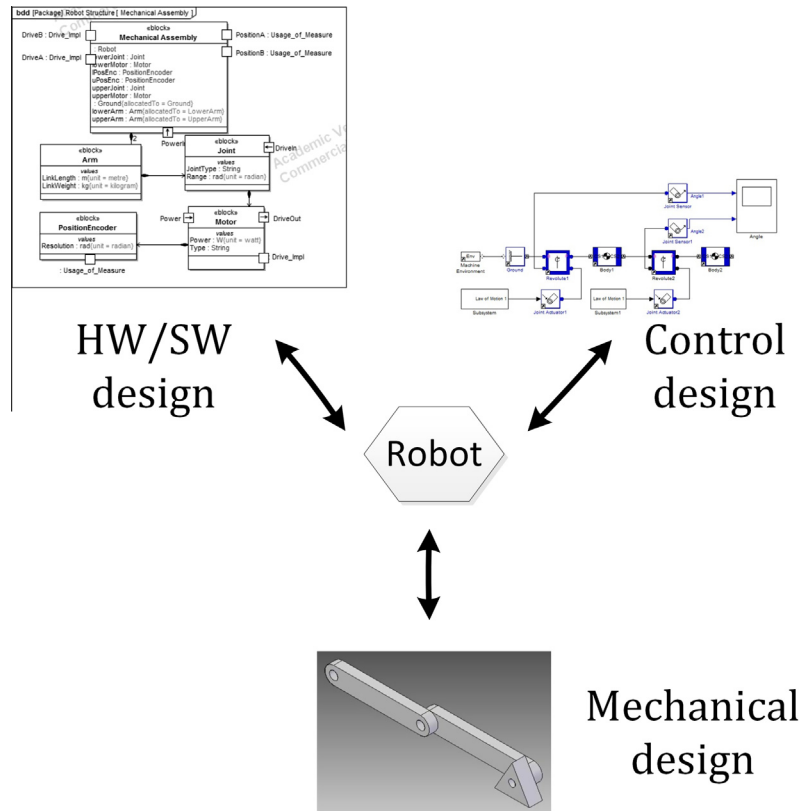


Fig. 3. Disparate models constituting different views on the robot, with several dependencies in between.

mechanical design, a control design tool (e.g., Matlab/Simulink [19]) and a software design tool (e.g., IBM Rational Rhapsody [20], a UML tool).<sup>1</sup>

Addressing the high level requirements such as for EERT and CPA involves the combined work of the three stakeholders. To meet the design requirements and perform a trade off analysis involves formulating a design study coupling the mechanical, control and Hw/Sw design models (see Fig. 3). As discussed earlier in Section 1, such a setting will require addressing the interrelations at people, models and tools levels, demanding necessary integrations within each of these levels. For example, in mechanical design, the choice of link length effects the inertia, resulting in changes in transfer function of the plant used in control design. This demands mechanical and control designer to interact and find a consensus by resolving the corresponding interrelations (solution at people level). On the other hand, the dependencies between link length, inertia and plant transfer function need to be formally captured for adequate management (solution at model level). In addition, tools supporting the mechanical and control views need to be integrated to support information exchange for resolving this interrelation (solution at tool level).

Efficient and effective robot design cannot be performed without managing such interrelations, since each stakeholder, apart from working on their own, has to synchronize with others. A design satisfying the goals of the mechanical designer may have a bad controllability, and a control system – having perfect closed loop performance – may be computationally too expensive for the embedded computer. In general, there will be interrelations between all the involved viewpoints, however for the sake of

simplicity, we disregard the interrelations between mechanical design and Hw/Sw design. In the next section, the challenges that one has to face for integrating viewpoints are presented.

### 3. Levels of viewpoint integration – the challenges

The following sections will elaborate the challenges at the people, models and tools level, with their respective solutions and their synergies discussed in Sections 4 and 5 respectively.

#### 3.1. People integration

Effective communication between stakeholders is necessary in order to manage the product design process and to have a common understanding of the design problem. In terms of engineering a product, the stakeholders have different interests and they are typically experts in one engineering domain [21], which means that although being subject matter experts, they are usually not well versed in terminologies and technologies used in other domains. In such a setting, terms like *performance* and *system* can be interpreted differently. A control engineer may associate performance with the closed loop system bandwidth or settling time, whereas the software engineer might instead consider processor load or response time to events.

For the robot example presented in Section 2, the considered design scenario requires mechanical and control designer to work together for designing the new control system based on the underlying mechanical system. On the other hand, control and software designers have to work together too, so that the controller performance metrics are met by the developed embedded computer. In these two scenarios, each stakeholder needs to understand the requirements and demands of each other but they also need to be aware of affects of their work on others. For example,

<sup>1</sup> For the example we make the assumption that the software design tool provides plugins and features required by the HW/SW designers (e.g., code generators, compilers, WCET analysis tools, end-to-end deadline analysis tools, etc.).

mechanical designer may put up a proposal for robot's mechanical design which carries bad controllability – a feedback provided by the control designer once the mechanical design characteristics are used in the plant model. Section 4.2 presents a possible solution to this challenge.

### 3.2. Model integration

A product is designed with the purpose of possessing certain properties, described as requirements in the design specification [22]. To gain information about product properties, product designers create models e.g. mathematical, analytical and functional models. Design is a propagation from model to model, and the design models help in answering different types of questions about the product, e.g. how is the performance, what is the cost etc.

The partitioning of the overall problem into domain-specific ones in mechatronic design leads to a situation where product properties are spread among a set of disparate models, and it is common that these properties influence each other, referred to as *dependencies* [23]. Such dependencies can also be thought of as dependencies between components (having certain properties), systems or sub-systems, or at a higher level – between models or files (containing models). Traditionally, these dependencies are usually only implicitly known, meaning that engineers possess knowledge about dependencies, which are never captured formally or explicitly in a model. In addition, in the current MBSE practice, modeling dependencies explicitly is not considered essential, hence support available in current modeling and simulation languages is inadequate for dependency management [23]. Therefore, it is difficult to deduce the impact of a change introduced in one model on the other models in the set, resulting in inconsistencies. Hence, seeking a solution to manage dependencies between disparate models is vital to avoid making decisions based on inconsistent data [24]. It has to be noted that dependencies are also required to be managed when performing a design optimization, such as optimizing the controller and the mechanical system concurrently.

The dependency management challenge can be illustrated through the robot example. For instance Fig. 3 illustrates three different modeling views that are established for the robot and among which several dependencies exist. For example, the dependency between the plant model and the dynamic properties of the robot involves two viewpoints: mechanical design and control design. In addition, there are dependencies within viewpoints, such as between (the properties) workspace and the length of each link in the mechanical design viewpoint. The dynamic properties of the robot are dependant on the chosen length and width of the link, along with the selected material. In the control design viewpoint, the selection of sampling frequency is dependant upon the controller performance metrics. In the Hw/Sw design viewpoint, the property  $WCET_c$  influences the sampling frequency selected earlier in the control design viewpoint. Mismanagement of such dependencies may lead to inconsistent property values, design delays, or wrong decision taken.

Understanding the effect of dependencies requires knowledge of dependencies not only between disparate models, but also dependencies within models. For example, to find the effects of changing the length of the link (of the robot) requires knowledge of dependencies within the CAD model that relate to the property *linklength*, such as the dependency between mass and link length.

Dependency management is a complex problem due to multiple tools utilized during the modeling process and different standards of information exchange used by such tools. For example STEP [25] standard may be utilized for CAD tools, but cannot be directly applied for control design tools. On the other hand, HW/SW design

tools may utilize XML/XMI as a data exchange format. Therefore, to manage the dependencies present between models created through such tools, information bridges have to be developed. Managing dependencies is thus tightly connected to the tool integration problem (further detailed in Section 3.3).

### 3.3. Tool integration

The tool landscape for developing mechatronic products is both large and diverse, consisting of a number of specialized tools that support specific aspects of development. Each tool focuses on supporting particular development tasks or aspects of the development process. In the running example (see Section 2), Solid Edge is used for mechanical engineering, Matlab/Simulink is mainly used for control engineering, whereas the UML tool is used for embedded software engineering. The associated development tasks – here the tasks for mechanical design, control design and software design – are not performed in isolation, but there are relations between them (see Section 3.1). Similarly, there is a relation between the product models (see Section 3.2). For a corresponding, comprehensive tool support throughout the development process, an engineering environment is needed that allows for the seamless use of several development tools, i.e. for suitable integration of the tools. Such an engineering environment needs to support sharing and reusing artifacts created with different tools, creating traces between these artifacts, invoking services of several tools and notifying developers of relevant activities that occurred. If such an engineering environment is automated and realized by software, we call it a *tool chain*. The data and services of the tools are however not readily accessible due to the technical, syntactic and semantic differences among the tools. While the technical and syntactic differences among the tools can be regarded as contributors to the accidental complexity of tool integration, the semantic differences contribute to the essential complexity, [8].

The development of tool chains, and thereby tool integration, deals with a number of challenges. In general, a tool chain supports a development process by automating the integration-related tasks of that process. Traditional tool chains support simple connections between a small number of tools, such as the connections between editor, compiler and linker. Tool chains with such a simple architecture can thus be realized by a pipe-and-filter architecture [26]. However, modern tool chains need to support development processes that are multi-disciplinary, model-based, iterative [27] and include a larger number of tools used in different phases of the system lifecycle. To get a tool chain accepted by its planned users, it is important that the users are involved [28] and the tool chain is customized to their individual requirements. Each development project may use a specific selection of development tools, and a specific development process. Due to the large number of development tools and even larger number of possible tool combinations, one-size-fits-all tool chains cannot provide adequate support. Instead, tool chains need to be tailored to the product development process, development team and development tools of each development project. As a consequence, a number of different, tailored tool chains need to be developed.

An established methodology for the development of tool chains is not available. Tool chains are often implemented ad hoc, leading to “fragile integrations” [29] that are difficult to extend and maintain. The lack of a methodology reveals itself in the form of insufficient support for synthesis and early analysis of tool chains – current platforms for tool integration provide partial solutions, but also introduce accidental complexity through the amount of coding, low-level technologies and configurations. Tool chains are manually implemented and can only be tested after the time and effort for implementation has been spent.

Integration approaches typically focus on describing the data of tools in the form of models and corresponding metamodels; the tool chain, however, is not explicitly modeled using concepts from the domain of tool integration. Instead, tool chains are built in a bottom-up manner and are often described on implementation level, using source code and the concepts offered by integration frameworks [30,31]. These concepts are very detailed and depend on the use of certain technology. Alternatively, tool chains are described by general purpose modeling languages and domain-specific languages; an example is the use of business process models for describing tool chains [31]. Borrowing descriptions and languages that were originally defined for other domains/viewpoints for the purpose of tool integration is a “workaround”, which introduces accidental complexity [32].

Several specialized technologies for realizing parts of tool chains are currently available, such as model transformation tools, tracing tools or libraries for exposing services of tools. Each of these integration technologies describes one aspect of the tool chain – a complete tool chain needs to cover several aspects. The functionality of a tool chain is thus fragmented across several integration technologies realizing parts of the tool chain. Each integration technology provides different, partial views of the tool chain, which are described using specific languages or configurations. To obtain an overview of the actually deployed or required functionality of the complete tool chain, several partial descriptions have to be consulted. The relationship between these descriptions is neither explicitly described nor managed, so inconsistencies can occur. This situation contributes to the accidental complexity of tool chain development.

#### 4. Solutions for integrating viewpoints

##### 4.1. Rationale for the proposed model-based approaches

A possible solution to describe and manage viewpoint interrelations is to employ a general purpose modeling language, e.g. OMG SysML™ [33]. Through model and tool integration, a holistic system view can be integrated with other domain or application specific views, as documented in [34,35], to address the viewpoint interrelations. However, often such a solution requires extension of the general purpose modeling language, which adds to the intrinsic complexity of the underlying system and hence increases accidental complexity [36]. It is also debatable whether such languages fulfill the full purpose of a common language between multiple stakeholders, especially in terms of understanding gained [9]. Therefore, in contrast to using a general purpose modeling language, this paper assumes solutions based on Domain-Specific Modeling Languages (DSMLs), each built for a specific and restrictive purpose, as per the demands of the particular viewpoint (in this case, the purpose of the support models) [37]. Our proposal is to employ multiple DSMLs (pertaining to multiple viewpoints) and minimize the accidental complexity by capturing the object of interest explicitly at the appropriate level of abstraction and through the most appropriate formalism [38]. In the following we present solutions to the challenge of viewpoint integration for the levels of people, models or tools. These solutions are discussed one by one in the following sections, whereas the synergy between them is discussed in Section 5.

##### 4.2. Viewpoint contracts

To solve the communication problem between people in a diverse setting, a possible solution is to consider a common vocabulary for the different stakeholders. Such vocabulary may be comprehensive or minimalistic. In either case, the vocabulary may be provided with translations to domain/disciplinary specific concepts.

In line with our proposals on utilizing a DSML instead of a general purpose modeling language, we consider defining a minimalistic vocabulary for specific design scenarios, instead of a comprehensive vocabulary to suit all possible scenarios. This vocabulary should obviously be useful and understandable by the corresponding stakeholders, for instance to solve design tasks between mechanical and control designers. The vocabulary will consist of a minimal set of concepts, which can then be made part of an agreement between the two stakeholders.

With inspiration from design contracts, [3,4], we here introduce the concept of *viewpoint contracts*, referring to contracts between two (or more) viewpoints, defining a *vocabulary* that is used as the basis for communication across two specific viewpoints.

We have identified the following steps in developing contracts:

1. Define which viewpoints, scenarios and other particular system characteristics that the contract applies to.
2. Define a vocabulary, the language among the stakeholders. This may necessitate formulating mappings from the vocabulary concepts to viewpoint specific concepts.
3. Formulate assumptions and constraints on the concepts part of the vocabulary, suited to the scenarios and system characteristics at hand.

These three steps together define the contract. The resulting agreement is said to be valid provided that both parties adhere to their obligations of the agreement.

When defining/choosing contracts, stakeholders have to meet and come to an agreement. Alternatively, persons knowledgeable in both domains will make the decisions. Regardless of this, an important starting point is to clarify the setting for the contract in terms of what questions and decisions the contract(s) is supposed to resolve. This will involve defining scenarios in which the stakeholders are likely to use information dependent on the other stakeholder, such as for example in control synthesis which is dependent on mechanical design parameters (Section 4.1.1 elaborates this for the robot example).

Given a number of scenarios, it may be that a number of contracts have already been developed and are available to choose from, or use with some minor modification. Alternatively, no (suitable) contract may be available, so a new one has to be developed.

We will now illustrate the definition of a viewpoint contract between Control and Embedded systems stakeholders (step 1 above) in the context of control and embedded systems design. Specific considerations in this example encompass controller timing parameters, such as periodicity and feedback delay, and the embedded system architecture, for example in terms of its execution strategy. In this case, we assume system characteristics that resemble Programmable Logic Controllers (PLC). More specifically, we assume PLC style controller execution as illustrated by the following pseudo code:

```
initialize state;
set periodic event h;
while (true) {
    await event h;
    sample inputs;
    write previously computed outputs;
    compute outputs, update state for next period;
}
```

In this execution pattern, a number of functions are invoked; *initialize*, *sample*, *write outputs*, *compute outputs*, *update states*, and specifications for when to execute these functions (periodically with a period *h*). This observation forms the basis for Step 2,

defining concepts that may be relevant for the contract vocabulary. Given the specific concern of controller timing, these functionalities (i.e. initialize, sample etc.) and the period  $h$ , constitute such concepts. Step 3 involves defining assumptions and constraints on these concepts. An example of a constraint is to explicitly define the periodicity of the controller and admissible jitter, i.e. acceptable deviations from nominal sampling instants with respect to performance.

A corresponding contract, here termed LET, can be formulated as follows, [4].

**LET:** A LET contract is specified by a tuple  $(M, h)$  where  $M$  is a state machine and  $h$  is a period measured in some time unit, e.g. seconds. The LET contract states that the inputs are sampled at times  $t_k^s = k \cdot h$ , that the outputs are computed and the machine performs a state update at some point in the interval  $[t_k^s, t_{k+1}^s)$ , and that the outputs are written at  $t_k^a = t_{k+1}^s = t_k^s + h$ . The sampling instant jitter is assumed to be zero and  $\Delta T = h$ .

The LET contract explicitly states that the time delay from sampling to actuation equals the sampling period and moreover that sampling instants will occur with negligible jitter. The contract implies obligations for both the control and embedded systems stakeholders. The latter have the obligation to make sure that the functions are executed according to the timing constraints of the contract. The main obligation of the control engineers is to ensure the correct behavior of the closed-loop system with the assumption that the embedded systems engineers will do their job. Given the contract, the stakeholders can partly proceed in parallel as long as the contract is not violated. In case something needs to be changed that will violate the contract, impact analysis across the viewpoints and a potentially renegotiated contract may be required. The degrees of freedom and level of decoupling can be increased by providing tolerances for the chosen concepts, [3].

While it could be argued that modern technologies such as code generation and co-simulation will to a large extent deal with viewpoint integration for the given example of the LET contract, we argue that viewpoint contracts fill an important complementary role; that of making the vocabulary and assumptions explicit. If, for example, the LET controller is to be reused without the contract, the timing assumptions may be lost. In such case, the expected performance may deteriorate, even if the experienced time delay for the controller is reduced, [3]. The fact that the sampling period jitter is assumed to be negligible is a further example of such an important assumption, with implications for both stakeholders. As shown by [4], simulation blocks can be developed to support the realization of contracts such as LET, thus operationalizing contracts.

#### 4.2.1. Viewpoint contracts for the robot example

In the robot example we have three types of stakeholders/viewpoints. This gives us the possibility to define a contract between each pair of viewpoints, or at least when essential dependencies among them exist. Given that we are dealing with mechatronics it is not surprising that such dependencies will indeed exist. For the purpose of the example, we here choose to illustrate viewpoint contracts only for Control/Hardware-software, and for Mechanics/Control.

Ultimately we would expect that many contracts have already been defined. For the interrelations between Control and Hardware/Software, this is the case, with existing contracts described by [4]. In this case, we assume that the LET contract is selected for the following reasons. The stakeholders have a preference for the LET type of execution scheme. A LET controller provides straightforward timing but on the other hand introduces a one sampling period time delay. A prerequisite for using the LET

contract is thus that this time delay does not violate performance requirements such as the *Settling time*. For the robot example, we assume this to be asserted through simulation and/or analytical computation.

For a situation when no existing contract fits the intended scenarios and characteristics, a new contract has to be created. A particular choice of vocabulary for the contract will yield one of the following two situations:

- *The concepts part of the contract vocabulary directly correspond to viewpoint concepts:* For the example, the period  $h$  can directly be related to viewpoint concepts. Admissible sampling periods as determined by closed loop performance requirements (inversely proportional to the sampling frequency) provide bounds for  $h$ , whereas the  $WCET_C$  provides a lower bound for the period that can be implemented, i.e.  $WCET_C < h < 1/f_s^{\min}$ .
- *The concepts part of the contract vocabulary do not correspond directly to viewpoint concepts, an explicit mapping to one or more viewpoint concepts thus has to be provided:* An example of a more complicated mapping is the relation between the response time of one or more tasks implementing a controller, and the time delay of a controller [39]. This relation is non-linear in the general case; for example for many cases of fixed priority scheduling the delay will be varying and different from the response time. An advantage of the LET contract is that it drastically simplifies this mapping, since for the LET contract,  $\Delta T = h$ , and it is sufficient to check that  $h > T_{\text{Response}}$ . This comes at the price of a longer, but fixed, time delay.

In the following we assume that no Mechanics/Control contract is available, and provide considerations concerning the choice of vocabulary. The mechanical and controller subsystems will be integrated to form a closed loop system, for example during simulation. This implies that their interfaces are a strong candidate to be made part of the contract's vocabulary. For example, the set points for the motor torque (for the two motors of the robot  $M_A$  and  $M_B$ ) can be considered to be part of the contract's vocabulary since the set points are provided from the control algorithm (control viewpoint) to the motors (mechanical viewpoint). Another possible scenario is to develop a contract that supports stakeholder interactions during the design of the control algorithm, where typically a plant model is used for controller synthesis or parameter tuning. For this scenario, possible (alternative) concepts to make part of the vocabulary include a transfer function of the robot ( $G(s)$ ), aggregated parameters such as inertia of links ( $I_A$  and  $I_B$ ), or basic mechanical parameters such as geometrical properties ( $L_A$ ). When choosing 'transfer function' as part of the vocabulary, a direct mapping to the concepts used by control engineers is established, but the mapping to the mechanical engineering concepts has to be provided.

#### 4.3. Dependency modeling

To manage consistency, an engineering change or a design process workflow over a set of disparate models, modeling and management of dependencies is central. The question is how to manage the dependencies and what type of dependencies can be encountered during a mechatronic design process. In simple words, a *dependency* can be considered as a relationship or an operation performed on a set of independent entities to find a dependant entity. In this sense, an equation to calculate the volume of a physical object, the heuristics used to select controller gains, a model used to predict the Worst Case Execution Time (WCET), all can be considered to express dependencies. A dependency can also be a function of other dependencies, such as an equation being a function of other equations. Section 3.2 concluded that implicit or ad hoc

solutions to dependency management are not effective, and support to formally capture dependencies is currently lacking. We here present an approach to formally capture and manage dependencies through a *Dependency Model* supported by a Domain Specific Language (DSL) called the *Dependency Modeling Language* (DML) and a supporting tool *Dependency Modeler* [5]. The aim is to support dependence modeling – within and in between disparate models – at the right level of abstraction and at different levels-of-detail.

As stated by [40], a design process involves two main steps: (1) find all possible design options, (2) find the best possible design option. The process of finding the best possible design option includes making predictions about the outcomes of a design option. In order to make a prediction, designers build design models, and express their beliefs in them, along with making a selection on different design variables that define a design option. Due to this distinction between prediction and selection, it is necessary that the DML supports keeping a distinction between predicted and selected properties. The DML does this by considering two types of dependencies; Synthesis Dependency (SD) which refers to a heuristics used to guide the selection of a property called a Synthesis Property (SP); and Analysis Dependency (AD) which refers to an equation or an analysis model used to predict (rather than select) a property called an Analysis Property (AP). Note that it is possible for an SD to be a parametric function of other SDs. Through this characterization, the human interaction in the design process is captured, where an SD reflects on a decision made by a designer, and an AD reflects the equation or the model used by the designer to make a prediction.

To manage an engineering change order, it is important to know which properties will be influenced by introducing the change, and to which model these properties belong. This necessitates modeling dependencies at the property level – i.e. the right level-of-abstraction for the dependency model is where the properties influenced by a particular dependency are modeled. However, when considering a dependency model, the stakeholders – based upon their background – may not be able to fully understand a model showing only dependencies between certain properties. The semantic context around a property is necessary to build a proper understanding, which is supported in the DML through *part-of* and *value-of* relationships. An example is shown in Fig. 4 where semantic context around a property *Linklength* is shown.

As the design process evolves, the knowledge about the dependencies increases, and some of the dependencies – not known initially – may be known later. Therefore, it is essential that a modeler is allowed to capture dependencies at different levels-of-detail, for example from a detail where only the existence of a dependency is modeled to a detail where the dependency validity conditions (e.g. parametric constraints) are also captured.

In the context of MBSE, model transformations that can support automating the process of dependency modeling are also considered for the *Dependency Modeler*. In [5], such transformations

are aimed to support the so-called *dependency patterns*, which are patterns that gather and illustrate known dependencies, such as equation-based and structural dependencies. For example, the equation  $Mass = Density * Volume$  captures known relationship between three properties, and is considered a dependency pattern, whose execution can be supported through a model transformation. Note that it is not necessary for properties to be present under the same model, rather properties are usually spread across disparate models, and the role of the pattern is to gather these properties and model dependency relationships between them. Another important point is that dependency patterns are not always mathematical equations, and there can be other types of patterns, for example, the structural relationships between SysML and Modelica models, where the mapping between the two meta-models is considered a dependency pattern. Overall, the role of such patterns is to help automate the process of modeling dependencies.

#### 4.3.1. Dependency modeling for the robot example

For the robot example presented earlier in Section 2, three viewpoints namely mechanical design, control design, and HW/SW design were considered. In the following, we will utilize the dependency modeling approach to capture the dependencies involving these viewpoints.

Fig. 5 shows a snapshot of the dependency model illustrating that three modeling views (abstractions) are constructed for the robot. Each view captures a set of robot's properties between which the dependencies are present. The dependencies can be seen by opening up a view within the dependency model, e.g. *RobotMechanicalDesign*. The robot *Contains* sub-components consisting of two arms, a control system, and an ECU (see Fig. 5). The figure is essentially a top-level view of the dependency model, where each of the containers can be expanded to view the detailed dependency model. Such a detailed view is illustrated in Fig. 6, where a graph visualization of the dependency model is presented, and the properties and dependencies corresponding to the involved views are shown. For example, the Controller Structure (CS), the Controller Gains (CG) and the sampling period ( $h$ ) selected in the control design viewpoint are used to synthesize the Controller Code ( $Code_c$ ) in the HW/SW viewpoint (see  $SD_{12}$ ). An important thing to note is that the requirements *CPA* and *EERT* (Synthesis Properties) are first used to construct a robot design solution, and later – for the chosen solution – these two properties are predicted, meaning that the properties of the current solution are compared against the specifications (requirements satisfaction).

#### 4.4. Tool integration modeling

The challenges of tool integration identified in Section 3.3 can be addressed by providing systematic support for describing and developing tool chains. The cornerstone for improved description and development of tool chains is a (DSML) language, which provides the means for expressing tool chains. Supplementary methods, a process and tools are proposed that allow for the efficient development of tool chains. The proposed approach enables the efficient development of tailored tool chains, which have the potential to improve the productivity of development. The aim is to increase the productivity of the tool chain architect, by finding an appropriate level of abstraction, at which tool chains can be specified. Functional analysis techniques allow early verification of the appropriate alignment between tool chain and process and they enable correctness checks. Non-functional analysis can examine if the tool chain design is within the required cost limits and fulfills the necessary safety qualification level.

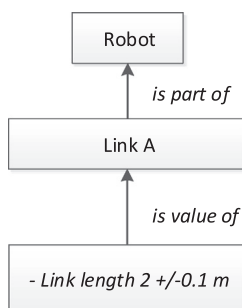


Fig. 4. Building semantic context around a property.

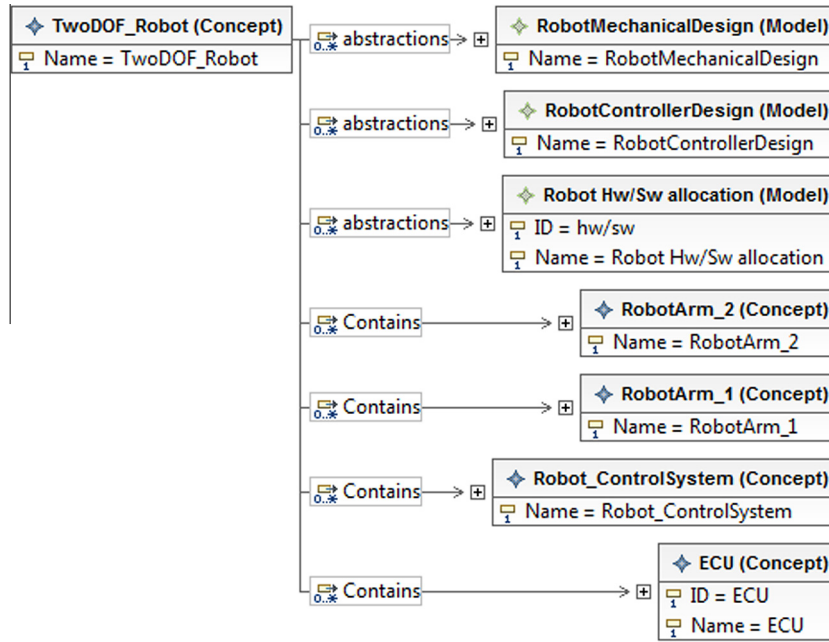


Fig. 5. Snapshot of the robot dependency model constructed in DML showing the involved modeling views and components (concepts) [90].

#### 4.4.1. Language for describing tool chains

The Tool Integration Language (TIL) is a domain specific modeling language (DSML) for describing tool chains systematically, graphically and with well-defined semantics. The language concepts are from the domain of tool integration and express the essential design decisions on an architectural level of abstraction. Such a description can support several phases of the development of tool chains. In the following, TIL is briefly introduced in terms of abstract syntax, concrete syntax and semantics. A more detailed description can be found in [6], compatible formal semantics of the behavior of TIL is described in [41]. The graphical concrete syntax of each language concept is introduced by a simple example in Fig. 7; the concrete mapping function – which maps the abstract to the concrete syntax – is defined by corresponding circled numbers ①...⑦ in Fig. 7 and also explained in the following text.

A **ToolChain** ① provides a container for all elements of a tool chain. ToolChains can be composed hierarchically.

A **ToolAdapter** ② exposes both functionality and data of an integrated development tool. A ToolAdapter makes two kinds of adaptation: (1) It adapts between the technical space of the tool and the technical space of integration for both data and functionality. (2) It adapts the structure of data and the signature of services available in the development tool to the data structure and service signature defined by the ToolAdapterMetamodel. A *ToolAdapter-Metamodel* specifies the structure of the data and the signature of the functionality exposed by the ToolAdapter. The ToolAdapter-Metamodel can be specified by the architect of the tool chain, it does not necessarily need to be equivalent to the tool data, but it provides a simplified view of the tool data and functionality. The ToolAdapter is responsible to parse the data of its tool and to make it available as a model, according to the ToolAdapterMetamodel. In addition, each object in the model is made available as a resource.

A **ControlChannel** ③ describes concurrent control flow. ControlChannels specify the *source* and *target* component, the *source\_service* in the source component that triggers the ControlChannel and the *target\_service* in the target component that is called by the ControlChannel.

A **Sequencer** ④ allows for sequential control flow. It executes a sequence of services in a defined order. Any of the incoming ControlChannels triggers the execution of the Sequencer, which

then executes all outgoing ControlChannels in the predefined order with blocking-call semantics.

A **User** ④ is a representative for a real tool chain user. It is used to describe the possible interactions of the real users with the tool chain. Outgoing ControlChannels from the user denote tool chain services that are invoked by the real tool chain user. The user can trigger these calls via the web-interface of the tool chain. Incoming ControlChannels to a User denote a notification sent to the user, e.g. by e-mail.

A **DataChannel** ⑤ transfers data between

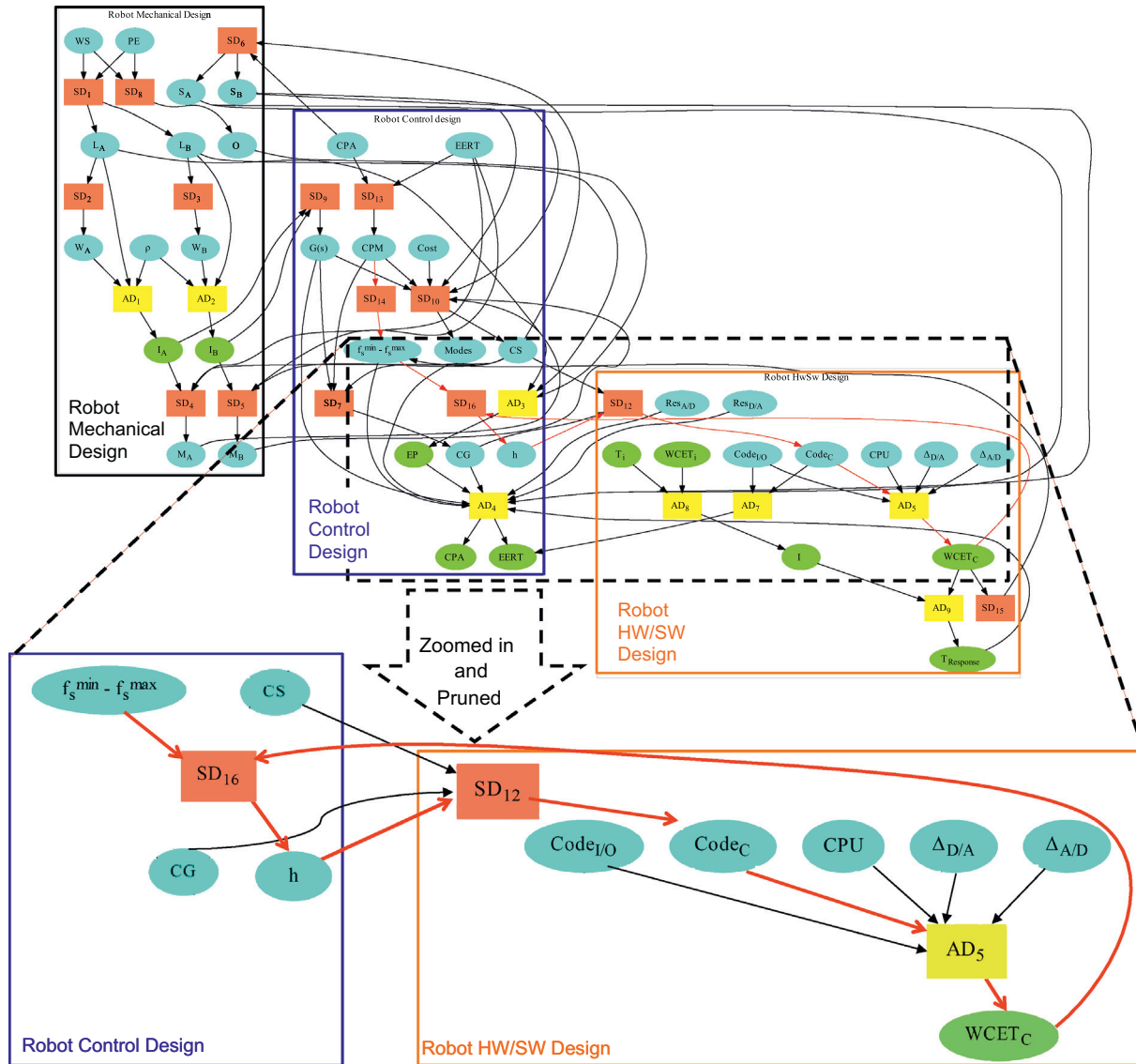
ToolAdapters. A model transformation can be attached to a DataChannel to resolve structural heterogeneities between the data of the ToolAdapters. It is executed at runtime of the tool chain. DataChannels specify the *source* and *target* ToolAdapter, the *source\_service* in the source ToolAdapter for extracting tool data and the *target\_service* in the target ToolAdapter for injecting the transferred tool data. To execute a DataChannel at runtime, it needs to be activated by a ControlChannel.

A **TraceChannel** ⑥ describes the possibility to establish trace links between tool data at runtime. At design time one can specify the type of data that can be linked by traces. The TraceChannel connects a source ToolAdapter and a target ToolAdapter.

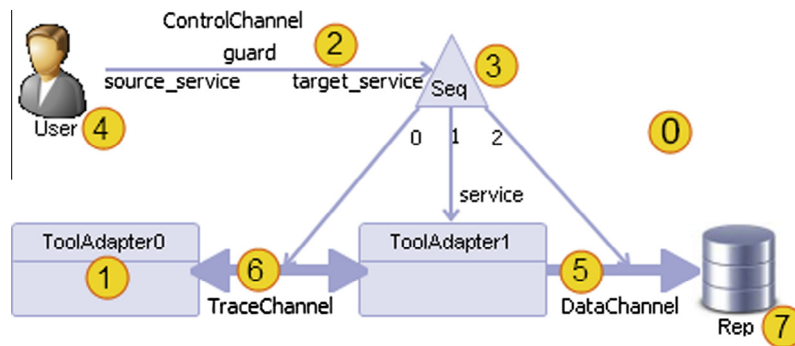
A **Repository** ⑦ is a tool adapter that provides storage and version management, e.g. a tool adapter for SVN. Data can be read and written from a repository by a data channel. In addition each repository can expose services, e.g. for accessing logs or older versions.

#### 4.4.2. Tool integration for the robot example

One possible tool chain that can support the running example (see Section 2) is described by the TIL model in Fig. 8. It shows the three design tools (CAD, Matlab/Simulink and UML), representative users for each of the three involved engineering disciplines (mechanics, control and software) and the links between the tools that will be used according to the process. The DMLToolAdapter represents the tool, which is used for creating the dependency models for the robot. The links between the tools can be realized in various ways, as ControlChannels to denote the invocation of tool functionality, as DataChannels to denote data exchange, or as TraceChannels to denote the creation of traces at runtime



**Fig. 6.** Dependency graph showing dependencies for mechanical, control and HW/SW design of the robot. Synthesis properties are shown in blue, analysis properties in green, synthesis dependencies in orange and analysis dependencies in yellow. Red arrows are used only for the purpose of discussion. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



**Fig. 7.** A simple TIL model for illustrating the graphical concrete syntax.

of the tool chain.<sup>2</sup> Similar tool chains have been implemented, as described in detail in [6]. The reader may note that the TIL model

<sup>2</sup> Note, that the TraceChannels are not traces, instead they describe the infrastructure that the tool chain user can apply for creating traces between the respective tools at runtime.

(Fig. 8) does not capture a link between the CADToolAdapter and the UMLToolAdapter (nor in the other direction). In this particular case, the reason is that we disregarded such interrelations in the example. However, in the general case, even if such relations would exist, their inclusion in the support models would depend on the need to describe such relations. For example, even if geometrical

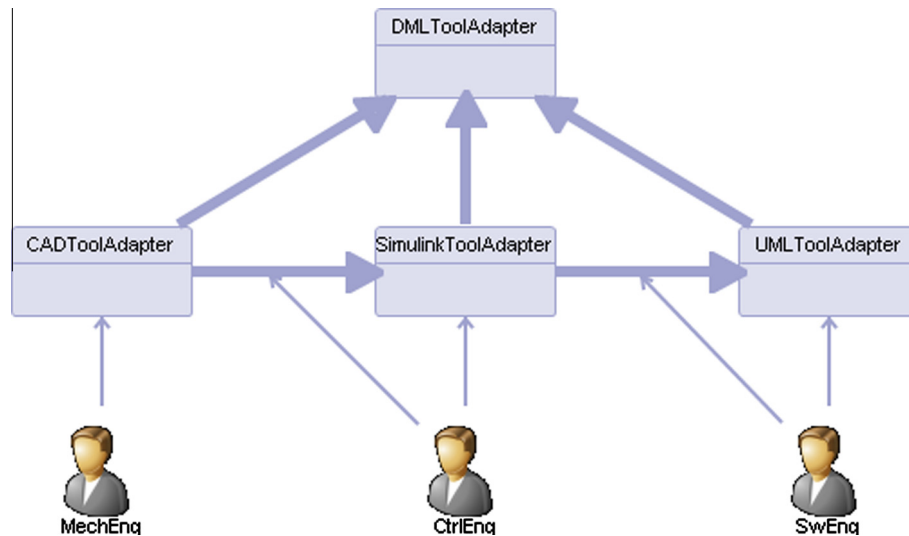


Fig. 8. The TIL model of the tool chain for the running example.

dependencies exist between electronics and mechanics, still it is not certain that tool integration that automates the handling of such dependencies is desirable.

#### 4.4.3. Relating TIL to the previously presented solutions

As we have argued in Section 1 (see especially Fig. 1), the different viewpoints for mechatronic system development need to be managed on each of the identified levels: the level of the involved people, the level of (product-) models and the level of development tools. TIL mainly targets the tool level, but does not single-mindedly focus on this level, rather it views the tool level in the wider context of the people and model levels. This is accomplished in TIL by describing some of the relationships between the levels, which are relevant for tool integration, namely the relationship between tools and people, and the relationship between tools and models: the involved tools are described by ToolAdapters, the models are described by ToolAdapterMetamodels, the people are described by TIL Users and the mechatronic development process is expressed by the composition of the different TIL concepts.

## 5. Unification proposal

This section highlights relationships between the different solutions proposed in Section 4 and explains how they can be used synergistically to attend to the mechatronic design challenges presented in Section 3. Two different engineering scenarios are first described to provide a perspective to the usage of the support models (Section 5.1) followed by a description of the relations between the solutions and recommendations for using them (Section 5.2).

### 5.1. Top-down vs. bottom-up development

Depending upon the design process, the engineering and support models can be built and utilized in a top-down or a bottom-up fashion. The order of creation and utilization of these models is discussed in the following sections.

#### 5.1.1. Top-down processes

A top-down process can be considered as a decomposition procedure where specifications for a system are first deduced and then a system is divided into its components – pertaining to the component specifications [42]. Hence, this scenario can be thought of as a “green field approach” where a new product is

developed from scratch. In the context of mechatronics design, we thus refer to a situation where neither engineering nor support models are available.

One question which then arises for a design team is whether to start with the engineering models or the support models (introduced in Section 4). As emphasized in systems engineering, see e.g. [16], it is important to have an early focus on the purpose and conceptual design of the product itself. This implies that selected engineering models focusing on requirements and architecture should be addressed first. Once one or more conceptual designs are available, it becomes essential to initiate efforts that deal with the architecting of the viewpoints. By ‘architecting viewpoints’ we refer not only to the selection of viewpoints but also to the elaboration of viewpoint relations; this is where the support models provide assistance. This approach in essence means that

- product concepts and overall architecture are emphasized first, followed by the
- architecture of the engineering environment in terms of (i) establishing viewpoints, and their (ii) interrelations.

An interesting scenario is to consider a top-down approach where a dependency model is constructed before the majority of the engineering models. Such an effort would emphasize a product level (and process) perspective and might be very useful as a basis for later viewpoint specific work. Naturally, most development efforts will be iterative and incremental, and concepts as well as the two types of architectures will evolve. However, the important take-away is that the support models will complement early architecting efforts by clarifying responsibilities, assumptions, dependencies between product properties and relations between tools across viewpoints.

Having motivated the importance of support models also in early stages, the next question is in which order contracts, dependency models, and tool integration models should be developed. A top-down approach without legacy tools, would start with contracts and dependency models. How to deal with contracts and dependency models, and their relations, is further elaborated in Section 5.2.1.

Assuming that legacy tools are already in place for the stakeholders, their interrelations can also be captured early on using tool integration models. It would then be feasible to start by developing TIL models and later on proceed to dependency and con-

tracts. Relationships between dependency and tool integration models, as well as between tool integration models and contracts are described in Sections 5.2.2 and 5.2.3 respectively.

Even in a green-field approach, an organization may already in previous developments have aggregated libraries of contracts, dependency patterns, and typical tool-chain architectures, that can be reused in developing support models. A possibility during detailed design of dependency and tool integration models, is to first elaborate these within the respective viewpoints, until the point where synchronization with other viewpoints is needed.

#### 5.1.2. Bottom-up processes

A bottom-up scenario can be considered as an integration process, where – based on the decisions taken on the chosen design solutions and the modeling performed – the designers try to find the cost of the product or address the integration issues [42]. In this context we use “bottom-up” to refer to a situation where the engineering models already exist but where the support models are built afterwards, i.e. a re-engineering effort. An example is the scenario where an existing mechatronic system needs to be changed and where it is found that there is lack of explicit descriptions of the viewpoint relations. For the purpose of understanding the impact of changes with respect to the product and/or engineering environment (e.g. a change of tools or how they interoperate), it can then be useful to develop support models.

As for the discussion for top-down processes, an organization can make a choice on which support models to emphasize. Concrete needs at hand, such as what interrelations are most pressing to clarify, will provide guidance. For example, given needs to understand impact of changes across models, it is natural to start with dependency modeling. If misunderstandings at a more conceptual level are detected, then contracts may provide the most useful starting point. With a bottom-up approach, both engineering models and tools will be in place, so starting with a TIL model provides an opportunity to clarify interactions between tools, which in turn in our experience can stimulate the identification of improvement potential (captured as new TIL models). In developing these support models, one possible approach is as follows: a dependency model and a TIL model per viewpoint can be first created until the point where synchronization with other viewpoints is needed. At that point, one could start with utilizing the contracts in order to discuss the roles and obligations corresponding to the different viewpoints, and later this information can be utilized to update the dependency model which can be supported by a tool-chain development (TIL model). The relations between the support models are elaborated in the following section.

### 5.2. Relations between support models

In the following we discuss relations between support models. Understanding such relationships carries potential benefits for designers, since one approach can complement and provide the foundation for the other, hence facilitating work.

#### 5.2.1. Relation between contracts and dependency models

Contracts define key vocabulary for communication among stakeholders, and make assumptions and obligations explicit. Dependency models, on the other hand, are used to explicitly codify relations between product properties as captured by models related to different viewpoints. In our work with contracts and dependency models we have found that there is a close relation between the two. Let us illustrate this relation through the dependency model visualization for the robot example, shown in Fig. 6. The bottom part of the Figure zooms in on a number of synthesis properties, one analysis property ( $WCET_C$ ) and two synthesis dependencies ( $SD_{16}$  and  $SD_{12}$ ). The zoom in, in particular

shows that  $SD_{16}$  and  $SD_{12}$  involve properties across the two viewpoints, i.e. that these dependencies will be making use of properties such as ( $WCET_C$ ) and  $h$ . The directed edges in Fig. 6 represent how properties and the corresponding dependencies are connected.

Two edges, highlighted in red, cross the Control and Hardware/Software viewpoints, arriving at  $SD_{16}$  and  $SD_{12}$  respectively. Taken from the Control viewpoint,  $SD_{16}$  represents the synthesis of a suitable sampling period based on closed loop performance requirements. As a result, the sampling period,  $h$ , is selected and then later on used in software design where the period is made part of the software algorithms, and used for defining a suitable period of a controller task. Since  $WCET_C$  provides a lower bound on  $h$ , it has to be used as part of the determination of  $h$ . The determination of  $WCET_C$ , as an analysis property, is illustrated at the bottom right corner in Fig. 6. The edge from  $WCET_C$  shows that it is used as part of  $SD_{16}$ . The red edges in the highlighted bottom part of Fig. 6, thus illustrate that control application requirements and implementation constraints both affect the decision on the sampling period,  $h$ . These dependencies and their relations to the properties form a loop that has to be resolved during the development; this typically includes human decision making.

Given this example of interrelations, a key observation is that the period,  $h$  at the same time constitutes a dependency model property used across the viewpoints, as well as a concept part of the vocabulary of the LET contract. This is no coincidence. Since vocabulary concepts are those that will be used in stakeholder communication, i.e. across viewpoints, they are also highly likely to be related to dependencies among properties across viewpoints. We note that the LET contract also defines basic functionalities of a controller. In the provided dependency model these functions are not explicitly included. They can instead be seen as a specification of basic vocabulary that is common to both viewpoints, and that may be useful in relation to other models, such as for co-simulation or code generation. Based on this example, we further note the following:

- Key properties and vocabulary used across viewpoints will be part of both contracts and dependency models. A dependency model will moreover describe how such a property relates to other properties part of the viewpoint models (such as  $WCET_C$  for the sampling period).
- *Dependency patterns* can be used to express known dependencies, for example the known relation between execution time and interference to yield response time (illustrated at the bottom right corner of the “Robot HW/SW design” box in Fig. 6). Dependency patterns can be used to facilitate the mappings from the contract vocabulary to viewpoint concepts.

For the case where no contract has been established, dependency modeling could provide the basis for establishing the vocabulary of contracts by analyzing dependencies across viewpoints. Conversely, if contracts are established first, dependency modeling can be used to describe how the vocabulary part of the contract link to model properties. We conclude that the two approaches are complementary. Contracts provide a more concise view compared to dependency models, and dependency patterns and validity conditions act as a bridge between contracts and dependency models.

#### 5.2.2. Relation between Dependency Models and TIL Models

Given that each engineering model is managed by a tool, the contents of the dependency model can be used to build a tool chain, which automates the data management with respect to the dependencies governing the disparate models. For example, in case of the two-DOF robot, the dependency model captures that

the plant transfer function  $G(s)$  in Simulink depends upon Inertia for each arm ( $I_A$  and  $I_B$ ) in the CAD tool. Such a relationship can now be operationalized by building a tool chain that supports the consistency between the dependency model, the CAD and Simulink models, e.g. by a model transformation that is triggered whenever the shared (dependent) property is updated in either of the two models.

In a bottom-up approach, it is assumed that the engineering models are already created and a tool chain in place. A TIL model can then be developed to describe the tool-chain interactions and be reused to support the construction of the dependency model. This involves enabling the tool chain adapters to extract the data from the tools and represent it to conform to the DML meta-model. In this way, the dependencies within the engineering models are made explicit. A dependency modeling tool can utilize this information and capture the corresponding dependencies inside the dependency model. At this point, the dependencies between engineering models are also captured, an activity which is also supported by the tool chain.

### 5.2.3. Relation between contracts and TIL models

As shown in the previous sections, dependency models act as an intermediary between contracts and TIL models. However, there are also direct relations between contracts and TIL models. To realize this, we need a mapping from the people level to the tools they are using. TIL models describe how tools and stakeholders interact, see for example the TIL model in Fig. 8. TIL models thus represent people (by the TIL concept *User*), tools (by the TIL concept *Tool-Adapter*) and connections between tools (by the TIL concepts *Data-Channel*, *TraceChannel*, *ControlChannel*). Contracts on the other hand provide a specifications among viewpoints, which refer to specific stakeholders. Given a set of contracts and a TIL model, it is possible to carry out the following analysis for a particular development setting.

- Given a contract between two viewpoints, does the TIL model provide some connection between the corresponding stakeholders? If there is a corresponding connection within the TIL model, the contract can be used as a specification to check whether the connection meets the requirements of the contract. If there is no such connection, it can be worthwhile to investigate if some level of automation through tool integration is possible.
- Given a tool integration connection in a TIL model for specific stakeholders, is there a corresponding contract available? If not, it can be assessed whether it is worthwhile to develop a contract in order to make assumptions and relations explicit. If a corresponding contract exists, it can be used for checking that the connection is compatible with the contract.

The above analysis and checking could be manual or possibly automated. The checking can be qualitative, e.g. just to indicate that there are indeed stakeholder connections for corresponding contracts. A detailed analysis can be used to assess that the vocabulary used is consistent. In order to accomplish this, the details of the inter-tool connections (such as represented by model transformations) need to be constructed to reflect the content of the respective contract.

## 6. Related work

Any complex system will involve multiple stakeholders and thus multiple views. As a consequence, multi-view systems have been studied in many different areas (e.g. in requirements, systems and software engineering, enterprise information systems, and in

model-driven engineering). For example, the so called *Controlled Requirements Expression* (CORE) from the early 80s highlights the concept of viewpoints as part of its method for determining functional requirements. CORE uses traditional structured analysis and design techniques in terms of data- and control-flow diagrams to identify relations between views [43].

Apart from applicability in multiple domains, multiview systems are also multifaceted and many perspectives can be applied to them. A large number of approaches that treat various aspects of multiview systems have therefore received attention, [44].

In this section, we give an overview of related work and briefly relate it to the contributions presented in this article. We have divided related work into a number of areas to ease the presentation. The areas are coarse grained, and for each, we attempt to describe the main characteristics of typical work in this field. As will be seen, the areas are not mutually exclusive and some build on others. The focus is naturally oriented towards MBSE approaches with relation to viewpoint integration. Out of context are organizational integration aspects such as organizational structure, training and social systems [11] and discipline specific approaches that have not been applied on a system level.

We cover the following areas:

- Providing integration-specific views,
- Processes and multiview frameworks,
- Model-driven engineering,
- Co-simulation and optimization,
- Tool integration,
- Consistency in multiview modeling.

### 6.1. Providing integration-specific views

Several approaches presented below provide dedicated languages and models (i.e. views) which have the purpose to support integration. Individually, our *support models* thus fit into this category.

#### 6.1.1. Functional models as a reference for system understanding

[22] provided a categorization of multiple views (supported by design models) and proposed function modeling as a bridging approach between views. This is a very important concept as it emphasizes the understanding of a common context and the purpose of a system. While sharing a common understanding of the system functionality will help to support integration, it provides an indirect support for dealing with viewpoint relations. It is typically after function modeling that the design process becomes domain-specific, and hence function modeling solely does not address the multi-view integration problem [9].

#### 6.1.2. Architectural models

Many approaches advocate the use of core architecture descriptions to capture the main characteristics of a system, thus providing a view which is complementary to functional models. For this purpose several Architecture Description Languages have been proposed (ADLs [45]), which have a main purpose to enhance the communication among multiple system stakeholders.<sup>3</sup> Product architecture views may provide a basis for defining or identifying shared concepts, and may also provide additional annotations to capture some assumptions and constraints. In contrast, our concept of contracts and dependency modeling has the main focus to make such shared concepts, assumptions and constraints explicit. Some approaches towards consistency of multiview systems also propose

<sup>3</sup> One example is the Architecture Analysis and Design Language (AADL), an SAE standard: <http://standards.sae.org/as5506b/>.

shared (architectural) models, see Section 6.6 for further details. Another example of establishing a common context is through A3 overviews ([46]) – which aim to provide an overview of the complete system architecture in terms of different aspects on an A3 paper, hence focusing on only a limited but useful set of information that the designer can use for communication with other designers.

#### 6.1.3. Contracts

are an essential aspect of software component-based design and interface theories [47] where components are described by their interfaces. Interfaces can also be seen as a contract between the component and its environment. The contract specifies the assumptions that the component makes on the behavior of the environment, for example that the value of a signal or flow at a certain (input) port will never exceed a certain threshold. In addition, a contract specifies the guarantees that the component provides, for example, that a certain produced output will not exceed a certain threshold. A number of formalisms have been developed that deal not only with software systems, but also with real-time and continuous dynamics aspects (e.g., see [48–51]). The related work is compatible with our concept of viewpoint contracts and some of the referenced formalisms could be used as concrete mechanisms for describing contracts. The main difference is that our contracts have been developed with the intention to reconcile and integrate viewpoints (people), whereas the main stream work on contracts for software components have focused on supporting composition of such components. In the work presented in [52], both horizontal and vertical contracts are introduced. Whereas ‘horizontal contracts’ focus on the relationships of different components at the same level of abstraction, primarily used for composition, vertical contracts focus on the relationships between components at different levels of abstraction such as between specifications and implementations. We share this idea, and the viewpoint contracts we introduced with the robot example illustrated both horizontal as well as vertical contracts.

#### 6.1.4. System modeling describing relations between properties

Work in this area attempts to provide a ‘system model’ to explicitly describe interrelations, thus providing approaches which at least to some extent are alternative to our proposed support models. System modeling is based on the use of systems modeling languages such as SysML and specific architectural description languages such as EAST-ADL. Taking the example of SysML, it includes a number of views as part of the language as well as means to express relations between modeling entities such as satisfy, verify, association and allocation relationships. The resulting system view can also be integrated with other views by adding profile extensions in SysML, e.g. [34,35,53]. In this way, a SysML model can thus provide a “unified” system model that incorporates modeling elements representing different views and relationships between them. As mentioned in Section 4.1, we instead favor the use of dedicated (domain specific) modeling languages as opposed to general purpose modeling languages. A common system model in SysML does not necessarily make the interrelations explicit. For instance, dependencies are not explicitly captured in SysML due to a lack of sufficiently expressive language constructs [23]. While new language constructs can be defined by extending the language, such extensions are still restrictive in terms of language design, e.g. restriction in concrete syntax. We believe that this often leads to additional accidental complexity and a better way is a solution composed of multiple DSLs. Nevertheless, our methodological findings could also be used as a basis for formulating SysML based solutions – which in essence then would represent another formalism for capturing the support models.

A few other solutions have been proposed for dedicated dependency modeling and management: through Design Structure Ma-

trix (DSM) ([54]), based on correspondence graphs supported by model transformations ([35]), or through Domain-Specific Languages (DSL) such as SysML to Modelica Transformation ([55]). However, most of the solutions either focus only on implicit management of dependencies, or fail to consider that dependencies can be captured at different levels of detail, and dependency modeling should be appropriately supported ([23]). Another direction related to dependency modeling uses architectural models as a basis for maintaining consistency with “domain” models, see for example [56,57]. However, these approaches do not provide an explicit language for modeling the dependencies. Another related work is the Contact and Channel Approach (C&C-A) [58] used to create a system architecture model which graphically represents a system configuration with underlying interrelations between function and form. Function modeling is performed (utilizing a function database) where such interrelations are explicitly captured. A part library is then utilized to create a possible configuration of components, and to further concretize the functional constraints.

#### 6.2. Processes and multiview frameworks

In terms of managing a mechatronic design process, the VDI2206 ([59]) process covers the design from the initial conceptualization to the verification and validation phases. It does not however explicitly address how to deal with relations between viewpoints.

Interactions between levels (from societal constraints over management to developers and tools) have been addressed in systems engineering work and in particular in the work on system safety, where failing interactions (and the lack of understanding of constraints for interactions) can lead to accidents, see e.g. ([10,60]).

Architectural frameworks define different types of views, that together provide a description of some system, see e.g. [57,61,62]. The included views have to be understood in the context of the goals for the respective framework. In general, such multiview frameworks place less emphasis on the relations between the involved models.

#### 6.3. Model-driven engineering

Model-driven engineering encompasses a number of principles and techniques including modeling language design, model transformations and process modeling (see e.g. [16]).

Model transformation [63] is a mechanism for describing and enforcing relations in MVM. Once the relation between different models is specified, it can be described in terms of model transformations. Model transformations provide a precise and executable descriptions. They can be used to create new models that fulfill the relation or to enforce a relation between existing models.

Process modeling is also promoted within MDE. Traditionally such languages have been used only to describe processes but they are being extended in order to express relations between models – thus being relevant for viewpoint integration. A representative example is given by the Formalism Transformation Graph and Process Model (FTG + PM) [64]. The FTG + PM combines a so called mega-model with a process model. The mega-model captures the involved languages and transformation definitions. The process model, modeled with UML activities, uses instances of the languages and transformations in the mega-model to create a process with control- and dataflow. FTG + PM models can be enacted. As such, MDE provides a number of techniques that are useful for viewpoint integration (we for example use model transformations in conjunction with TIL).

#### 6.4. Co-simulation and optimization

A large number of approaches aimed at addressing viewpoint relations stem from the areas of co-design, co-simulation, trade-off analysis and optimization. ([65]) provided an example of mechanical and control system co-design for a DC motor. [66] presented a topology optimization approach managing the coupling between dynamic, mechanical and control system properties through co-simulation. Standards for co-simulation are also making progress, as for example manifested by the FMI standard for dynamic systems.<sup>4</sup> Many examples of co-simulation stem from the field of embedded systems such as ([67–69]). [70] explored an optimization approach based on formally capturing the design problem in SysML and searching (along with optimizing) the candidate solutions based on algebraic model of components. ([71–73]) are other examples of integrated design and optimization of control and mechanical systems of a product.

Co-simulation approaches are essential to support early evaluation of integrated behavior. Such evaluation does not, however, directly address misconceptions among stakeholders (people level) and pinpoint dependencies between viewpoints, although problems may be seen through improper system behavior. Contracts and dependency models can help to avoid misunderstandings in the first place, and in a latter stage be used for analyzing what went wrong. Standard exchange formats constitute agreed common data models for specific scenarios, such as co-simulation. This implies that contracts and exchange standards (such as FMI for co-simulation) should be closely correlated for such scenarios. In this sense, standards for exchange of simulation models are strong candidates for use in data exchange, and the corresponding meta-models can be incorporated as part of our tool integration models. Finally, co-simulation does not provide a generic technique for tool integration capabilities such as data reuse, traceability and notifications. Co-simulation is thus complementary to the solutions we have provided.

Trade-offs are essentially inherent in a system with multiple viewpoints and thus widely studied in systems engineering and mechatronics. Qualities, such as reliability, performance and cost, will depend on multiple design parameters as discussed in this paper. Different solutions will have advantages and disadvantages in comparison to each other, depending on which stakeholder's point of view is taken. Quantitative trade-off analysis allows the comparison of alternative design solutions in regards to multiple metrics. A common approach is to introduce mathematical trade-off functions that reduces the optimization problem to a single optimization goal, where partial metrics are combined into an overall one (for example through a linear combination) [74]. We believe that criteria and trade-off functions used in optimization could very well fit as part of viewpoint contracts.

An interesting heuristic technique developed to deal with trade-off analysis is that of ATAM, the Architecture Trade-off Analysis Method. In ATAM, quality attributes such as reliability and usability, and their relations to architectural properties, are identified with the purpose to develop an understanding of the architectural properties that will constitute trade-off points. In this sense, ATAM provides a method that helps to identify shared concepts, which in turn could help in defining architectural models, as well as contracts and dependency models.<sup>5</sup>

Our viewpoint integration techniques provide a basis and support for the introduction of trade-off analysis and conflict resolution techniques. Our solutions (support models) help in establishing a common ground with the aim to decouple the development to a certain degree, and pinpoint the conflicts as they arise. Later, impact analysis can be performed to define a conflict

resolution strategy. Conflict resolution in engineering design is a research subject on its own. Some of the related work in this area is based on design rationale capture [75] and through consensus modeling [76]. However, in this paper we do not explicitly deal with conflict resolution per se.

#### 6.5. Tool integration

Tool integration is the activity of producing an *engineering environment* of tools that supports the systems development process ([77]). Characteristically for this situation, the tools are available first, are built without considering integration, and their integration is performed afterwards. This situation is also described as a *posteriori integration* ([78]), in contrast to a hypothetical *a priori integration*, where tools are built so they fulfill specific integration requirements. Some tools may provide limited capabilities for a priori integration with a limited set of tools, but in general, the need for a posteriori integration can be assumed, since mostly commercial off-the-shelf (COTS) development tools are used ([79]). In its simplest form, the engineering environment may be realized as a selection of tools and a set of guidelines that users of the tools, i.e. the developers, need to follow to realize the integration. In this case, integration is performed by human operators, most likely the developers themselves. This situation results in inefficiencies and high costs of development ([80]), and without any formal framework for capturing potential dependencies across the engineering disciplines covered by the integrated tools. More effective engineering environments, called *tool chains*, are realized as a software solution. Tool chains are effective, since integration-related activities are automated. This allows systems developers to focus on the creation of the product instead of manually performing the integration. Tool chains are not necessarily connected in a linear way, but can form networks of tools. Tool chains can enable higher reuse of development artifacts across tools and they enable the automation of development activities. The assumption of the research community is that the main benefits of using tool chains are increased productivity ([80]), cost savings, reduced human error and reduced time-to-market ([81]).

#### 6.6. Consistency in multiview modeling

Ensuring consistency between multiple views of a system is a big challenge. As stated by Finkelstein [24], inconsistencies in a real design setting are difficult to detect – there could be statements which are not semantically grounded or that inconsistency is distributed across many assertions. The information itself is changing rapidly and not all inconsistencies require an action, for instance an inconsistency that does not effect a designer's verdict on a typical requirement. Commonly studied challenges for managing multi-view consistency are mentioned by [82,83]. Among the solutions, the use of description or prepositional logic to maintain consistency is profound, e.g. [84–86]. Others have proposed rule-based consistency checking such as [87,88]. Another direction uses architectural models as a basis for maintaining consistency with “domain” models, for example [56,57]. Nevertheless, inconsistency management is a topic of ongoing research, and is especially relevant for multi-view modeling environments. Our proposed solutions provide a basis for detecting when inconsistencies can occur and for implementing (partial) automatic solutions such as notifications and information exchange – thus providing solutions which can help in designing solutions for consistency management.

### 7. Discussion

Establishing efficient concurrent engineering in a mechatronics setting, clearly requires addressing many concerns relating to

<sup>4</sup> Functional Mock-up Interface (FMI) – <https://www.fmi-standard.org/downloads>.

<sup>5</sup> For ATAM, see for example <http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm>.

viewpoint integration. We have provided a model-based approach that makes interrelations explicit at three levels: people, models and tools. Our support models complement other solutions such as co-simulation techniques and the provisioning of core functional models to increase product understanding across a range of stakeholders. The use of modeling languages, such as SysML, potentially provides another way to implement parts of our support models.

Providing suitable methods and techniques for dealing with interrelations paves way for

- Effective concurrent engineering. By understanding and managing interrelations, a higher degree of controlled concurrency is obtainable since it becomes possible to understand when synchronization (among people, models and tools) is necessary.
- Automated Computer Aided Engineering (CAE) support for mechatronics design including optimization and efficient information management to enhance quality, and reduce inconsistencies.
- Better planned and designed engineering environments, since the interrelation models will pin-point what type of integration is needed and how to go about it during the design process (see Section 5.1).

As with all model-based approaches, a particular organizational setting with its needs should guide the usage of models, [89]. The presented approach is modular in that parts of the solutions (contracts, dependency modeling or tool integration modeling) could be applied depending on the particular needs. Moreover, the solutions could already be applied as informal techniques to support social integration, i.e. the techniques can assist inter-team discussions about common vocabulary and by developing informal dependency and tool integration models. In our experience, using the models for white-board discussions were helpful in identifying and highlighting the relationships between viewpoints. Such an informal adoption may be more suitable for less stable product settings, for example as part of an initial design synthesis phase (where the design space is searched for possible solutions). The use of formalized support models may be inefficient in such a phase where the dynamics of decision making and the rate at which the changes are introduced is high.

By use of formalized (support and engineering) models, design automation becomes possible. However, this will require adequate tools to be made available for capturing and managing support models, and for leveraging them for analysis and synthesis. In our current work we have developed prototype tools for dependency modeling – Dependency Modeler [5] and tool-integration (TIL) modeling – the TIL Workbench [6].

Clearly defined organizational responsibilities are essential for dealing with the integration of viewpoints. As support models and tools are introduced to assist with the integration, even further stakeholders are introduced. It then becomes necessary to manage the support models and tools, and for this reason, *support processes*, i.e. processes to support the development of the support models (that is, contracts, dependency models and TIL models), also need to be elaborated.

## 8. Conclusions

Relating back to Fig. 1, we have proposed solutions dealing with viewpoint relations for each of the people, model and tool level. Contracts provide a common vocabulary and constraints which in turn support people communication, decoupling of viewpoint processes, dependency modeling, co-simulation and potentially

other model-based design scenarios. Dependency modeling provides an explicit view of product properties and dependencies, and provides means to manage dependencies between these properties. In addition, it also supports management of different work-flows (concurrent engineering), change management and consistency checking. The visualization of the dependency model shown in Fig. 6 clearly illustrates both intra- and inter-viewpoint dependencies, and points to the need for understanding and managing such complex dependencies. TIL provides an ability to model integration between tools. TIL models can be used for systematic development and description of tool chains.

Viewpoint integration is a multifaceted problem that involves many stakeholders. Apart from domain/disciplinary engineers and managers, we identified needs for new stakeholders that take responsibility for the support models. Similarly, there are needs for tools and processes dedicated to the engineering of support models.

We used a robotics case study to describe our solutions and to elaborate and illustrate how they can support each-other. The actual implementation of the case study is partial in the sense that we did not fully implement all these solutions for the same system.

Contracts, dependency modeling and tool integration models, provide different and complementary views of interrelations. While contracts emphasize conceptual alignment (a given contract could be valid for many products), dependency models focus on relations between concrete properties part of a specific product. A dependency modeling tool can be made part of a tool chain where its interactions are described through a tool integration model. Tool integration technologies can leverage existing engineering models to provide a desired level of automation of the engineering processes. A tool integration model will include tools and their users, relating back to the contracts. It is clear that the support models are partly overlapping. This can be utilized in the development of the support models, for example, by developing or checking one model based on the other(s). There is a lot of potential in developing tool support that takes such synergies into account. Let us give a few examples here:

- Making a dependency modeling tool available and well integrated as part of a tool-chain, opens up a large number of opportunities. The dependency modeling tool can be used for impact analysis and for signaling conflicts during changes in engineering models.
- We see an opportunity to apply dependency modeling and tool integration techniques to the support models themselves. For example, when a dependency model has been developed, there is a need to keep track of what properties correspond to the vocabulary of the contract; dependency models can be used to make such relations explicit. One capability of tool integration is related to traceability among data, dispersed over different tools. Together, TIL and dependency models provide the basis for generating 'traceability' information, combining information about properties and dependencies, with information about their tool and storage context.
- Tool integration solutions can be used to operationalize the dependencies across support models, for example by supporting traceability and notifications upon changes in support models.

In our work, we have mainly addressed the development stages of mechatronic products, and are now planning to evaluate and extend the usage of support models to also include other life-cycle stages. These topics together with larger scale evaluation will be part of future work.

## Acknowledgement

The authors wish to acknowledge the anonymous reviewers for their helpful suggestions.

## References

- [1] ISO. Software and systems engineering – architecture description (ISO/IEC/IEEE 42010). Tech. rep., ISO; 2011.
- [2] Estefan JA. Survey of model-based systems engineering (MBSE) methodologies. Revision B. Tech. rep., Jet Propulsion Laboratory, California Institute of Technology; 2008.
- [3] Törngren M, Tripakis S, Derler P, Lee EA. Design contracts for cyber-physical systems: making timing assumptions explicit. Tech. rep. UCB/Eecs-2012-191, EECs Department, University of California, Berkeley; 2012. <<http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-191.html>>.
- [4] Derler P, Lee EA, Törngren M, Tripakis S. Cyber-physical system design contracts. In: ICCPS '13: ACM/IEEE 4th international conference on cyber-physical systems; 2013. <<http://chess.eecs.berkeley.edu/pubs/959.html>>.
- [5] Qamar A. Model and dependency management in mechatronic design. Ph.D. thesis, KTH – Royal Institute of Technology; 2013.
- [6] Biehl M, El-Khoury J, Loiret F, Törngren M. On the modeling and generation of service-oriented tool chains. *J Softw Syst Model* 2012;0275. <http://dx.doi.org/10.1007/s10270-012-0275-7>.
- [7] Sussman JM. Collected views on complexity in systems. Tech. rep. ESD-WP-2003-01.06, Royal Institute of Technology (KTH); 2003. <<http://esd.mit.edu/WPS/internal-symposium/esd-wp-2003-01.06.pdf>>.
- [8] Brooks FP. No silver bullet essence and accidents of software engineering. *Computer* 1987;20(4):10–9. <http://dx.doi.org/10.1109/MC.1987.1663532>.
- [9] Torrey-Smith JM, Qamar A, Achiche S, Wikander J, Mortensen NH, During C. Challenges in designing mechatronic systems. *J Mech Des* 2013;135(1):011005. <http://dx.doi.org/10.1115/1.4007929>.
- [10] Leveson N. *Engineering a safer world*. MIT Press; 2012. <<http://mitpress.mit.edu/books/engineering-safer-world>>. ISBN: 9780262016629.
- [11] Adamsson N. Interdisciplinary integration in complex product development: managerial implications of embedding software in manufactured goods. Ph.D. thesis, KTH – Royal Institute of Technology; 2007.
- [12] Malvius D. Integrated information management in complex product development. Ph.D. thesis, KTH – Royal Institute of Technology; 2009.
- [13] Hehenberger P. Proceedings 1st workshop on mechatronic design. Johannes Kepler University, Linz; 2012. <<http://mechatronic-design.jku.at>>.
- [14] van Amerongen J. Mechatronic design. *Mechatronics* 2003;13(10):1045–66. [http://dx.doi.org/10.1016/S0957-4158\(03\)00042-4](http://dx.doi.org/10.1016/S0957-4158(03)00042-4).
- [15] Frey E, Ostrosi E, Roucoules L, Gomes S. Multi-domain product modelling: from requirements to CAD and simulation tools. In: 17th International conference on engineering design (ICED'09). Stanford, CA, USA: Desing Society; 2009. p. 477–88.
- [16] Oliver DW, Kelliher TP, James. *Engineering complex systems with models and objects*. McGraw-Hill; 1997. ISBN: 0070481881. <<http://www.worldcat.org/isbn/0070481881>>.
- [17] Tjalve E. Systematic design of industrial products. Institute of Product Development, The Technical University of Denmark; 2003. ISBN: 9788798136019.
- [18] Siemens PLM Software. Solid edge; 2011. <[http://www.plm.automation.siemens.com/en\\_us/products/velocity/solidedge/index.shtml](http://www.plm.automation.siemens.com/en_us/products/velocity/solidedge/index.shtml)>.
- [19] Mathworks. Matlab/Simulink; 2013. <<http://www.mathworks.com/products/simulink>>.
- [20] IBM. Rational rhapsody developer; 2013. <<http://www-142.ibm.com/softw/ware/products/us/en/ratirhap>>.
- [21] Tomiyama T, D'Amelio V, Urbanic J, ElMaraghy W. Complexity of multi-disciplinary design. *CIRP Ann – Manuf Technol* 2007;56(1):185–8. <http://dx.doi.org/10.1016/j.cirp.2007.05.044>.
- [22] Buur J, Andreasen MM. Design models in mechatronic product development. *Des Stud* 1989;10(3):155–62. [http://dx.doi.org/10.1016/0142-694X\(89\)90033-1](http://dx.doi.org/10.1016/0142-694X(89)90033-1).
- [23] Qamar A, Paredis CJJ, Wikander J, During C. Dependency modeling and model management in mechatronic design. *J Comput Inform Sci Eng* 2012;12(4):041009. <http://dx.doi.org/10.1115/1.4007986>.
- [24] Finkelstein A. A foolish consistency: technical challenges in consistency management. In: 11th International conference on database and expert systems applications. London, UK: Springer-Verlag; 2000. p. 1–5. <<http://dl.acm.org/citation.cfm?id=648313.755542>>.
- [25] Pratt MJ. Introduction to ISO 10303—the STEP standard for product data exchange. *J Comput Inform Sci Eng* 2001;1(1):102–3. <http://dx.doi.org/10.1115/1.1354995>.
- [26] Shaw M, Garland D. *Software architecture*. Prentice Hall; 1996.
- [27] Tratt L. Model transformations and tool integration. *Softw Syst Model* 2005;4(2):112–22. <http://dx.doi.org/10.1007/s10270-004-0070-1>.
- [28] Christie A, Levine L, Morris EJ, Riddle B, Zubrow D. Software process automation: interviews, survey, and workshop results. Tech. rep., SEI; 1997. <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.9896>>.
- [29] Derler P, Lee EA, Vincentelli AS. Modeling cyber-physical systems. *Proc IEEE* 2012;100(1):13–28. <http://dx.doi.org/10.1109/JPROC.2011.2160929>.
- [30] Frost R. Jazz and the Eclipse way of collaboration. *IEEE Softw* 2007;24(6):114–7. <http://dx.doi.org/10.1109/MS.2007.170>.
- [31] Hein C, Ritter T, Wagner M. Model-driven tool integration with ModelBus. In: Workshop future trends of model-driven development; 2009.
- [32] Atkinson C, Kühne T. Reducing accidental complexity in domain models. *Softw Syst Model* 2008;7:345–59. <http://dx.doi.org/10.1007/s10270-007-0061-0>.
- [33] Object Management Group. OMG systems modeling language specification V1.3; 2012a. <<http://www.omg.org/spec/SysML/1.3/>>.
- [34] Shah AA, Kerzhner AA, Schaefer D, Paredis CJJ. Multi-view modeling to support embedded systems engineering in SysML. In: Engels G, Lewerentz C, Schäfer W, Schürr A, Westfechtel B, editors. *Graph transformations and model driven engineering. Lecture notes in computer science*, vol. 5765. Berlin, Heidelberg: Springer; 2010. ISBN 978-3-642-17321-9. p. 580–601.
- [35] Cao Y, Liu Y, Paredis CJJ. System-level model integration of design and simulation for mechatronic systems based on SysML. *Mechatronics* 2011;21(6):1063–75.
- [36] Brooks FP. No silver bullet – essence and accident in software engineering. *IEEE Comput* 1987;20(4):10–9.
- [37] Vallecillo A. On the combination of domain specific modeling languages. In: Modeling foundations and applications. Lecture notes in computer science, vol. 6138; 2010. p. 305–20. doi:10.1007/978-3-642-13595-8\_24.
- [38] Mosterman PJ, Vangheluwe H. Computer automated multi-paradigm modeling: an introduction. *Simul: Trans Soc Model Simul Int* 2004;80(9):433–50. <http://dx.doi.org/10.1177/0037549704050532>.
- [39] Bini E, Cervin A. Delay-aware period assignment in control systems. In: *Proceedings of the 2008 real-time systems symposium (RTSS'08)*. Washington, DC, USA: IEEE Computer Society; 2008. ISBN 978-0-7695-3477-0. p. 291–300. <http://dx.doi.org/10.1109/RTSS.2008.45>.
- [40] Hazelrigg GA. A framework for decision based engineering design. *J Mech Des* 1998;120(4):653–8.
- [41] Biehl M. Semantic anchoring of TIL. Tech. rep. ISRN/KTH/MMK/R-12/19-SE, Royal Institute of Technology; 2012. <<http://www1.md.kth.se/biehl/files/papers/semantics.pdf>>.
- [42] Takai S, Jikar VK, Ragsdell KM. An approach toward integrating top-down and bottom-up product concept and design selection. *J Mech Des* 2011;133(7):071007. <http://dx.doi.org/10.1115/1.4004233>.
- [43] Nuseibeh B, Finkelstein A. ViewPoints: a vehicle for method and tool integration. In: *Proceedings of the fifth international workshop on computer-aided software engineering*. IEEE; 1992. ISBN 0-8186-2960-6. p. 50–60. <<http://dx.doi.org/10.1109/CASE.1992.200130>>.
- [44] Persson M, Törngren M, Qamar A, Westman J, Biehl M, Tripakis S, et al. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In: *Emsoft 2013 – int. conference on embedded software*. ACM; 2013.
- [45] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description language. *IEEE Trans Softw Eng* 2000;26(1):70–93.
- [46] Borches PD. A3 architecture overviews – harvesting architectural knowledge to enhance evolvability of embedded systems. In: van de Laar P, Punter T, editors. *Views on evolvability of embedded systems*. Embedded systems. Springer; 2011. ISBN 978-90-481-9848-1. p. 121–36.
- [47] de Alfaro L, Henzinger T. Interface theories for component-based design. In: *EMSOFT'01. LNCS*, vol. 2211. Springer; 2001.
- [48] de Alfaro L, Henzinger TA, Stoelinga MIA. Timed interfaces. In: *EMSOFT'02: 2nd intl. workshop on embedded software*. LNCS. Springer; 2002. p. 108–22.
- [49] Alur R, Weiss G. Regular specifications of resource requirements for embedded control software. In: *IEEE real-time and embedded technology and applications symposium*; 2008. p. 159–68.
- [50] Benveniste A, Caillaud B, Passerone R. Multi-viewpoint state machines for rich component models. In: Mosterman P, Nicolescu G, editors. *Model-based design for embedded systems*. CRC Press; 2009.
- [51] Sun X, Nuzzo P, Wu CC, Sangiovanni-Vincentelli A. Contract-based system-level composition of analog circuits. In: *DAC'09. ACM*; 2009. p. 605–10.
- [52] Sangiovanni-Vincentelli A, Damm W, Passerone R, Taming Dr. Frankenstein: contract-based design for cyber-physical systems. *Eur J Control* 2012;18(3):217–238.
- [53] Thramboulidis K. The 3+1 SysML view-model in model integrated mechatronics. *J Softw Eng Appl* 2010;3(02):109–18. <[http://www.scirp.org/Journal/PaperDownload.aspx?paperID=1354&fileName=JSEA20100200002\\_27803637.pdf](http://www.scirp.org/Journal/PaperDownload.aspx?paperID=1354&fileName=JSEA20100200002_27803637.pdf)>. doi:10.4236/jsea.2010.32014.
- [54] Eppinger SD, Browning TR. Design structure matrix methods and applications. Engineering systems. MIT Press; 2012. ISBN: 9780262017527. <<http://books.google.se/books?id=MPQoumoGXHYC>>.
- [55] Object Management Group. SysML – modelica transformation (SyM) V1.0; 2012b. <<http://www.omg.org/spec/SyM/1.0/>>.

- [56] Bhawe A, Krogh BH, Garlan D, Schmerl B. View consistency in architectures for cyber-physical systems. In: 2011 IEEE/ACM second international conference on cyber-physical systems. IEEE; 2011. ISBN 978-1-61284-640-8. p. 151–60. <http://dx.doi.org/10.1109/ICCPS.2011.17>.
- [57] Gausemeier Jurgen, Kahl Sascha. Architecture and design methodology of self-optimizing mechatronic systems. Rijeka, Croatia: Intech; 2010.
- [58] Albers A, Braun A, Sadowski E, Wynn DC, Wyatt DF, Clarkson PJ. System architecture modeling in a software tool based on the contact and channel approach (C&C-A). *J Mech Des* 2011;133(10):101006–8. <http://dx.doi.org/10.1115/1.4004971>.
- [59] Association of German Engineers. VDI 2206, design methodology for mechatronic systems. Berlin: Beuth Verlag; 2004.
- [60] Asplund F. Tool integration and safety – a foundation for analysing the impact of tool integration on non-functional properties licentiate thesis, October 2012, Stockholm, Sweden. Ph.D. thesis, KTH – Royal Institute of Technology; 2012.
- [61] Siegers R. The abcs of afs: understanding architecture frameworks. In: Proceedings of INCOSE international symposium, Rochester, NY, July 10–15; 2005.
- [62] Mark Maier. The art of systems architecting. Boca Raton: CRC Press; 2004.
- [63] Biehl M. Literature study on model transformations. Tech. rep. ISRN/KTH/MMK/R-10/07-SE, Royal Institute of Technology; 2010.
- [64] Levi L, Mustafiz S, Denil J, Vangheluwe H, Jukss M. FTG + PM: an integrated framework for investigating model transformation chains. *SDL 2013: model-driven dependability engineering*. Lecture notes in computer science; 2013. p. 7916:182–202. doi:10.1007/978-3-642-38911-5\_11.
- [65] Reyer Ja, Papalambros PY. Combined optimal design and control with application to an electric DC motor. *J Mech Des* 2002;124(2):183. <http://link.aip.org/link/JMDEDB/v124/i2/p183/s1&Agg=doi>. doi:10.1115/1.1460904.
- [66] Albers A, Otnad J. Integrated structural and controller optimization in dynamic mechatronic systems. *J Mech Des* 2010;132(4):41008. <http://dx.doi.org/10.1115/1.4001380>.
- [67] Kawahara R, Dotan D, Sakairi T, Kirshin A. Verification of embedded system's specification using collaborative simulation of SysML and simulink models. In: International conference on model-based systems engineering; 2009. p. 21–28. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5031716>.
- [68] Brisolara LB, Oliveira MFS, Nascimento FAM, Carro L, Wagner FR. Using UML as a front-end for an efficient simulink-based multithread code generation targeting MPSoCs. In: 4th international UML DAC workshop. California, USA; 2007. p. 11–6.
- [69] Reichmann C, Gebauer D, Müller-Glaser KD. Model level coupling of heterogeneous embedded systems. In: 2nd RTAS workshop on model-driven embedded systems; 2004.
- [70] Kerzhner AA. Using logic-based approaches to explore system architectures for systems engineering. Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, USA; 2012. <https://smartechn.gatech.edu/handle/1853/44748>.
- [71] Ravichandran T, Wang D, Heppler G. Simultaneous plant – controller design optimization of a two-link planar manipulator. *Mechatronics* 2006;16(3–4):233–42. <http://dx.doi.org/10.1016/j.mechatronics.2005.09.008>. <http://linkinghub.elsevier.com/retrieve/pii/S095741580500139X>.
- [72] da Silva MM, Bröls O, Desmet W, Van Brussel H. Integrated structure and control design for mechatronic systems with configuration-dependent dynamics. *Mechatronics* 2009;19(6):1016–25. <http://dx.doi.org/10.1016/j.mechatronics.2009.06.006>. <http://linkinghub.elsevier.com/retrieve/pii/S0957415809001196>.
- [73] Yan HS, Yan GJ. Integrated control and mechanism design for the variable input-speed servo four-bar linkages. *Mechatronics* 2009;19(2):274–85. <http://dx.doi.org/10.1016/j.mechatronics.2008.07.008>. <http://linkinghub.elsevier.com/retrieve/pii/S0957415808001207>.
- [74] Shell T. The synthesis of optimal systems design solutions. *Syst Eng* 2003;6(2):92–105. <http://dx.doi.org/10.1002/sys.10037>. <http://doi.wiley.com/10.1002/sys.10037>.
- [75] Klein M, Lu SCY. Conflict resolution in cooperative design. *Artif Intell Eng* 1989;4(4):168–80. <http://www.sciencedirect.com/science/article/pii/0954181089900137>. [http://dx.doi.org/10.1016/0954-1810\(89\)90013-7](http://dx.doi.org/10.1016/0954-1810(89)90013-7).
- [76] Ostrosi E, Haxhijaj L, Fukuda S. Fuzzy modelling of consensus during design conflict resolution. *Res Eng Des* 2011;23(1):53–70. <http://dx.doi.org/10.1007/s00163-011-0114-9>. <http://link.springer.com/10.1007/s00163-011-0114-9>.
- [77] Wasserman AL. Tool integration in software engineering environments. In: Long F, editor. Software engineering environments, international workshop on environments proceedings. Lecture notes in computer science. Springer-Verlag; 1989. p. 137–49. <http://www.springerlink.com/content/p582q2n825k87nl5/>.
- [78] Brown AW, Carney DJ, Morris EJ, Smith DB, Zarrella PF. Principles of CASE tool integration. New York, NY, USA: Oxford University Press, Inc.; 1994. ISBN: 0-19-509478-6. <http://portal.acm.org/citation.cfm?id=199265>.
- [79] Broy M, Feilkas M, Herrmannsdoerfer M, Merenda S, Ratiu D. Seamless model-based development: from isolated tools to integrated model engineering environments. *Proc IEEE* 2010;98(4):526–45. <http://dx.doi.org/10.1109/JPROC.2009.2037771>.
- [80] Wicks M, Dewar R. A new research agenda for tool integration. *J Syst Softw* 2007;80(9):1569–85. <http://dx.doi.org/10.1016/j.jss.2007.03.089>. <http://dx.doi.org/10.1016/j.jss.2007.03.089>.
- [81] Yang Y, Han J. Classification of and experimentation on tool interfacing in software development environments. In: Proceedings of the third Asia-Pacific software engineering conference. APSEC '96. Washington, DC, USA: IEEE Computer Society; 1996. ISBN 0-8186-7638-8. [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=566740&#38;tag=1](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=566740&#38;tag=1).
- [82] Dijkman R. Multi-viewpoint architectural design: consistency in multi-viewpoint architectural design. Ph.D. thesis, University of Twente, Netherlands; 2006.
- [83] Engels G, Heckel R, Taentzer G, Ehrig H. A combined reference model- and view-based approach to system specification. *Int J Softw Eng Knowl Eng* 1994.
- [84] Schätz B, Braun P, Huber F, Wisspeintner A. Consistency in model-based development. In: IEEE international conference and workshop on the engineering of computer-based systems. IEEE; 2003. ISBN 0769519172.
- [85] Simmonds J, Van Der Straeten R, Jonckers V, Mens T. Maintaining consistency between UML models with description logic. In: Jérôme E, Bernard C, editors. *L'OBJET*, vol. 10; 2004. p. 231–44.
- [86] Straeten R.V.D, Mens T, Simmonds J, Jonckers V. Using description logic to maintain consistency between UML models. *UML 2003, the unified modeling language modeling languages and applications*. Lecture notes in computer science; 2003. p. 2863:326–40. doi:10.1007/978-3-540-45221-8\_28.
- [87] Mens T, Straeten RvD, D'Hondt M. Detecting and resolving model inconsistencies using transformation dependency analysis. In: Model driven engineering languages and systems. Lecture notes in computer science, vol. 4199. Berlin, Heidelberg: Springer; 2006. ISBN 978-3-540-45772-5. p. 200–14.
- [88] Van Der Straeten R, D'Hondt M. Model refactorings through rule-based inconsistency resolution. In: Proceedings of the 2006 ACM symposium on applied computing, SAC '06. New York, NY, USA: ACM; 2006. ISBN 1-59593-108-2. p. 1210–7. <http://doi.acm.org/10.1145/1141277.1141564>.
- [89] Törngren M, Chen D, Malvius D, Axelsson J. Model-based development of automotive embedded systems. CRC Press; 2008. ISBN: 9780849380266.
- [90] Qamar A, Herzig SJJ, Paredis CJJ. A domain-specific language for dependency management in model-based systems engineering. In: 7th workshop on multi-paradigm modeling, proceedings of international conference on modeling foundations and applications. FL, USA: Miami; 2013.