Standard Article

Concurrent Engineering: Research and Applications
1–14
© The Author(s) 2015
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1063293X15585008
cer.sagepub.com
SAGE

Variability and consistency in mechatronic design

Daniela Lettner[1], Peter Hehenberger[2], Alexander Nöhrer[3], Klaus Anzengruber[2], Paul Grünbacher[1], Michael Mayrhofer[2] and Alexander Egyed[1]

Abstract
Mechatronic products combine hardware and software; and today, much of software engineering is directly or indirectly involved to support mechatronic design. Due to the high level of standardization, especially among its hardware, the design of mechatronic products is strongly characterized by integrating standardized components and thus seems an ideal environment for product line engineering techniques, which allow dealing with the variability of reusable components if fully definable a priori. However, while many aspects of mechatronic design are standardized, there is also the need for the continuous construction of new components. However, today, it is difficult to seamlessly integrate the reuse of standardized components with the development of new components. This article presents a model-based approach for integrating component variants with user-defined components to better support the mechatronic design process. We present a combination of existing approaches addressing certain needs of the mechatronic design domain, for example, the integration of components is ensured through incremental consistency checking. Our approach is illustrated using the example of a basic articulated robot.

Keywords
mechatronic design, modularization, variability, consistency

Introduction

Mechatronic product development relies on the ability to integrate several disciplines (Schäfer and Wehrheim, 2007). One important stage in the development of mechatronic systems is architectural design, that is, the definition of a modular system structure allowing rudimentary simulations. Mechatronic system architectures are described in terms of modules (e.g. parts and components) and their relations (Kusiak, 2002), similar to software architectures (e.g. Alvarez Cabrera et al., 2011; Medvidovic and Taylor, 2000). Mechatronic designers consider the modules' properties and interfaces when defining the system topology, that is, the modules' physical and logical arrangement. According to the reports in the literature (Bishop, 2007; Pahl et al., 2007), up to 80% of the modules in mechatronic architectures are standardized and reused in current mechatronic products. This means that new products are developed by composing existing standardized modules and providing new capabilities via custom-made or user-defined modules.

Imagine a company building a robot, which requires a hydraulic engine. A supplier company of hydraulic engines may offer a portfolio of variants—a product line (PL). Using such an engine in a mechatronic product is then alike reusing from their PL. Naturally, a robot requires not only an engine. Building a robot relies on reusing standardized components such as switches, grippers, cables, joints, maybe even screws—different, preexisting PLs—and integrating them with newly created components as needed.

It is challenging to decide about using standardized or user-defined components according to the given

[1]Institute for Software Systems Engineering, Johannes Kepler University of Linz, Linz, Austria
[2]Institute of Mechatronic Design and Production, Johannes Kepler University of Linz, Linz, Austria
[3]I-NEW Unified Mobile Solutions AG, Mattersburg, Austria

Corresponding author:
Peter Hehenberger, Institute of Mechatronic Design and Production, Johannes Kepler University of Linz, 4040 Linz, Austria.
Email: peter.hehenberger@jku.at

requirements (e.g. cost and usability). On one hand, one of the most common motivators for promoting standardized components is the need to allow a large variety of products to be constructed from a much smaller set of different modules and components. The result is that any combination of modules and components, as well as the assembly equipment, can be standardized. Potential benefits of standardization and modularity are the increased product variety. The use of modules means that a great product variety can be achieved using different combinations of (even the same set of) modules. These modules may be manufactured in relatively large volumes, so the logistics of production can be organized in order to reduce manufacturing lead time. Disadvantages are the higher assembling and setting costs, often also the high required space. On the other hand, user-defined modules focus on the function and structure integration with a low number of parts and miniaturization. Our contribution aims to address this issue in industry using methods from product line engineering (PLE) and consistency management.

Research on PLE and variability modeling has developed a number of approaches for this context. Variability modeling techniques such as feature modeling or decision modeling have been proposed to define and manage commonalities and differences in product families (Czarnecki et al., 2012). Such techniques can be applied to mechatronic modules; however, it is quite hard to embed them in the mechatronic design process: for instance, it is challenging to integrate component variants derived from a mechatronic PL with user-defined components into a single architecture. In particular, preserving the consistency of multiple derived components with user-defined elements is demanding. In PLs, inconsistencies are typically avoided through "engineering for reuse," that is, carefully integrating newly developed components into the PL. However, in mechatronic design, a newly developed component is rarely standardized outright, and its standardization occurs through repeated use. Therefore, the addition of unprecedented and unanticipated user-defined elements can, thus, easily invalidate the consistency properties of a PL. Existing consistency checking approaches from software engineering are potentially useful in mechatronic design (e.g. Egyed, 2006) as they aim at maintaining consistency between scattered information about reused and new components handled by diverse design tools. Furthermore, in mechatronic design, variability needs to be managed at multiple levels of mechatronic components, modules, and systems, a case similar to MPLs, i.e., multi product lines (Brink et al., 2012; Holl et al., 2012) in which products originate from different PLs. Existing approaches allow structuring large-scale PLs (Holl et al., 2011, 2012) and support a step-wise (Czarnecki et al., 2004) and distributed (Holl et al., 2012) product derivation process.

The contribution of this article is a model-based approach supporting mechatronic designers that need to integrate standardized components and user-defined components in mechatronic architectures. The approach uses existing techniques to address variability and consistency at different levels of mechatronic design: (i) component providers precisely define the variability of mechatronic modules in models, which are then used by designers and component integrators to configure these mechatronic modules to their specific needs and requirements; and (ii) the approach adopts incremental consistency checking to discover mismatches between derived component variants and user-defined components to ease the integration of standardized and user-defined components in mechatronic systems. Our approach does not change the manner how mechatronic modules are created and maintained but allows the early discovery of inconsistencies between existing well-known and newly defined components. The key benefit of the approach is the integrated consideration of multidisciplinary design and the supply chain with a focus on the variants.

This article briefly discusses existing work on variability and consistency (section "Background"). Then, we illustrate the multi-stage mechatronic design process using an industrial robot example (section "Characteristics of the mechatronic design process"). We define the characteristics of mechatronic modules (section "Mechatronic modules") and describe our ongoing work on developing a model-based approach (section "Model-based approach"). We provide an application scenario (section "Application example: designing a gripper for an existing robot") and conclude this article with an outlook on future work (section "Conclusion").

Background

In mechatronic engineering, several methods exist to design products for variety. For instance, the method presented in Ko and Kuo (2010) considers external drivers (e.g. market and customer needs) during the analysis phase and internal drivers (e.g. interactions between physical components) during redesign and provides a visualization in the concept phase. Software engineering research has led to a wide range of approaches to model variability in the engineering process and to discover inconsistencies. Background research relevant for this article also exists in the emerging field of MPLs.

Variability modeling is essential to define and manage the commonalities and variability of similar products and to support engineers during product derivation (Czarnecki et al., 2012). Numerous variability modeling techniques have been proposed by the research community. Well-known examples are feature modeling and decision modeling. A feature is a “prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system” (Kang et al., 1990). Feature models represent the users’ perspective of a system in terms of commonalities and differences offered by different product variants. Decision models emphasize differences between product variants, that is, they comprise decisions to distinguish the different members of a PL and to guide the derivation of customized products (Czarnecki et al., 2012). An example is the Decision-Oriented Product Line Engineering for effective Reuse (DOPLER) approach (Dhungana et al., 2011) comprising a variability modeling tool and a configuration wizard supporting product derivation. DOPLER variability models contain decisions and assets. *Decisions* are used to describe differences between products and represent the choices that need to be made when deriving a customer-specific product from a PL. *Assets* in DOPLER models represent the reusable core artifacts of the PL. Assets are linked to decisions via inclusion conditions defining when a particular asset is present in a derived product.

Consistency checking is a state-of-the-art technique in model-based design approaches (Egyed, 2006). During the design phase, consistency checking reduces costs because conflicting design decisions can be detected and handled by arbitrary rules and constraints early on. Fundamental concepts in consistency management of mechatronic design models are discussed in Herzig et al. (2011) and Tomiyama et al. (2007). Herzig et al. (2011) provide a mathematical foundation to define consistency in a formal manner, and Tomiyama et al. (2007) discuss different types of complexity in multidisciplinary engineering processes. In addition to ensuring model correctness with respect to a metamodel, consistency checking can also be used to address domain-specific issues, for example, when guaranteeing compatible mounting interfaces between two components. It also plays an important role during variability modeling. For instance, Egyed’s (2006) incremental consistency checking approach was successfully applied to the DOPLER variability modeling tool (Vierhauser et al., 2010). Furthermore, many advances were made recently concerning the speed and the calculation of repair strategies.

MPLs have been described as “sets of several self-contained but still interdependent product lines that together represent a large-scale system” (Holl et al., 2012). MPLs rely on support for structuring PL models and on checking consistency among MPLs. The model fragment (Dhungana et al., 2010) approach defines guidelines on how to structure the modeling space while the compositional variability management (CVM) framework (Abele et al., 2010) allows defining a product hierarchy (Reiser and Weber, 2006) in an MPL. The context variability model approach (Hartmann and Trew, 2008) supports merging different feature models into a single feature model. Consistency checking in MPLs informs the users about potential problems during modeling and product derivation. It can be achieved by statically checking PL models to detect inconsistencies across PL boundaries or by dynamic checks during modeling and during distributed product derivation. An example is the product line configuration (PLiC) framework (Elsner et al., 2010), which automatically ensures consistency during product configuration by invoking builders on diverse configuration changes.

Characteristics of the mechatronic design process

Mechatronic design is a multi-level process (Hehenberger and Zeman, 2004) relying on both existing and developed mechatronic components. Requirements need to be satisfied at different levels ranging from high-level system requirements to detailed design decisions. When designing mechatronic systems, engineers are combining derived components (likely from different PLs) with custom-made components, a scenario that needs further investigation.

Articulated robots are widely used in industry for the handling of goods and tools due to their flexibility, reachable positions, and executable tasks. For instance, the basic robot shown in Figure 1 provides 6 degrees of freedom leading to manifold configurations and

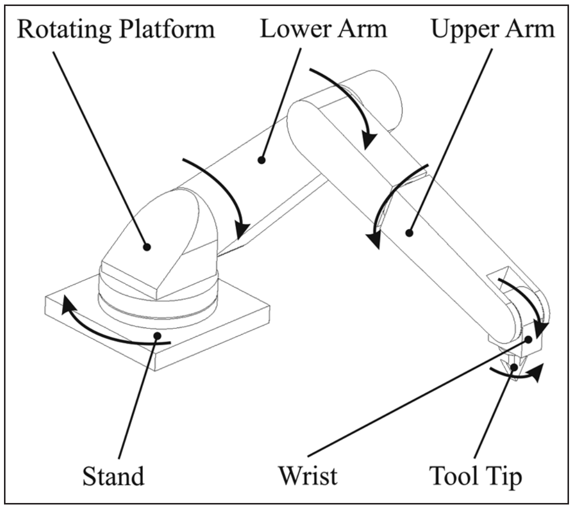


Figure 1. Basic articulated robot.

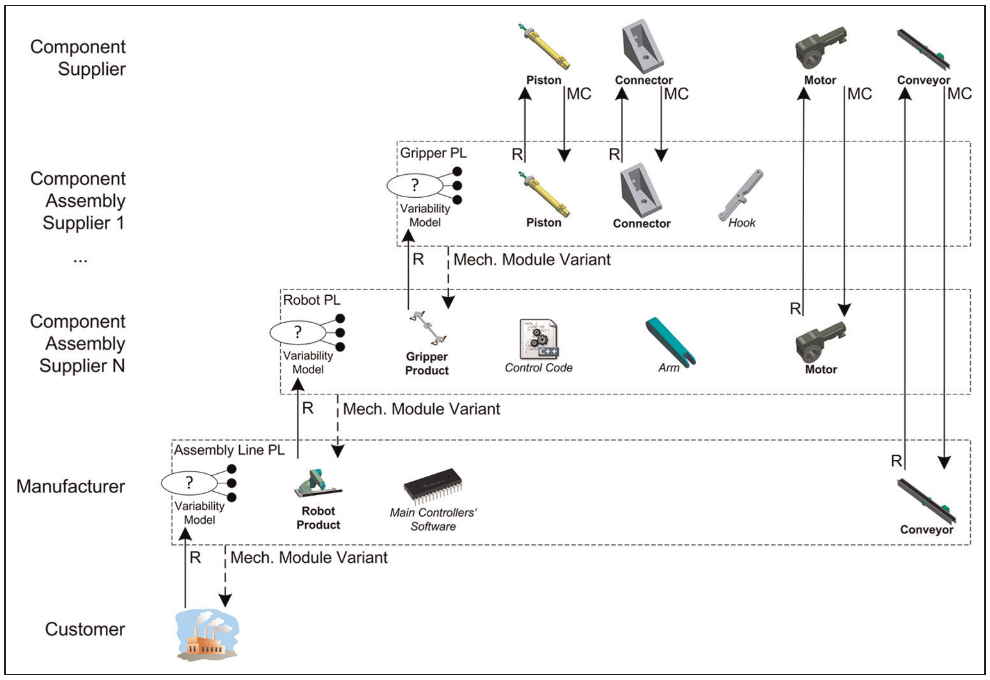


Figure 2. Mechatronic supply chain of a basic articulated robot.

applications in modern production lines, for example, gripping, welding, painting, and cutting. Using the example of such an articulated robot, Figure 2 illustrates a part of the supply chain of standardized components (label shown in boldface) and user-defined components (italics) together forming mechatronic components (framed with dotted lines) that are needed in the design process.

Based on high-level customer requirements, engineers select and configure standardized mechatronic components when designing such a robot. They develop user-defined components to adapt a system to a particular industrial application. Depending on the percentage of components manufactured in-house, the process occurs at multiple levels (Czarnecki et al., 2004) and involves components, component assemblies, mechatronic systems as well as their integration into the target environment. For the basic mechatronic components of an articulated robot such as pistons, connectors, or motors, standardized components can be used to reduce development costs. However, for the specific components required in industrial applications (e.g. a particular hook for the gripper for grabbing an object) such standardized components may not exist. These components still need to be developed for a specific purpose based on physical laws, component catalogs, design rules or reference projects.

In the example shown in Figure 2, a *Customer* plans to extend an existing factory with an assembly line comprising an articulated robot picking up photovoltaic panels and placing them on transport pallets. Requirements involve the robot's total cost, maximum payload, and positioning accuracy. The *Customer* communicates the requirements to the *Manufacturer* representing the next link in the supply chain. The manufacturer manages the *Assembly Line PL* comprising standardized and user-defined components. From the manufacturer's perspective, the *Robot Product* represents a standardized component that needs to be configured and manufactured by a *Component Assembly Supplier (N)* according to the requirements. The *Conveyor* represents another standardized component which is not configured but selected from the *Component Assembly Supplier's* product portfolio. In contrast, the integrated *Main Controller's Software* constitutes a user-defined component that needs to be adapted and developed for the assembly line. Inconsistencies can hinder the integration of standardized and user-defined mechatronic components. Therefore, the manufacturer has to ensure consistency at component and module interfaces. The *Robot PL* involves a *Gripper Product* representing a standardized component and both the *Control Code* and an *Arm* constituting user-defined components. The *Motor* component represents another standardized component that needs no additional configuration. Since the *Gripper Product* has to be configured and manufactured by another supplier, *Component Assembly Supplier (N)* passes requirements on to *Component Assembly Supplier 1*. *Component Assembly Supplier 1* manages a *Gripper PL* involving standardized components, for example, a *Piston* and a *Connector*, needing no additional configuration. Moreover, a user-defined component, for example, a *Hook*, needs to be developed adapting the gripper to the respective application.

Variability of standardized components plays an important role as illustrated in this example. Variability models can be seen as configuration interfaces of the mechatronic modules in the example. For instance, the variability of the robot needs to be defined and managed by Component Assembly Supplier (N) to allow its subsequent configuration by the manufacturer. The Robot PL again relies on configuring a Gripper product variant that is derived from the Gripper PL defined in a variability model by Component Assembly Supplier 1. Designers of robot applications further need to ensure *consistency* when integrating standardized and user-defined components to satisfy the customers' requirements and to avoid costly errors during operation. There are simple checks such as matching interfaces between mechanical or electrical components. However, designers also face more complex constraints, which are very hard to verify as they require access to details on individual parts and/or variant. For instance, the total weight of all attached parts may be limited by the carrier's maximum allowed load. Such constraints and rules need to be defined by domain experts to cover a wide range of design errors. Each standardized component is expected to be (internally) consistent. However, the consistency with user-defined components or other standardized components (e.g. derived from a PL) cannot be ensured a priori and, therefore, also needs to be checked. Nonetheless, even if two standardized components are internally consistent, combining them may be problematic. Consistency rules are able to detect some of these problems.

The described challenges also apply to additional examples of mechatronic systems. For instance, a *machine tool* is a powered mechanical device, typically used to fabricate metal parts by machining, which is the selective removal of metal. The term "machine tool" is usually reserved for tools that use a power source other than human power for the relative movement between tool and workpiece. Examples are press brakes, injection modeling machines, or milling machines. Machine tools may be controlled manually or automatically. Automatic machines may employ "hard" automation in the form of cams and fixed sequence mechanical or pneumatic systems or "soft" automation in the form of numerical control or other easily reprogrammable and hence flexible systems. A *milling machine* is another example of a power-driven machine used for complex shaping operations of parts. The basic form consists of a worktable for a workpiece and a rotating cutter which rotates concentrically to the spindle axis. These two subsystems have to perform motions relative to each other. The basic layout of a three-axis machine tool structure consists of an open serial chain of four mechatronic modules, the spindle, the X-axis, the Y-axis, the Z-axis, and the table with the clamping devices. Therefore, several possibilities for the arrangement of the modules exist. The components themselves in the several arrangements are identical. Therefore, the variability can be analyzed in a similar way.

An advantage of the presented approach is its extensibility to large-scale systems such as a steel production line. There are different productions steps (e.g. steelmaking, casting, and rolling), and their realizations have to be integrated as illustrated in Holl et al. (2012).

Mechatronic modules

Modularization is a design principle dividing a system or structure into interchangeable elements or modules. Modularity is viewed by Ulrich and Tung (1991) as the similarity between the physical and the functional architecture to minimize incidental interactions between physical components. In a modular system, the common elements and interfaces provide a platform allowing the development of different product variants that constitute a product family (Gershenson et al., 2004; Kang et al., 1990).

Mechatronic designers need to make decisions on the product architecture already during the conceptual design stage (Ariyo et al., 2008; Hehenberger and Zeman, 2004). This means that the decisions are typically based on a hierarchical functional model representing the overall mechatronic system at the highest level under consideration. Structuring systems hierarchically allows recognizing and describing internal interactions of mechatronic modules and thus facilitates the integration of these modules. This increases the visibility of relevant interactions, interdependencies, and interfaces in the design process.

Mechatronic modules frequently integrate components from several mechatronic disciplines (e.g. mechanical components and automatic control components), and their identification is crucial for the overall design: a mechatronic module at one level of integration may be decomposed into discipline-specific components but not into further mechatronic modules. A mechatronic module, therefore, designates the "smallest" indivisible mechatronic subsystem within the set of mechatronic subsystems at a particular stage of the mechatronic design process.

The meta-model depicted in Figure 3 illustrates the structure of mechatronic modules and their role in the design process of a *Mechatronic System* realizing a set of *System Requirements*. More specifically, a

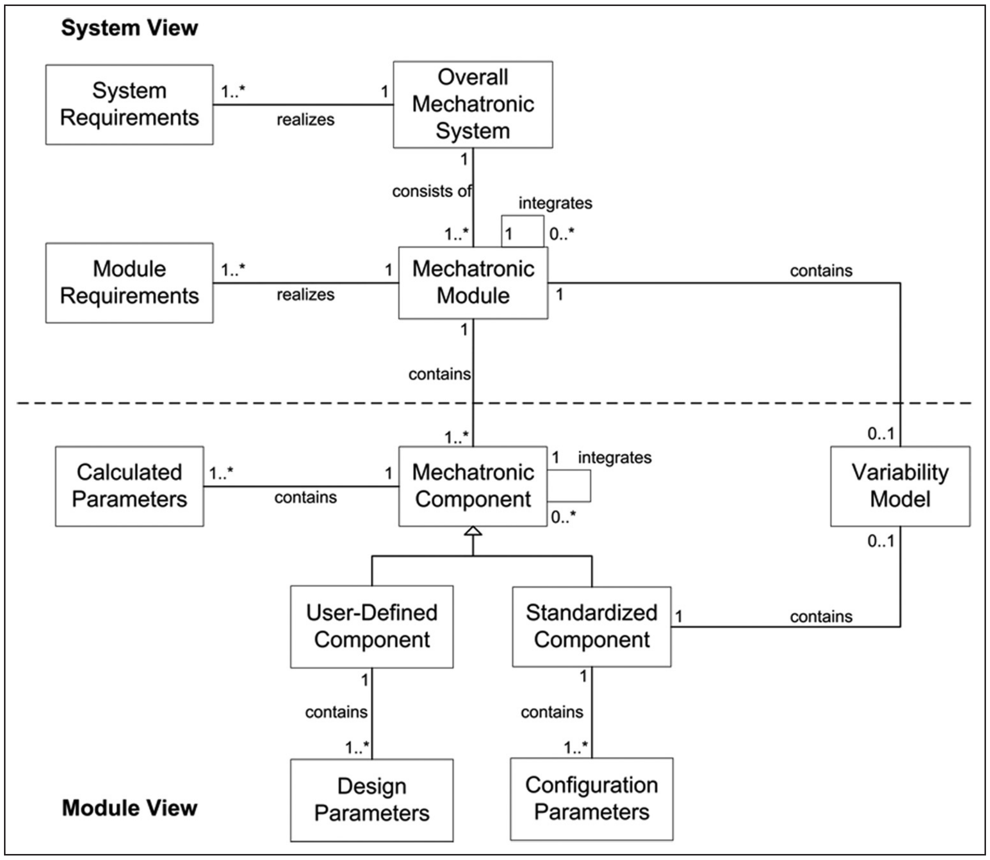


Figure 3. Basic structure of a mechatronic module.

Mechatronic Module is an element of a mechatronic system either provided by suppliers or developed earlier in-house. A mechatronic module can be of different granularities—it might be a simple nozzle but can also be an entire power train—depending on a specific context and domain. Mechatronic modules may integrate further mechatronic modules and contain mechatronic components relating to different *Mechatronic Disciplines* such as mechanical engineering, electrical engineering or electronics, and information technology. They are typically modeled along with the physical mechatronic system and address specific *Module Requirements*. They may contain a *Variability Model* defining its possible variants to ease the derivation of mechatronic module variants (Common Variability Language, 2013).

Mechatronic Components are either user-defined or standardized parts of mechatronic modules: *User-defined (parametric) Components* comprise at least one feature with continuous variables (e.g. position and size). For instance, they can be used to create three-dimensional (3D) models and manufacturing drawings. Thus, the *Design Parameters* specify the user-defined components and the range of allowed values. *Standardized Components* comprise discrete-parametric components, nonsimilar series, and identical components: discrete-parametric components evolve from parametric 3D models by discretizing the range of values of independent parameters. Their properties are similar geometry, the ability to be scaled by at least one or more independent parameters with discrete ranges of values in predefined increments, and the availability of a parametric 3D model and a manufacturing drawing that depends on the parameters. Nonsimilar series differ more significantly, for example, regarding size or geometry. They have a finite number of 3D models and manufacturing drawings that must not be changed but need to be replaced. Identical components remain unchanged in their shape and size, regardless of the application and the size of the surrounding parts. Such assemblies consist of a 3D base model and a manufacturing drawing. *Configuration Parameters* describe standardized components regarding the allowed configuration options, which can, for instance, be defined in a variability model.

Model-based approach

A variety of models are typically used in mechatronic design to address the needs of the involved disciplines

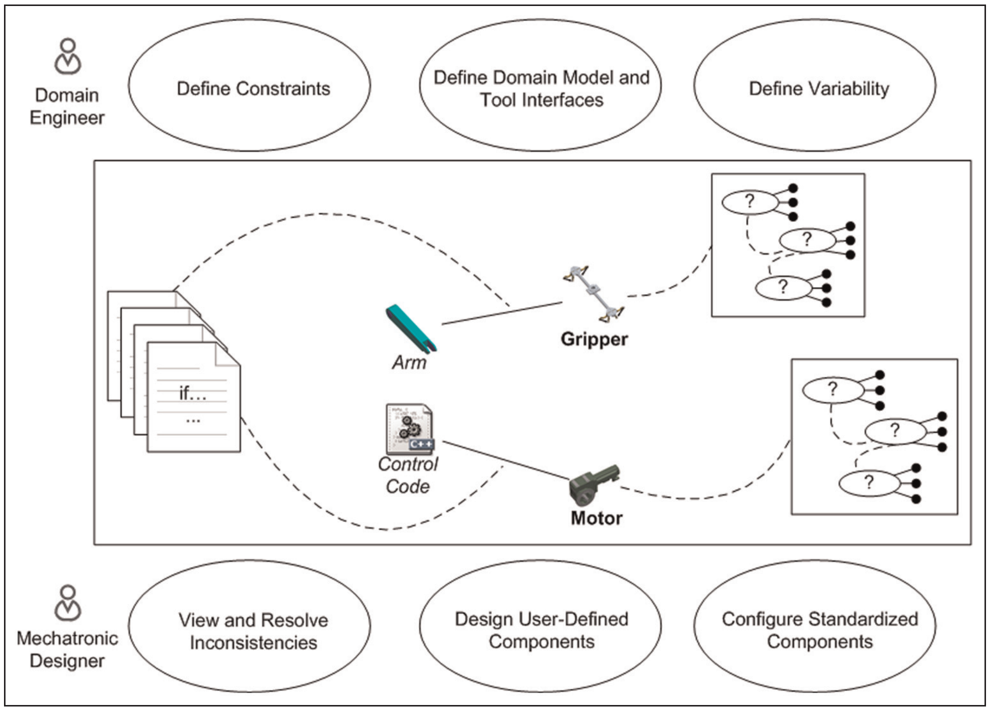

Domain Engineer
Define Constraints
Define Domain Model and Tool Interfaces
Define Variability
if…
…
Arm
Gripper
Control Code
Motor
Mechatronic Designer
View and Resolve Inconsistencies
Design User-Defined Components
Configure Standardized Components

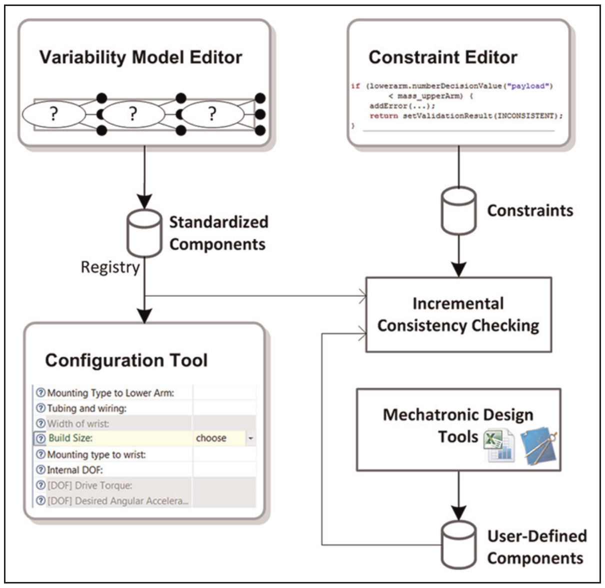
Figure 4. Roles and tasks of domain engineers and mechatronic designers.

(e.g. mechanics) and the different stages of the design process (e.g. conceptual models vs detailed models). Furthermore, models are used fulfilling a specific purpose in the engineering process (e.g. simulation of specific properties of the system).

The focus of our model-based approach is on variability modeling and custom modeling of user-defined components combined with consistency checking to detect composition errors. A high-level view showing its major elements is depicted in Figure 4. The approach distinguishes the views of domain engineers and mechatronic designers on constraints, components, and variability models.

The *domain engineer* prepares the platform for a specific environment. The domain engineer defines constraints for the domain that are automatically checked at design time to ensure that modules form a valid assembly. The domain engineer further creates a domain model defining key concepts and specifies how existing mechatronic tools can be integrated in the mechatronic design process. Integrated tool environments simplify this problem, but most engineering tools are not, or weakly, integrated. Today, engineering tools are "stand-alone" tools, and while they often provide separate means for detecting inconsistencies, they typically fail in detecting inconsistencies that involve artifacts scattered across multiple tools. A key problem here is maintaining consistency during changes where a change in one tool requires changes in other tools. Therefore, we developed tool interfaces and demonstrated it on several commonly used mechatronic tools, such as the Creo Elements Pro Engineer computer-aided design (CAD) tool, IBM Rational Software Architect, Eclipse, and MS Excel (which is frequently used to support calculations in the design process). The domain engineer also prepares variability models for standardized mechatronic components using a variability model editor. In our platform, we use decision-oriented variability models and the DOPLER tools (Dhungana et al., 2011).

The *mechatronic designer* engineers mechatronic modules based on standardized and user-defined components. The designer makes use of arbitrary mechatronic design tools for developing user-defined components that have already been linked into the platform. Furthermore, the designer uses a configuration tool to derive a suitable variant based on a standardized mechatronic component. In our platform, we use the DOPLER Configuration Wizard (Rabiser et al., 2012) for that purpose. Our platform uses a generic consistency checking framework originally described in Egyed (2006) and refined for different software engineering tasks (Reder and Egyed, 2010; Vierhauser et al., 2010, 2012). When engineering new systems, the incremental consistency checker monitors the constraints defined for the domain and presents constraint violations to the mechatronic designer.



```
if (lowerarm.numberDecisionValue("payload")
        < mass_upperArm) {
    addError(...);
    return setValidationResult(INCONSISTENT);
}
```

Mounting Type to Lower Arm:	
Tubing and wiring:	
Width of wrist:	
Build Size:	choose
Mounting type to wrist:	
Internal DOF:	
[DOF] Drive Torque:	
[DOF] Desired Angular Accelera...	

Figure 5. Tool architecture.

Figure 5 shows the tool architecture's main building blocks. A variability model editor is used to create standardized components which are managed in a registry. A configuration tool is used to refine standardized components. Mechatronic design tools are used to develop user-defined components. As was mentioned earlier, problems during composition range from incompatible interfaces to inconsistent behaviors. Such problems are detectable through constraint checking, and we make use of an incremental constraint checking mechanism (Egyed, 2006) that instantly recognizes such problems. Using a constraint editor, the composition requirements regarding the integration of standardized and user-defined components are defined and stored. An example of such a problem is the failure to connect up provided and required interfaces in components (e.g. if a component requires an input that is presently not provided, then there is a problem). Naturally, the ability to detect such conflicts is limited to the ability to express constraints. However, diverse languages are available today, and the limitations are mostly in the domain expert's ability to formalize a composition problem such that there are no false problems detected.

Variability model editor

The domain engineer uses a variability model editor to define and manage PL models for a specific PL. The editor supports engineers in creating meta-models defining asset types with attributes and relationships. Domain-specific variability models can be created based on a specific meta-model.

Constraint editor

The domain engineer uses a constraint editor to specify constraints. Constraints can be specific to a certain model element but usually are specific to all instances (model elements) of a certain concept. Currently, constraints can be specified in the Object Constraint Language (OCL) or Java.

Registry

The standardized components are realized as product line bundles (PLiBs) (Holl et al., 2011). Similar to the concept of configurable units in the common variability language (CVL; Common Variability Language, 2013), PLiBs represent self-contained components encapsulating a variability model and additional artifacts needed to derive component variants. PLiBs comprise a domain-specific meta-model for defining variability models and generators for deriving component variants. PLiBs also provide an interface to access their variability model (Holl et al., 2011). The PLiB registry serves as a repository for managing available PLiBs.

Configuration tool

Using a configuration tool, the mechatronic designer makes decisions to select features for deriving standardized components (Holl et al., 2012).

Incremental consistency checking

We use the ModelAnalyzer approach as an incremental consistency checking mechanism which builds up a scope for each constraint validation (Egyed, 2006). The scope represents a set of model elements that was accessed during the validation. A constraint only needs to be validated in case a model element belonging to the scope changes—this avoids unnecessary validations. Hence, the checking is incremental and fast. In addition, since constraint validations are handled individually, new constraints can easily be added at any time. The constraints we currently support are mostly structural in nature (e.g. compatible component interfaces, data replications among different tools and incompleteness). Behavioral constraints are typically harder to express and are, thus, not often possible. The detected inconsistencies are, thus, not exhaustive, but they provide quick feedback on many common problems.

Mechatronic design tools

The domain engineer connects mechatronic design tools used in the mechatronic design process with the model

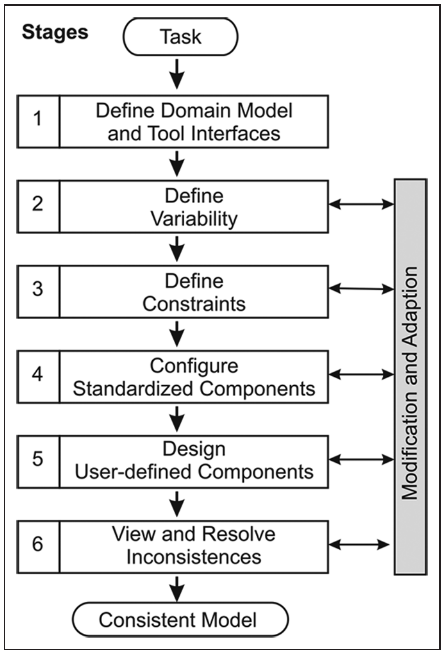

Stages
Task
1 Define Domain Model and Tool Interfaces
2 Define Variability
3 Define Constraints
4 Configure Standardized Components
5 Design User-defined Components
6 View and Resolve Inconsistences
Modification and Adaption
Consistent Model

Figure 6. Typical workflow.

analyzer environment. Figure 6 shows a typical workflow of applying the proposed model-based approach implemented in the tool architecture.

Application example: designing a gripper for an existing robot

We illustrate the presented approach from a robot manufacturer's perspective using a realistic mechatronic scenario. The robot manufacturer intends to introduce a PLE approach to reduce the complexity in designing new robot solutions.

Developing a robot PL

An experienced domain engineer familiar with the company's products and workflows coordinates framework development including extending the tools' functionalities to export data fragments, defining the meta-model objects, setting up constraints between these objects and refining information on variability.

Define domain model and tool interfaces. Articulated robots consist of a robot base, a robot tool, and further robot links all connected by mechanical connectors resulting in the domain model shown in Figure 7. Furthermore, we closely examine a more specific robot type comprising an upper and a lower arms and a wrist. A flange is used as mechanical connector. Robot parts are primarily characterized by their weight and their maximum payload. For some specialized robot parts,

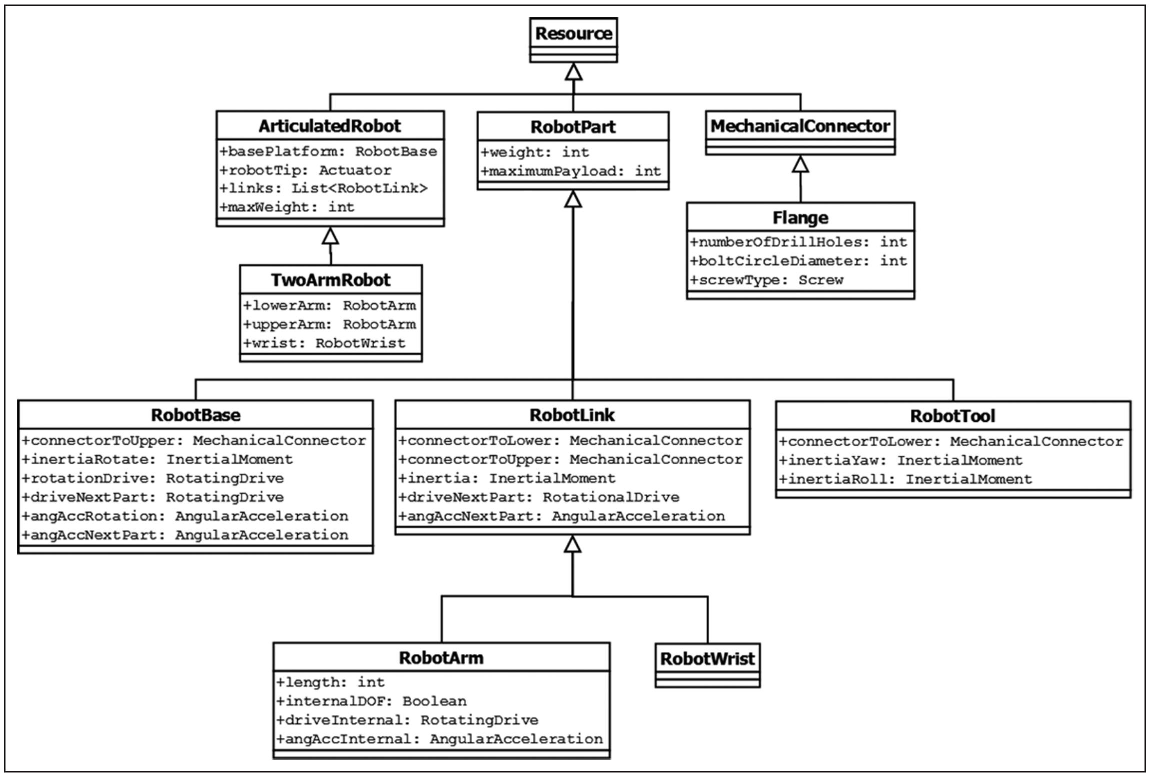
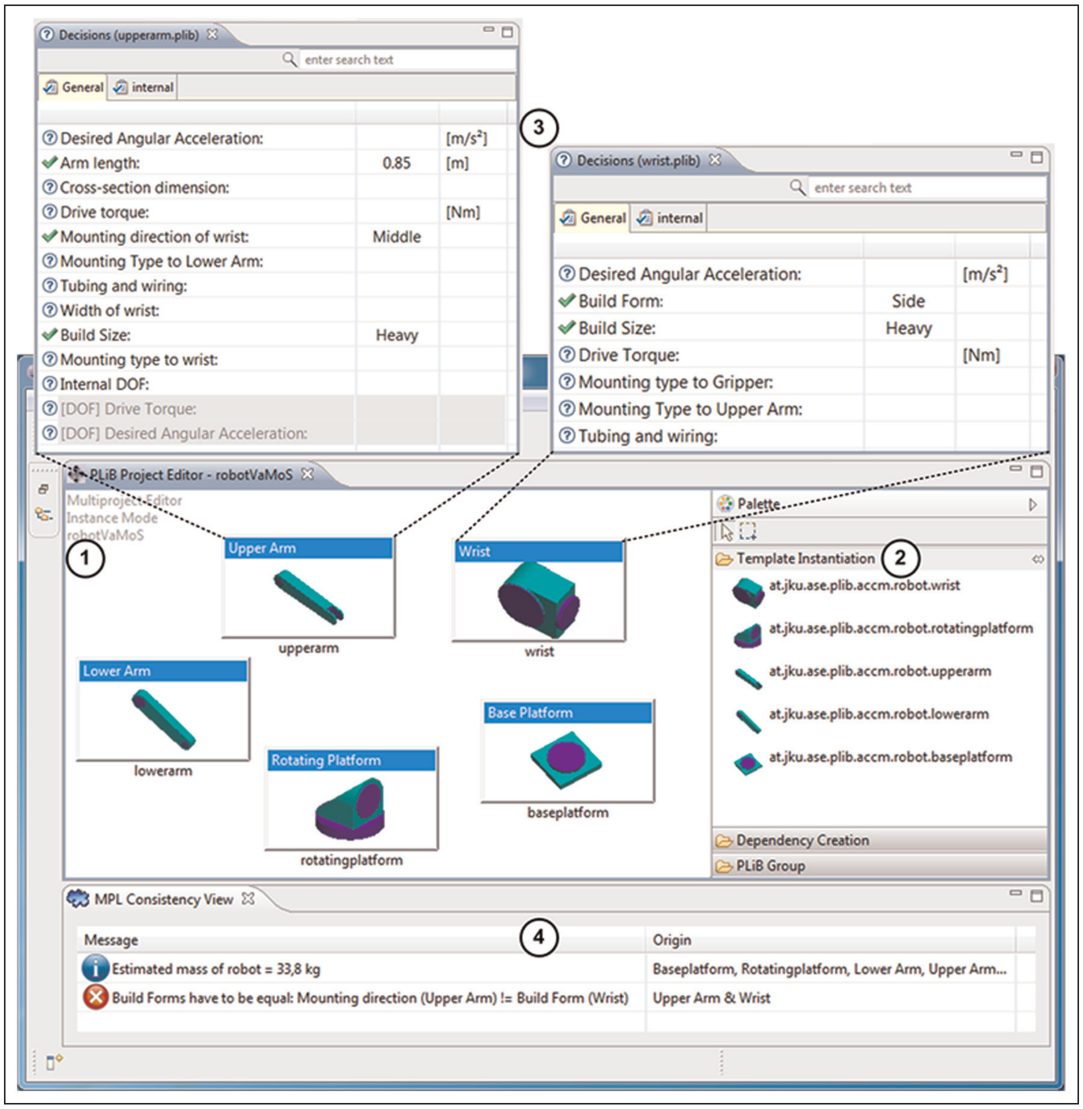

Resource
ArticulatedRobot
+basePlatform: RobotBase
+robotTip: Actuator
+links: List<RobotLink>
+maxWeight: int
RobotPart
+weight: int
+maximumPayload: int
MechanicalConnector
Flange
+numberOfDrillHoles: int
+boltCircleDiameter: int
+screwType: Screw
TwoArmRobot
+lowerArm: RobotArm
+upperArm: RobotArm
+wrist: RobotWrist
RobotBase
+connectorToUpper: MechanicalConnector
+inertiaRotate: InertialMoment
+rotationDrive: RotatingDrive
+driveNextPart: RotatingDrive
+angAccRotation: AngularAcceleration
+angAccNextPart: AngularAcceleration
RobotLink
+connectorToLower: MechanicalConnector
+connectorToUpper: MechanicalConnector
+inertia: InertialMoment
+driveNextPart: RotationalDrive
+angAccNextPart: AngularAcceleration
RobotTool
+connectorToLower: MechanicalConnector
+inertiaYaw: InertialMoment
+inertiaRoll: InertialMoment
RobotArm
+length: int
+internalDOF: Boolean
+driveInternal: RotatingDrive
+angAccInternal: AngularAcceleration
RobotWrist

Figure 7. Domain model.



Decisions (upperarm.plib)

Desired Angular Acceleration:		[m/s^2]
✓ Arm length:	0.85	[m]
Cross-section dimension:		
Drive torque:		[Nm]
✓ Mounting direction of wrist:	Middle	
Mounting Type to Lower Arm:		
Tubing and wiring:		
Width of wrist:		
✓ Build Size:	Heavy	
Mounting type to wrist:		
Internal DOF:		
[DOF] Drive Torque:		
[DOF] Desired Angular Acceleration:		

Decisions (wrist.plib)

Desired Angular Acceleration:		[m/s^2]
✓ Build Form:	Side	
✓ Build Size:	Heavy	
Drive Torque:		[Nm]
Mounting type to Gripper:		
Mounting Type to Upper Arm:		
Tubing and wiring:		

Message	Origin
Estimated mass of robot = 33,8 kg	Baseplatform, Rotatingplatform, Lower Arm, Upper Arm...
Build Forms have to be equal: Mounting direction (Upper Arm) != Build Form (Wrist)	Upper Arm & Wrist

Figure 8. Prototype tool environment.

certain inertial moments influence the mechanical behavior. Therefore, rotational drives with a certain torque are attached to realize target angular accelerations of connected parts. When more flexibility in position and orientation is required, the robot arm can be extended with an additional, internal rotational degree of freedom.

Define variability. The design of a robot depends on the projected tasks and the properties of the specific robot tool: the weight and inertias for rotating and tilting and the required mounting interface to the robot. The second set of parameters stems from the customer's requirements on cycle times for certain moves. The selection for each type of robot part (platform, arm, and wrist) is divided into three purpose classes: light, medium, and heavy duty. Weight and inertia, as well as maximum loads, for the resulting parts are measured and stored in a table. Arms with enabled internal degrees of freedom require an additional selection of a drive. Possible drives are selected from a catalog of the most common solutions used in the past. The parameters for connections depend on the connection type, for example, screws for flange connections can be reduced to standardized screw classes, whereas the diameters in a tight fit connection need to be defined up to a precision of micrometers.

Define constraints. The following constraints are defined for the robot manufacturing domain: (i) each part's payload must be greater than the total weight of all attached parts (payload); (ii) at each joint, the connection type and parameters have to fit for both sides (connection type); (iii) the connection-dependent maximum connection torque and the drive's torque must be greater or equal to the desired angular acceleration multiplied by the sum of attached parts' inertias and the Steiner shares taking account of the distances between joint and the parts' centers of mass (maximum torque); and (iv) the wrist has to be paired with a lower arm comprising the same mounting direction (build form).

Mounting a new gripper

A customer operating an articulated robot based on the robot PL needs to exchange the gripper to manipulate lightweight profiles with a gripper capable of transporting solar panels. This requires the mounting of a different, heavier gripper.

Configure standardized components. The mechatronic designer loads the stored configuration data and checks relevant modifications with the customer.

Design user-defined component. In this case, there is no need to design user-defined components, as the gripper has been purchased by the customer from a different supplier. Furthermore, to add the gripper to the solution, it is only necessary to create a new robot tool with weight, inertia, geometric dimensions, and mounting parameters provided by the customer.

View and resolve inconsistencies. Figure 7 shows a screenshot of our prototype modeling environment, using several standardized components from the robot example. When selecting a specific component, the configuration tool (cf. Figure 5) supports designers in setting the configuration values (e.g. mounting direction or build form; Figure 8: 1 and 3) based on the variability models available for this component. The system checks the consistency rules (using incremental consistency checking mechanisms discussed in section "Model-based approach") defined for the domain and displays violations in a separate view (cf. Figure 8: 4). Designers can add and thereby instantiate further components to the workspace from the palette at any time (cf. Figure 8: 2). The tool environment informs the mechatronic designer that the build forms of the upper arm and the wrist have to be equal (cf. Figure 8: 4). Furthermore, the loads imposed by the new gripper exceed the maximum payload for the medium duty lower arm. Hence, the mechatronic designer replaces the lower arm with a heavy duty arm. These changes influence also the robot's control software, as the masses and inertias change, and the positioning ranges in the path planning software, due to the changed gripper dimensions.

Discussion

The example shows that managing consistency in mechatronic design is challenging as standardized and user-defined mechatronic components may not be treated in isolation but need to be integrated in mechatronic modules. In addition to the need for maintaining consistency within mechatronic components, there is the need for maintaining consistency across component boundaries.

Conceptually speaking, maintaining consistency within one component has a lot in common with managing consistency across diverse components. However, while preserving consistency within the boundaries of a standardized component (variability model) is state-of-the-art (Egyed, 2006; Reder and Egyed, 2010, 2012; Vierhauser et al., 2010, 2012) and well researched, maintaining consistency between multiple standardized components is more demanding (Holl et al., 2012). This challenge is manageable as long as the variability of different PLs is documented within one tool or variability models use common data structures with clearly defined interfaces for configuration data exchange (e.g. as proposed in the upcoming CVL; Common Variability Language, 2013). Frequently, user-defined components are at an early development stage from a PLE view as the necessary information is not as structured as, for example, it is the case for standardized components. For instance, it is common that the required information is scattered over several mechatronic design tools. As a consequence, in order to maintain consistency between standardized components and user-defined components, we need consistency checking mechanisms that can handle heterogeneous tools lacking common models, standardized data structures, and compatible interfaces. The purpose and semantic meaning of cross-component constraints is quite diverse. There are constraints that ensure equality, for instance, guaranteeing the same type of mounting interfaces between standardized components (e.g. gripper and wrist) and user-defined components (e.g. arm). Furthermore, there are constraints concerning mass spanning across all of the robot's components including the user-defined components. Each component has to be able to support the mass of all attached components, meaning, of course, the gripper has to support its payload; the wrist has to support the gripper's mass plus the gripper's payload; and finally, the stand has to support all components' masses plus the gripper's payload. These constraint types can be checked quite easily once the information is available. However, there are more challenging types of constraints. For instance, it is necessary to check whether it is geometrically possible for a robot to transport the desired object (e.g. photovoltaic cell) from the source to the target location. Therefore, simulations of integrated views of several components, including standardized and user-defined components, need to be supported.

Conclusion

Standardization in mechatronic design aims at reducing costs while retaining quality. Engineers need to integrate both standardized and user-developed mechatronic components at different levels of integration. Since inconsistencies between individual mechatronic components are likely to be introduced and costly to discover and resolve later on, consistency checking is required throughout the mechatronic design process. However, managing consistency in mechatronic systems is challenging as it involves cross-component constraints between standardized and user-defined components. We have been developing a model-based approach that addresses both variability and consistency in the mechatronic design process. We are currently investigating the usefulness of the approach based on different examples of mechatronic systems.

Acknowledgements

The authors thank all the involved partners for their support. Additionally, this work was conducted in cooperation with Primetals Technologies and Siemens CT and was supported by the Christian Doppler Forschungsgesellschaft.

Funding

This work was supported by the Linz Center of Mechatronics (LCM) in the framework of the Austrian COMET-K2 Programme ACCM.

References

Abele A, Johansson R, Lönn H, et al. (2010) The CVM framework—a prototype tool for compositional variability management. In: *Proceedings of the 4th international workshop on variability modelling of software-intensive systems*, Universität Duisburg-Essen Linz, 27–29 January, pp. 101–105.

Alvarez Cabrera AA, Woestenenk K and Tomiyama T (2011) An architecture model to support cooperative design for mechatronic products: a control design case. *Mechatronics* 21(3): 534–547.

Ariyo OO, Eckert CM and Clarkson PJ (2008) Hierarchical decompositions for complex product representation. In: *International design conference*, Dubrovnik, 19–22 May, pp. 737–744.

Bishop RH (2007) *Mechatronic Fundamentals and Modeling* (The mechatronics handbook). New York: CRC Press Inc.

Brink C, Peters M and Sachweh S (2012) Configuration of mechatronic multi product lines. In: *Proceedings of the 3rd international workshop on variability & composition*, Potsdam, 25–30 March, pp. 7–12: ACM New York, NY, ISBN: 978-1-4503-1101-4.

Common Variability Language (2013) Common variability language. Available at: http://www.omgwiki.org/variability/doku.php (accessed 8 November 2013).

Czarnecki K, Grünbacher P, Rabiser R, et al. (2012) Cool features and tough decisions: a comparison of variability modeling approaches. In: *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*, Leipzig, 25–27 January, pp. 173–182. ACM New York, NY, ISBN: 978-1-4503-1058-1.

Czarnecki K, Helsen S and Eisenecker U (2004) Staged configuration using feature models. In: *Proceedings of the 3rd international conference on software product lines*, pp. 266–283. Boston, MA, USA; Date and Month of Conference: August 30-September 2; Pubisher: Springer Berlin Heidelberg

Dhungana D, Grünbacher P and Rabiser R (2011) The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering* 18(1): 77–114.

Dhungana D, Grünbacher P, Rabiser R, et al. (2010) Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software* 83(7): 1108–1122.

Egyed A (2006) Instant consistency checking for the UML. In: *Proceedings of the 28th international conference on software engineering*, Shanghai, China, 20–28 May, pp. 381–390. ACM New York, NY, ISBN: 1-59593-375-1.

Elsner C, Ulbrich P, Lohmann D, et al. (2010) Consistent product line configuration across file type and product line boundaries. In: *Proceedings of the 14th international software product line conference*, Jeju Island, South Korea, 13–17 September. pp. 181–195. Springer Berlin Heidelberg.

Gershenson JK, Prasad GJ and Zhang Y (2004) Product modularity: measures and design methods. *Journal of Engineering Design* 15(1): 33–51.

Hartmann H and Trew T (2008) Using feature diagrams with context variability to model multiple product lines for software supply chains. In: *Proceedings of the 12th international software product line conference*, Limerick, 8–12 September, pp. 12–21. IEEE.

Hehenberger P and Zeman K (2004) Hierarchical structuring of mechatronic design models. In: *Proceedings of the 3rd IFAC symposium on mechatronic systems* Sydney, Australia, 6–8 September, 2004. Elsevier IFAC.

Herzig S, Qamar A, Reichwein A, et al. (2011) A conceptual framework for consistency management in model-based systems engineering. In: *Proceedings of ASME international design engineering technical conferences & computers and information in engineering conference*, Washington, DC, 28–31 August.

Holl G, Grünbacher P and Rabiser R (2012) A systematic review and an expert survey on capabilities supporting multi product lines. *Information and Software Technology* 54: 828–852.

Holl G, Grünbacher P, Elsner C, et al. (2012) Supporting awareness during collaborative and distributed configuration of multi product lines. In: *Proceedings of the 19th Asia-Pacific software engineering conference*, Hong Kong, 4–7 December, pp. 137–147. New York: IEEE.

Holl G, Vierhauser M, Heider W, et al. (2011) Product line bundles for tool support in multi product lines. In: *Proceedings of the 5th workshop on variability modeling of software-intensive systems*, Namur, Belgium, 27–29 January. pp. 21–27. ACM.

Kang K, Cohen S, Hess J, et al. (1990) *Feature-Oriented Domain Analysis (FODA) feasibility study*. Technical Report, CMU/SEI-90TR-21. Available at: http://www.sei.cmu.edu/reports/90tr021.pdf

Ko YT and Kuo PH (2010) Modeling concurrent design method for product variety. *Concurrent Engineering* 18(3): 207–217.

Kusiak A (2002) Integrated product and process design: a modularity perspective. *Journal of Engineering Design* 13(3): 223–231.

Medvidovic N and Taylor RN (2000) A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering* 26(1): 70–93.

Pahl G, Beitz W, Wallace K, et al. (2007) *Engineering Design: A Systematic Approach*. London: Springer Science & Business Media.

Rabiser R, Grünbacher P and Lehofer M (2012) A qualitative study on user guidance capabilities in product configuration tools. In: *Proceedings of the 27nd IEEE/ACM international conference on automated software engineering*, Essen, 3–7 September 2012, pp. 110–119. IEEE.

Reder A and Egyed A (2010) Model/analyzer: a tool for detecting, visualizing and fixing design errors in UML. In: *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*, Antwerp, 20–24 September, pp. 347–348. New York: ACM.

Reder A and Egyed A (2012) Incremental consistency checking for complex design rules and larger model changes. In: *Proceedings of the 15th international conference on model driven engineering languages and systems*, Innsbruck, Austria, September 30–October 5. pp. 202–218. Springer Berlin Heidelberg.

Reiser M-O and Weber M (2006) Managing highly complex product families with multi-level feature trees. In: *Proceedings of the 14th IEEE international requirements engineering conference*, Minneapolis, MN, 11–15 September, pp. 146–155. New York: IEEE.

Schäfer W and Wehrheim H (2007) The challenges of building advanced mechatronic systems. In: *2007 future of software engineering*, Minneapolis, MN, 23–25 May, pp. 72–84. New York: IEEE.

Tomiyama T, Amelio VD, Urbanic J, et al. (2007) Complexity of multi-disciplinary design. *CIRP Annals: Manufacturing Technology* 56(1): 185–188.

Ulrich K and Tung K (1991) Fundamental of product modularity. In: *ASME winter annual meeting symposium on issues in design/manufacturing integration*, Atlanta, GA, November, pp. 73–79. ASME.

Vierhauser M, Grünbacher P, Egyed A, et al. (2010) Flexible and scalable consistency checking on product line variability models. In: *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*, Antwerp, 20–24 September pp. 63–72. ACM.

Vierhauser M, Grünbacher P, Heider W, et al. (2012) Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines. In: *Proceedings of the 15th international ACM/IEEE conference on model driven engineering languages & systems*, Innsbruck, Austria, September 30–October 5 pp. 531–545. Springer Berlin Heidelberg.

Author biographies

Daniela Lettner is a member of the scientific staff in the Christian Doppler Laboratory on Monitoring and Evolution of Very-Large-Scale Software Systems at Johannes Kepler University of Linz (JKU), Austria. She received her master's degree of software engineering from JKU in 2012. She is interested in variability and evolution of large-scale software ecosystems.

Peter Hehenberger is senior researcher at the Institute of Mechatronic Design and Production of the JKU. He received is diploma degree in mechatronics and his doctorate degree from JKU in 2000 and 2004, respectively. His research interest is in model-based mechatronic design and has published numerous papers in this area. In addition, he authored and co-authored three books related to this field.

Alexander Nöhrer was a senior researcher at the Institute for Software Systems Engineering (ISSE) at the JKU. He received his PhD in computer science from JKU in 2012 and has published extensively on decision making and correct reasoning in product line engineering. He now works as a software architect at I-New Unified Mobile Solutions AG, Austria.

Klaus Anzengruber is working on his PhD degree at the Institute of Mechatronic Design and Production of the JKU. He received his MSc in mechatronics from JKU in 2011. He is interested in mechatronic design process models with the focus on assumptions.

Paul Grünbacher is an associate professor at JKU and a research associate at the Center for Software Engineering (University of Southern California, Los Angeles). He received his MSc in applied computer science from the University of Linz (1992) and a PhD in computer science and economics from the University of Linz (1995). His research interests include requirements engineering, product lines, and software monitoring. He is a member of ACM, ACM SIGSOFT, IEEE, and the IEEE Computer Society.

Michael Mayrhofer is working on his MSc degree at the Institute of Mechatronic Design and Production of the JKU. He received his BSc from JKU in 2013 in mechatronics. He is interested in system modeling of mechatronic systems.

Alexander Egyed is a full professor at the JKU and head of the ISSE. He received his doctorate degree from the University of Southern California, United States, and previously worked for Teknowledge Corporation, United States (2000–2007) and the University College London, United Kingdom (2007–2008). He was recognized as a top 1% scholar in software engineering in the Communications of the ACM, Springer Scientometrics, and Microsoft Academic Search. He was also named an IBM Research Faculty Fellow and received a Recognition of Service Award from the ACM, Best Paper Awards from COMPSAC and WICSA, and an Outstanding Achievement Award from the USC.