Formal computer-aided product family architecture design for mass customization

Bonev, Martin; Hvam, Lars; Clarkson, John; Maier, Anja

Published in:
Computers in Industry

Link to article, DOI:
10.1016/j.compind.2015.07.006

Publication date:
2015

Document Version
Peer reviewed version

Citation (APA):
Bonev, M., Hvam, L., Clarkson, J., & Maier, A. (2015). Formal computer-aided product family architecture design for mass customization. *Computers in Industry*, *74*, 58–70. https://doi.org/10.1016/j.compind.2015.07.006

Formal computer-aided product family architecture design for mass customization

Authors: Bonev, M., Hvam, L., Clarkson, J., Maier, A.

Keywords: Product family architecture, Design synthesis, Mass customization, Structural analysis, Generic modelling

Abstract: With product customization companies aim at creating higher customer value and stronger economic benefits. The profitability of the offered variety relies on the quality of the developed product family architectures and their consistent implementation in configuration systems. Yet existing methods are informal, providing limited support for domain experts to communicate, synthesize and document architectures effectively. In single product design explicit visual models such as design structure matrices and node-link diagrams have been used in combination with structural analysis methods to overcome the limitation of the informal approach. Drawing on thereto established best practises, this paper evaluates and extends the relevant methods and modelling techniques, to create a consistent and formal approach for the design and customization of entire product families. To validate it's applicability, the approach is tested on a case study at a manufacturing company offering bespoke industrial applications. A generic modelling method termed the *integrated design model* (IDM) is developed and complemented with a computational structural analysis method, to assist domain experts in their daily work. When combined with a configuration system, the presented IDM tool automates the documentation and formalizes the synthesis of architectures, thereby making any decision about a preferred solution explicit and transparent.

1 Introduction

A growing demand towards higher product variety has been reported in many industries (Funke and Ruhwedel, 2001; Klenow and Bils, 2001). Acting upon this trend, companies aim at obtaining higher customer value and stronger economic benefits through rapidly responding to individual needs for customization (Trentin et al., 2011). Nonetheless, high and diverse product mixes are not always beneficial but often challenge manufacturers with a related increase in operational complexity and decrease in efficiency in sales, design, production and distribution (Åhlström and Westbrook, 1999). Platforms and modules built into product family architectures have been reported to facilitate this trade-off (AlGeddawy and ElMaraghy, 2013). In this context, architectures are defined as an abstract structural representation of the functional units and the corresponding physical components of engineering artefacts (Ulrich, 1995). Their development is complex and long lasting and their performance can have wide-ranging effects on the success of manufacturers (Yassine and Wissmann, 2007). Designing architectures suitable for customization raises additional difficulties to organizations, since the right product composition and part compatibility needs to be ensured. With product configuration systems or configurators, manufacturers are able to handle these demanding requirements for information processing, storage and retrieval of feasible variant combinations (Trentin et al., 2011).

Configurators are software-based expert systems that capture the generic architecture of product families in a computer model, through which users are supported in creating feasible product solutions with a minimum number of choices (Hvam et al., 2011). If combined with well-designed product family architectures, companies can utilize configurators to mass customize their offerings, i.e. to automate operational activities related to product customization and to increase their efficiency to a level which is close to mass production (Jiao et al., 1998). However, architectures per se are qualitative and current methods supporting their design and documentation are informal and limited (Li et al., 2011; Wyatt et al., 2011). Hence, it can be difficult to identify ‘good’ architectures during product design and to sustain their subsequent implementation in a configuration system. At the same time, configuration software vendors are of no help in this respect, as they are typically not interested in providing a transparent and easy way to create and communicate the architectures, but rather emphasize consulting services around their modelling and maintenance (Forza and Salvador, 2008). Consequently, with the development progress of product families, software experts have problems in keeping an overview of what had been implemented in the computer model and verifying the obtained architectures with domain experts, making it one of the main reasons why designing and mass customizing products is still difficult to achieve (Haug et al., 2012).

This paper presents a formal computer-assisted approach that addresses the requirements for the design of product family architecture as identified by academia and industry. Section 2 first discusses existing approaches from both engineering and software domains, to define a consistent architecture design framework. Next, the challenges with conventional informal approaches are discussed and requirements for a formal support method are developed. Section 3 elaborates and further extends existing modelling techniques and relevant formal architecture support measures. In Section 4 a formal approach is presented and complemented with a case study of a major plant and machinery provider of highly customizable products, to develop a concrete example on a real world problem. The introduced approach combines the capabilities of the utilized configurator with automatically generated grammar graphs representing the implemented architectures. The graphs are modelled with an integrated design model (IDM), using the suggested extended modelling techniques for generic structures. A developed IDM tool is further employed to assist domain experts in synthesising feasible architectures and to evaluate their structural characteristics computationally through a series of metrics, potentially leading to better solutions. Finally, Section 6 concludes with an assessment of the proposed approach.

2 Relevant literature

To evaluate the limitations of existing approaches to architecture design for mass customization, a literature review on relevant frameworks, methods and modelling techniques is performed and requirements for a formal method are developed.

2.1 Approaches to architecture design in engineering and software development

The design of architectures and their subsequent implementation in configurators involves domain experts from different departments and often physically disconnected teams. Several researchers have acknowledged the related organizational challenges and have proposed methods on how to arrange corresponding activities in a more systematic manner (Ardissono et al., 2003; Forza et al., 1994; Hvam et al., 2004). In engineering domains Pahl et al. (2006) address architecture design on several stages, from formulating customer needs to the construction of embodiment and detailed design (Pahl and Beitz, 1996). Corresponding to these different phases of development, Jiao et al. (1999) argue for an architecture modelling framework which in addition considers several views of a product (Jiao and Tseng, 1999). At the same time frameworks dealing with architecture design for expert systems typically fall within the area of software systems and base their methods on the life-cycle of object-oriented software development as introduced by Booch (1986). Booch’s object-oriented procedure was originally developed to handle the complexity of large software projects by breaking down the development work into phases of object-oriented analysis, design, implementation and maintenance (Booch, 1986). The transformation from a real world design into a computer model is organized in several steps, where the observed reality is gradually abstracted and formalized (Duffy and Andreasen, 1995). To enable the representation of a large number of physical artefacts with components and variant combinations, related frameworks commonly build upon methods for modelling software architectures using the unified modelling language (UML) (Felfernig et al., 2000).

Although the UML standard proved to be particular useful for defining entire product families, its application within engineering design remains limited. In consequence, synergies on coinciding aspects of architecture design are seldom achieved. For example, the challenge of modelling different architecture views has been repeatedly addressed within the two domains and has resulted in comparable outcomes (Brière-Côté et al., 2010; Haug et al., 2010; Jiao and Tseng, 1999). Moreover, advancements within engineering design are seldom adopted to software design and vice versa, in particular with regard to the formal computational management of structural properties in complex architectures (Lindemann et al., 2009). Secondly, the development of a product family architecture for expert systems is often organized within IT and product data management departments. The process is regarded as a liberally new modelling approach which is detached from any preceding design activities of the product development phase (Speel et al., 2001). This means that in praxis the design of architectures is not coordinated across the organization, leading to computer models which are very likely to differ from the original design intent of the engineers (Haug et al., 2012). Especially for more complex products, this lack of consistency increases the risk of providing undesired product variety to the market (Martin and Ishii, 2002). As a benchmark report with more than 300 manufacturers of custom tailored products reveals, the top performing companies with engineering intensive portfolios try to overcome this coordination burden by better involving development engineers in the architecture design process for their configuration systems (Aberdeen, 2008). This suggests that a more integrated approach to mass customization is needed, which considers equally both the architecture design process and the subsequent implementation into configuration systems.

2.2 Challenges with conventional informal architecture design methods

Fig. 1 illustrates how a consistent framework of the architecture design process and its transformation into a computer model may be organized. The model is based on the Wyatt et al. (2011) generic scheme for architecture design in engineering, and combines this with the discussed transformation into a computer model. The focus of this paper is indicated by the grey area in the model, where design aspects from engineering are incorporated with the software domain of configurators. The procedure is initiated by a design or handling problem and ends with a customized solution created by the user of a configuration system. As expressed in the model, supporting methods can be informal, relying on subjective interpretations of domain experts, or formal, involving codable and systematic procedures. The two alternative approaches may be organized along a five phase model of exploration, generation, evaluation, implementation and communication, which is based on the established development model of design science (Cross, 2008).

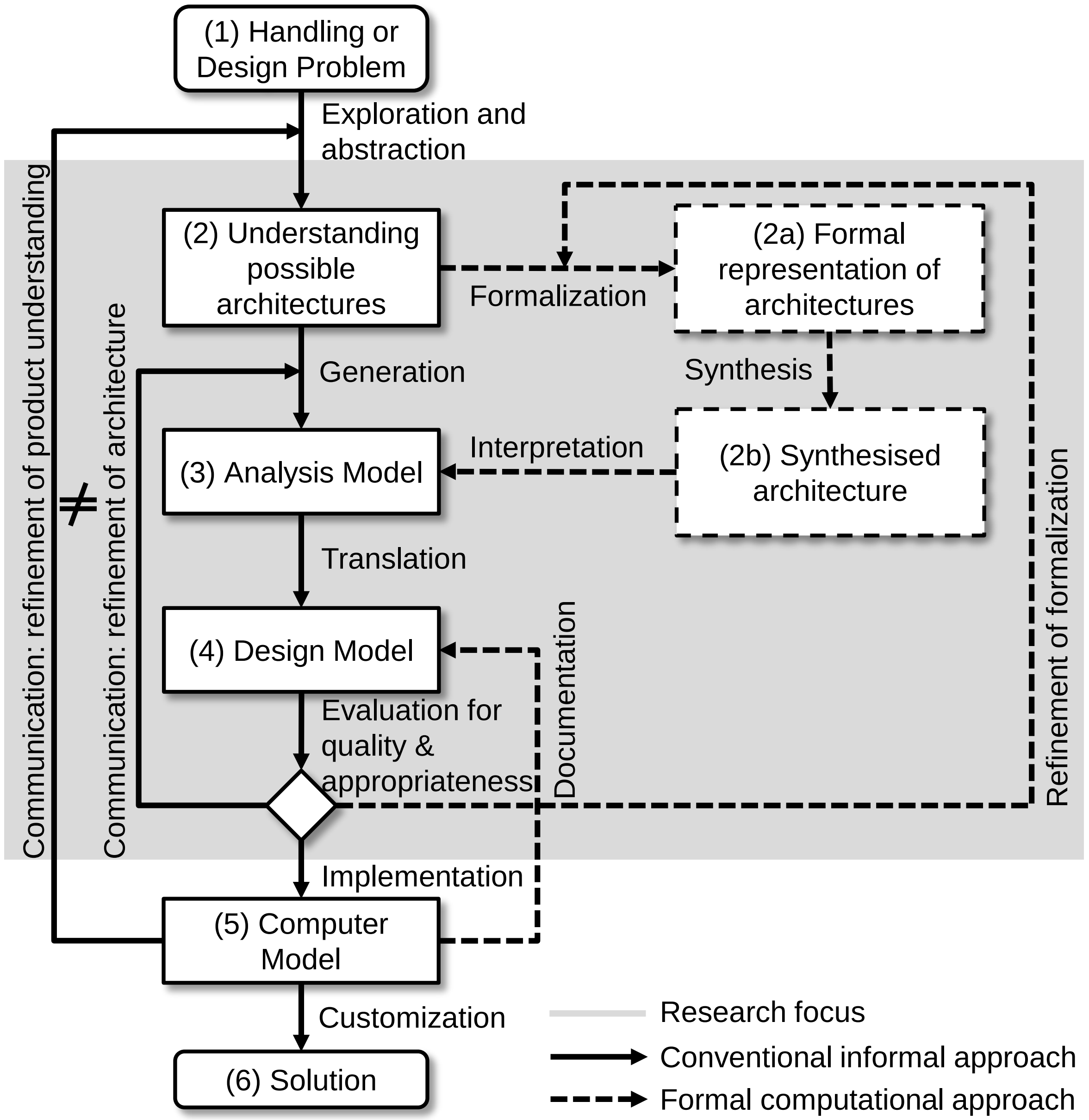

(1) Handling or Design Problem
Exploration and abstraction
(2) Understanding possible architectures
Formalization
(2a) Formal representation of architectures
Synthesis
(2b) Synthesised architecture
Interpretation
Generation
(3) Analysis Model
Translation
(4) Design Model
Evaluation for quality & appropriateness
Documentation
Refinement of formalization
Implementation
(5) Computer Model
Customization
(6) Solution
Communication: refinement of product understanding
≠
Communication: refinement of architecture
Research focus
Conventional informal approach
Formal computational approach

Fig. 1: A consistent model for designing and mass customizing product family architectures based on informal and formal methods (after Wyatt et al., 2011)

In analogy to Wyatt et al.'s (2011) architecture design framework, the informal approach can be described as follows:

- *Exploration* helps engineers to examine the handling of existing design or the work on a new design problem. Typically, product information can exist in many different formats, such as diagrams, tables, formulas, computer aided design (CAD) files, bills of materials (BOMs) etc. Different departments within a company may even have their own representations of products. By *abstracting* the relevant product information (1), engineers develop an understanding of possible architectures (2).

- Based on a created understanding of possible architectures, engineers *generate* a specific family architecture in the form of an analysis model, which may be the same as previous solutions and further contain errors (3). Discussions on the product architecture during the object-oriented analysis may involve various domain experts coming from product design to sales and marketing. Since not all departments are necessarily familiar with the same technical details of a product this is often done by visually representing the product family in graphical models and describing the combinatorial possibilities in a way which is similar to natural language. For instance, using pseudo-code for constraints instead of mathematical expressions, in the form of ‘component A has to be as wide as component B’, makes the models more appropriate for a cross-disciplinary communication.
- The analysis model has to be *translated* into a design model (4), which is more suitable for the subsequent *implementation* in a computer model (5). The aim of this step is to adjust the representation language of the analysis model into a format which is common to one of the final computer models of the configurator. Rules describing the combinatorial feasibility and solution principles of a product family have to be expressed in mathematical equations, making them readable and understandable by the software. In addition, the product family architecture may be extended with information related to the configurator design, such as the user interface, details on the implemented methods or the interaction with other IT systems. Depending on the experience of the project stakeholders, in praxis this step may not be strictly separated from creating the analysis model, but often involves further detailing of the architecture.
- The design model is *evaluated for quality and appropriateness* to determine whether the created solution fulfils the problem at hand in the best possible way. Has the architecture been accepted, the design model can be *implemented* as a computer model (5) in the configuration system. If not, the architecture is *communicated* to the design team, to *refine the solution* iteratively.
- Users (internal or external) of the configuration system can *customize* their solution based on the implemented computer model. If the offered solution space is either faulty (wrong configuration) or does not reflect the desired variety (missing or unwanted configuration), the computer model may be *communicated to refine the understanding* of the problem. Though both aspects are critical for the acceptance of the configurator, the latter becomes particularly important in markets where demands are frequently changing and enterprises need to keep pace with these changes. Since with this informal approach, no mechanism is typically established to ensure the constancy between design and computer model, the two communication processes illustrated in Fig. 1 do not necessarily represent the same product architecture and are thus to be considered separately.

The described informal methods for architecture generation largely depend on human creativity and may include simple brainstorming principles (Osborn, 1963), and more guided brainwriting concepts (Heslin, 2009). However, architectures can be created in many different ways (Kimmance et al., 2004). The qualitative character of the design space makes it difficult for domain experts to develop new architectures, or even to be able to consider alternative solutions for a product family (Wyatt et al., 2011). If lacking a systematic guidance, domain experts often base their work on experiences from previous design problems. When a new design task occurs, they tend to commit early to familiar solutions which may be premature and not well suited for the underlying problem. This so called fixation effect restricts practitioners from constructing previously unknown yet potentially better solutions (Purcell and Gero, 1996). In the same way, fixation has a detrimental impact on the quality of the architecture in the computer model. To guide developers in creating new models, modern configurators contain knowledge base editors and supportive debugging methods (Liao, 2005). They assist software experts in constructing executable computer models within the software environment, but fail to abstract, document and represent the product architecture so that it can be retrieved and communicated effectively (Li et al., 2011). Hence, configurator experts have little or no possibility to collaborate with domain experts when developing computer models, which additively reinforces the fixation problem. Moreover, they have to go through architecture models with potentially thousands of elements within the configurator and manually compare them with the previously developed architectures without being able to abstract adequately the underlying design problem.

2.3 Opportunities with formal architecture design methods

As acknowledged in the literature, the limitations of the informal approach can be critical for the quality of the obtained architecture. Especially as the complexity of the designs increases, formal approaches are becoming increasingly important (Prasad, 1998). In complex design problems they are often based on computational models which are used to synthesize potential architectures (Cagan et al., 2005). In order to evaluate a solution based on a formal synthesis, the architecture problem has to be made explicit, thereby providing a transparent and more reliable form of reasoning. In addition, proper documentation and knowledge representation methods may enable an intuitive comparison of architectures and hence increase the reliability of the expert system (Verhagen et al., 2012). The dashed lines in Fig. 1 illustrate how the challenges with the informal approach can be addressed by a formal computational solution:

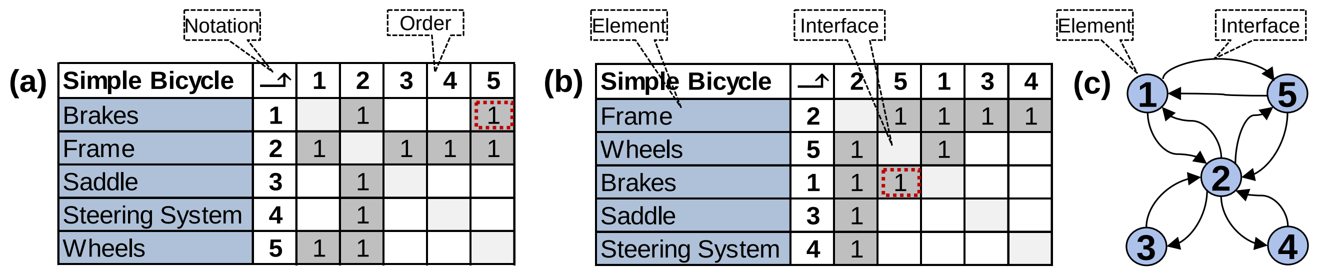
- The understanding of the possible architectures (2) can be *formalized* through a guided modelling environment and the representation of alternative architectures (2a).
- The computational methods assist domain experts in *synthesizing* a possible architecture (2b), which is then *interpreted* as an analysis model (3), and further translated into a design model (4). If the solution does not meet the evaluation requirements of the underlying problem, the development team may iteratively *refine the formalization*. Different measures relevant for the assessment of the particular design problem can be selected and graphically presented.
- Has the design model been accepted, it may be *implemented* as a computer model (5). To ensure the consistency between computer and design model, it needs to be *documented* and compared against the design model. C*ommunication* helps to refine the architecture and/or the product understanding, which may be internal (towards the development team) or external (towards customers). Since product architectures are typically developed iteratively over time, for large and interconnected models proper documentation and communication becomes particularly important (Maier et al., 2009). In such cases the documentation and communication of already developed implemented architectures is a prerequisite for any further development.

Methods representing entire products are often based on ontologies, i.e. grammars applied to graphs to display architectures (Schmidt and Cagan, 1997). A widely used technique for such graphs is to map architectures through their structure with nodes and links, i.e. to create an abstract representation of the underlying elements identified by their type and relations (Andreasen et al., 1995). To obtain a deeper understanding for a supportive method, the next section uses an illustrative modelling example to address briefly the limitations of existing grammar graphs.

2.4 Modelling product family architectures with grammar graphs

2.4.1 Adjacency matrices

As one of the first supporters of an abstract modelling method, Steward (1981) applied adjacency matrixes also known as design structure matrices (DSMs) to display relationships of entities (functions or components) of the same type for single products (Steward, 1981). DSMs provide a well-organized and compact representation, since each element is represented by a row and a column, while entries in the DSM indicate a link from one node of the matrix to the other. Two different conventions exist in the literature to describe the direction of a link. The IR/FAD convention uses element inputs shown in rows and outputs in columns. The IC/FBD convention of the other hand shows element inputs in columns and outputs in rows. The two notations are based on the same information, where one is the matrix transpose of the other (Eppinger and Browning, 2012). To illustrate the functionality of the DSM method, we use the letter notation in a simplified modelling example of a manufacturer offering customized bicycles. As model (a) in Fig. 2 shows, our bicycle consists of five main components. All elements have been ordered alphabetically and their interfaces to each other are shown through the entities of the DSM. In this way the product structure consists of interconnected components shown as a squared intra-domain matrix. Alternatively, to represent the structure of two different domains within the matrix, e.g. components and functions, the additional domain may be listed on the other axis. This variation of the DSM is also called domain mapping matrix (DMM) and is based on the same modelling notation as the DSM. The DSM layout requires product elements to be listed strictly on the horizontal and vertical axes, making it a rather rigid but at the same time very compact and scalable way of describing structures of single products (Abuthawabeh et al., 2013). This well-defined arrangement has proved to be particular useful for computational analysis methods. For example, a common way to identify potential modules is to cluster the links between elements in chunks. This method is illustrated in model (b) of Fig. 2. The order numbers in the DSM indicate how the elements have been rearranged compared to the alphabetical order to form a potential module.



Simple Bicycle		1	2	3	4	5
Brakes	1		1			1
Frame	2	1		1	1	1
Saddle	3		1			
Steering System	4		1			
Wheels	5	1	1			

Simple Bicycle		2	5	1	3	4
Frame	2		1	1	1	1
Wheels	5	1		1		
Brakes	1	1	1			
Saddle	3	1				
Steering System	4	1				

Fig. 2: Different analysis models of a hypothetical bicycle, (a) DSM (alphabetical order), (b) DSM (clustered), and (c) a node-link diagram

2.4.2 Node-link diagrams

Node-link diagrams offer an increasingly popular alternative graphical representation of product structures for single products. Initially such graphs widely been used in social network analysis studies with the purpose of characterizing the nature of social relationships among a set of actors (Freeman, 2004). Each actor (component or function) within a given network is represented in a node and arrows between the nodes stand for links between them. To display their relationships, the nodes of a model can be placed freely within the entire two-dimensional space, making this type of graph very flexible in layout. Especially for large models with many nodes, this flexibility can be very convenient. Model (c) in Fig. 2 illustrates our bicycle example in form of a standard node-link diagram. As indicated by the layout, the frame is central for the entire structure of the model. It provides input to all other components and at the same time is connected through the same number of interfaces from them. Depending on the actual analysis problem, a rearrangement of the graph allows the user to access visually only relevant network areas leaving less important aspects unconsidered (Ghoniem et al., 2005). Additional colour and distance coding may help to display social clusters and the strength of individual relationships. An extensive study of algorithms for drawing node-link graphs can be found in (Battista et al., 1994) for example.

2.4.3 Generic product models

Popular models representing entire product families typically include the product family architecture (PFA) approach (Jiao et al., 1998), the use of class diagrams and CRC cards (Aldanondo et al., 2000), the frames parts components (FPC) model (Magro and Torasso, 2003), and the product variant master (PVM) (Hvam et al., 2005). The majority of the so called generic methods utilize variations of object-oriented modelling based on the UML standard to describe hierarchical composition of elements (generic part-of-structure), their possible variants (kind-of-structure), and their combinatorial interfaces to other elements (collaborations) (Felfernig et al., 2000). The UML notation includes the object constraint language (OCL) as an expression language of how elements in a model are combined with each other. Due to their additional notation, generic methods can be regarded as an extension to the structural representation of the grammar graphs discussed above.

Fig. 3 displays an example of the bicycle model expressed in the PVM notation introduced in Sect. 2.5. Similar to assembly models in computer aided design (CAD) systems, the model imitates the aggregation of elements through a hierarchical list connected with lines. The different colour codes represent the element type and the letter size indicates the corresponding hierarchy level. In general there are four different element types in a model: *parts* (functions or components), *kinds* (variants), *attributes* (properties), and *constraints* (rules). Each part and kind element stands for an *object* or *class* in the model. As an example, a wheel is an object in the model and a different wheel type is modelled as a separate object. The character of an object can be explained by attributes and constraints. *Attributes* are defining the properties of an element, i.e. length or with of a wheel, while the constraints are specifying how these properties operate within the product. An important difference between *parts* and *kinds* is that parts can have both sub-parts and sub-kinds, while kinds may only include other sub-kinds, e.g. a van may be a family van or a transporter (Haug et al., 2010). The *cardinality* of parts is indicated by an index above each part. It defines how many times a particular component is to be found in the model and whether this component is optional *(0,1..n)* or mandatory *(1..n)*. To illustrate the representation of hierarchies and variants, further details have been added to the bicycle model. The steering system of the bicycle can for example be described as the *aggregation* of a front fork and a handle bar. If viewed separately, each of the two components has an individual set of *attributes* and *constraints*. For example, the front fork has a clamp diameter that needs to fit with the wheels and the handle bar requires a certain type of brake system. Only in combination however, they create the required functionality for steering. As shown with the DSM technique, without this part-of structure we would have to decide which level to focus on at the first place, leaving out many other essential aspects unrevealed.

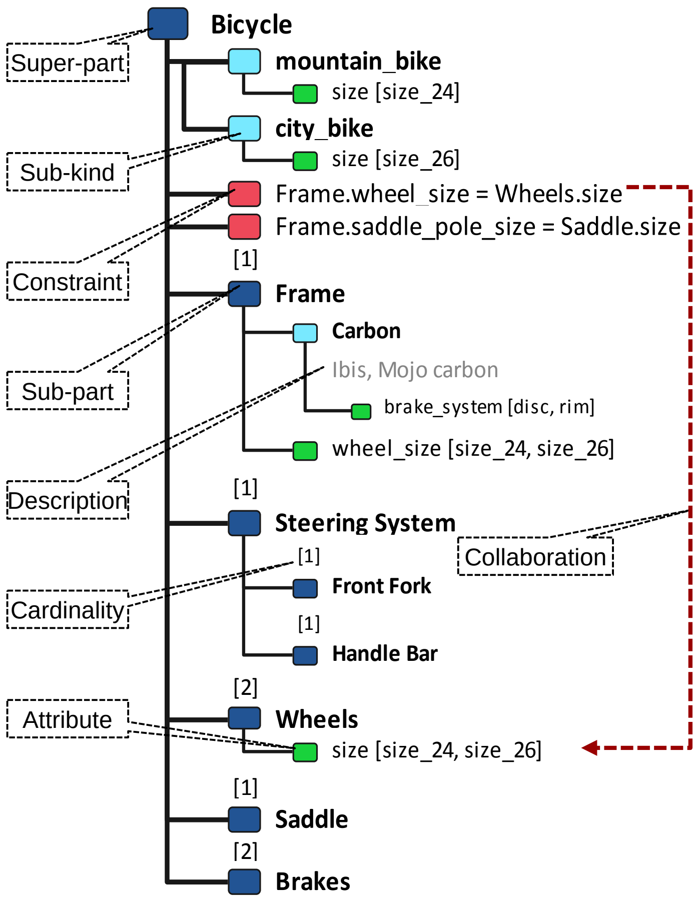


Fig. 3: PVM example of a hypothetical bicycle family

The principle of constraints can be illustrated by two additional examples which have been included on the top level within the model. To obtain a design model (see Sect. 2.2), the constraints use attributes with mathematical equations to specify the geometrical relationship between the frame, the wheels and the saddle. Another important feature of such object-based models is the concept of *inheritance* and *encapsulation*. *Inheritance* means that the sub-kinds of elements inherit the generic properties of the super-element. For example, all bicycles have the same major components shown in Model (c). A mountain bike however may have a particular wheel size. *Encapsulation* on the other hand restricts objects at the same hierarchy from interfering with each other's properties. This means that a relationship between two components from the same hierarchy can only be expressed by constraints on the parent object. In this case, the bicycle frame has to fit with the wheels and the pole size of the saddle has to fit with the equivalent size of the frame, which has to be listed directly under the super-part of the model, as indicated by the dashed arrows in Model (c). In object-oriented modelling these interfaces are referred to as *collaboration* or *association* between two objects. In accordance with the common modelling environment of modern model-based expert systems (Acatec, 2014; Oracle, 2014; Tacton Systems, 2014), typically the PVM notation provides no standard visual representation for such a connection. Hence, because all interfaces between components are expressed though constraints, the generic approach alone proved to be disadvantageous when it comes to documenting and analysing the structural properties of product architectures (Bodein et al., 2014). As studies within the automotive industry show, especially for complex products designers and software engineers found it difficult to identify the relevant relationships among product elements, which creates additional challenges for changing and verifying existing architectures (Salehi and McMahon, 2011). Moreover, the extended syntax of the generic methods requires some experience in creating valid architectures. Modelling mistakes can easily occur if no systematic guidance through dedicated modelling tools is provided, which however is missing to date. As a result, incorrect generic models can even be observed in examples provided in the literature, where for instance inheritance has been ignored (Haug et al., 2010).

2.5 Requirements for a formal architecture synthesis

The majority of methods for formal architecture design synthesis are based on engineering design literature (Chakrabarti et al., 2011). They vary from numerical optimization approaches of partial design problems for single products (Ziv-Av and Reich, 2005), through heuristics for module optimization in product families (Jiao et al., 2008), to morphological analysis methods for incremental design improvements (Kurtoglu and Campbell, 2009). Based on Steward's work (Steward, 1981), a number of additional (computational) matrix-based techniques have been proposed over the years (Eppinger and Browning, 2012). Examples for computational design synthesis using structural grammar graphs can be found in (Kreimeyer and Lindemann, 2011). They included DSMs and node-link diagrams which can be used to create entire new architectures or to evolve existing ones. This is particularly useful since architecture design is typically incremental, where products are upgraded over time and their components are reused in alternative or later products (Clarkson et al., 2004).

Research dealing with configuration systems has likewise recognised the need for a formal architecture design and implementation approach, where for example the handling of complex highly connected models has been addressed explicitly (Tiihonen et al., 1996; Wielinga and Schreiber, 1997). In particular the challenge of documenting and communicating entire product family architectures has been discussed in several studies (Haug et al., 2010; Hvam et al., 2005). The authors conclude that the complexity of the models makes it infeasible to update and visualize each model manually without any guidance but requires dedicated methods and software tools. At the same time, comparable computer-based design synthesis methods as suggested for single product design are still missing. Hence, despite the advantages of computational synthesis methods, their application in industry has been limited. This may be partly explained by the mismatch between the needs for such methods in architecture design praxis and the systems developed hitherto. To overcome this, a related study conducted by Wyatt et al. (2011) combines systematic literature reviews with empirical investigations to propose general requirements that address the described aspects of formalization, synthesis, interpretation and refinement for single architecture design. The identified requirements are summarized in Table 1 and complemented with discussed context specific aspects of *documentation* and *communication* of product families.

Table 1: Requirements for a formal computational architecture synthesis for mass customization (after Wyatt et al., 2011)

Category	Requirement		Content
Formalization	F1	Incremental design	Guided architecture creation as a staring point for synthesis
	F2	Problem decomposition	Abstract sections and focus on relevant scope
	F3	Problem-specific architecture	Represent relevant elements in ways that fit the problem
	F4	Declarative evaluation	Declarative constraint-based representation of the solution space
Interpretation	I1	Interpretation support	Present synthesised architecture and allow for further evaluation
	I2	Feature-based evaluation	Specify structural features according to lifecycle objectives
Refinement	R1	Refinement of formalization	Support modification of the problem formalization
Documentation	D1	Consistent architecture design	Ensure architecture consistency throughout design and implementation
Communication	C1	Complete and correct representation	Consider graphically all structural aspects of product families

Since the design process is typically incremental, a *formal* method should guide engineers to specify initial architectures as a starting point for synthesizing new solutions. The corresponding design problems may then be decomposed into smaller interlinked sub problems, represented by the relevant model elements, while the possible solution space should be declared explicitly through constraints. For instance, architectures representing the energy consumption of diverse production plants might not necessarily include all elementary machine elements, but rather consider major factors (components and attributes) and their ranges influencing this value (Orsvärn and Bennick, 2014). Next, synthesized architectures should be presented and *interpreted* through their structural features which have favourable or unfavourable effect on any lifecycle objectives of the product family (Tang et al., 2009). Frequently used metrics for example investigate the commonality and modularity of different architectures (Jiao and Tseng, 2000; Sosa et al., 2007). The problem formalization may then be *refined* by the engineers as a consequence of their interpretation of the synthesized solution. The obtained architecture is *documented* to ensure its consistency throughout development and implementation, and is *communicated* in a correct and complete manner to modify the understanding of the problem. For example, new production lines might need to be added to an implemented model of a plant, for which the already created architecture would be required. Since the described recommendations and the underlying graphical methods are reduced to the special case of designing single product architectures, they have to be tailored to the context of this paper. The most profound aspect arguably addresses the ability to model, synthesize and communicate entire product family architectures using the discussed graphical grammar approach. The next section evaluates in more detail the existing informal methods and proposes ways to address them with a formal approach.

3 Evaluation and extension of formal architecture design methods

When considered separately, many of the requirements in Sect. 2.5 cannot be fulfilled with the modelling and synthesis methods discussed in Section 2, in particular:

- Grammar-based methods are by definition procedural and qualitative and need to be supplemented with constraints to obtain a descriptive explanation of the design problem (Requirement F4).
- While several clustering algorithms exist in the literature (Steward, 1981), our simple example also shows the limitations of such a method. Despite the small number of major elements, the components of a bicycle are connected in ways which do not allow creation of any obvious modules. This limitation is often compensated for by emphasizing on the visual display of structures (Requirement F2). Here, matrix-based representations have proved to be more suitable for most of the tasks in large and dense graphs (Ghoniem et al., 2005), which are very likely in the context of this study. Due to their rather restricted yet scalable layout, DSMs are applicable for products consisting of many interconnected components. As the layout of node-link diagrams is less prescribed, in complex product structures nodes and edges tend to overlap in ambiguous ways, making it challenging to navigate through the network and to identify patterns. However, if used properly in a dedicated software, a network representation with nodes and links is a still more intuitive graphical representation and better suited with respect to finding paths between two nodes (Keller et al., 2006) (Requirement F3). Nevertheless, the two methods do not consider any representation of hierarchical compositions or variants, making them too simplistic and impractical for modelling product families or for any form of customization (Requirement C1) (Keller et al., 2006; Malmqvist, 2002).
- The discussed grammar graphs as such do not provide any information about the quality of the identified architecture. To obtain more detailed understanding about possible features, additional external metrics (Requirement I2) and interpretation support (Requirement I1) is needed. Yet existing methods apply only for single products (Lindemann et al., 2009), giving the need to develop new methods. This may be partly overcome with the extended notation of the generic models, such as the PVM. However, these models do not visually represent interfaces (collaborations) between

components (Requirement F2), but rather use mathematical equations to express such through constraints (Requirement C1). Furthermore, the extended modelling notation requires additional guidance to obtain correct architectures (Requirement F1).

- Documentation is not explicitly supported by the existing methods but requires additional mechanisms, increasing the risk for obtaining an inconsistent architecture design (Requirement D1).

The evaluation of the methods suggests that several of the requirements can be addressed explicitly by extending the existing notation of the relatively simple grammar based approach of DSMs and node-link diagrams. Especially Requirement F2, F4 and C1 can be met directly with a modelling technique which includes aspects of the generic grammar, but which also provides a complete graphical representation of structures. Fig. 4 presents how such an extension may be realized in correspondence with the common perception that multiple views of an architecture help to understand better the underlying design problem (Keller et al., 2005) (Requirement F3). The so called integrated design model (IDM) combines the different functionalities of DSM, node-link graphs and PVM into a consistent representation form. Model (a) in Fig. 4 shows the generic structure of the bicycle in a matrix format (generic DSM). In addition to the main components from Fig. 2, rows and columns in the model may include sub-parts, kinds, constraints and attributes. Entries in the matrix are used to express existing interfaces for part-of structures, kind-of-structures and collaborations. The scalable layout of a DSM further allows to consider two additional types of interfaces. *Constraint-links* define which attributes are being used in this particular constraint, while *attribute-links* display the connection between these attributes. Accordingly, collaborations exist whenever there are constraints causing an interface between two objects. It is worth noting that interfaces caused by constraints are by definition symmetrical, which in our example means that both frame and wheels have to fit to each other. Hence, entries for attribute-links and collaborations appear on both sides of the matrix diagonal.

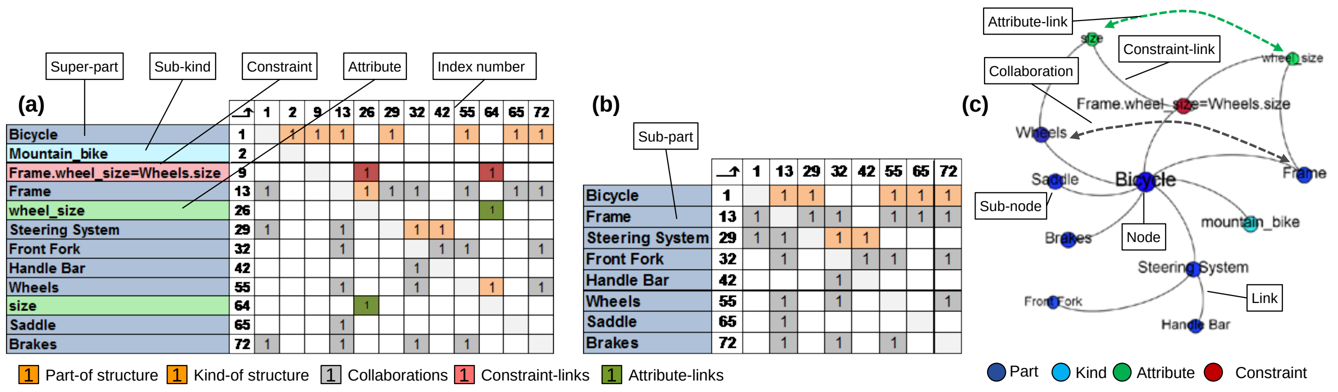


Fig. 4: Different views of a generic product structure, (a) generic DSM (partly collapsed), (b) generic DSM (collapsed), (c) generic node-link diagram

The extended notation of the generic DSM enables users to *abstract* the underlying architecture design problem (Requirement F2), which may be done by: (1) changing the *level of detail*, i.e. connections represent the architecture at any level of granularity, and (2) changing the *scope*, i.e. to focus on a particular set of elements, without altering the remaining architecture. The principle of abstraction can be demonstrated by comparing Model (a) and (b) in Fig. 4. While the first model is to some extent showing a higher level of detail of the entire model, Model (b) displays the same generic architecture of the bicycle family in a fully collapsed format, which is indicated by the visible elements and their index numbers. Especially for large graphs it can be very useful to create an initial overview over architectures by filtering out details in the model, without taking away any existing interfaces

(Elmqvist and Yi, 2013). The same generic structure can be expressed by analogy with a generic node-link diagram. To limit the discussed risk of having overlapping elements and connections in large and dense graphs, Model (c) narrows the representation of interfaces to the essential aspects. Hence, part-of-structures, kind-of-structures, and constraint links are expressed as previously described, leaving out redundant connection types (dashed arrows). Engineers can benefit from the graphical advantage of quickly identifying patterns and following important paths in the model (Requirement F2), without losing the required understanding for the present interfaces. The context of interfaces is preserved by using the original naming of all elements, which may be particularly important when investigating the cause of collaborations between two components (Requirement F3-F4).

Metrics can then be applied to evaluate a quality of a specific aspect of an architecture with respect to any lifecycle objective of the product (Huang, 1996) (Requirement I2). Key measures addressing product architectures typically include aspects of variant-oriented design (see Sect. 2.5), i.e. product complexity (Sinha and de Weck, 2013), normal or weighted modularity (Gershenson et al., 2004; Sosa et al., 2007), or communality (Thomas, 1992), and may in combination or alone access the considered design problem. Moreover, the significance of parts to a particular design problem are generally rated based on their influence on other components (active sum) and the impact other parts have on them (passive sum) (Lindemann et al., 2009). However, since the majority of metrics proposed in the literature are based on graph-theoretical characteristics of social networks (Bounova and de Weck, 2012), they need to be adjusted to the convention of generic structures for product families. Table 2 describes the adjusted measures with respect to their impact on the design work of the entire architecture of a family or to a chosen sub-section *A*. Metrics 1 to 4 in the table represent basic characteristics of the architecture, i.e. the number of parts, kinds, attributes and collaborations. Metrics 5 to 10 indicate the discussed structural properties, while measure 11 refers to the classification of how significant a constraint is with respect to the underlying design problem.

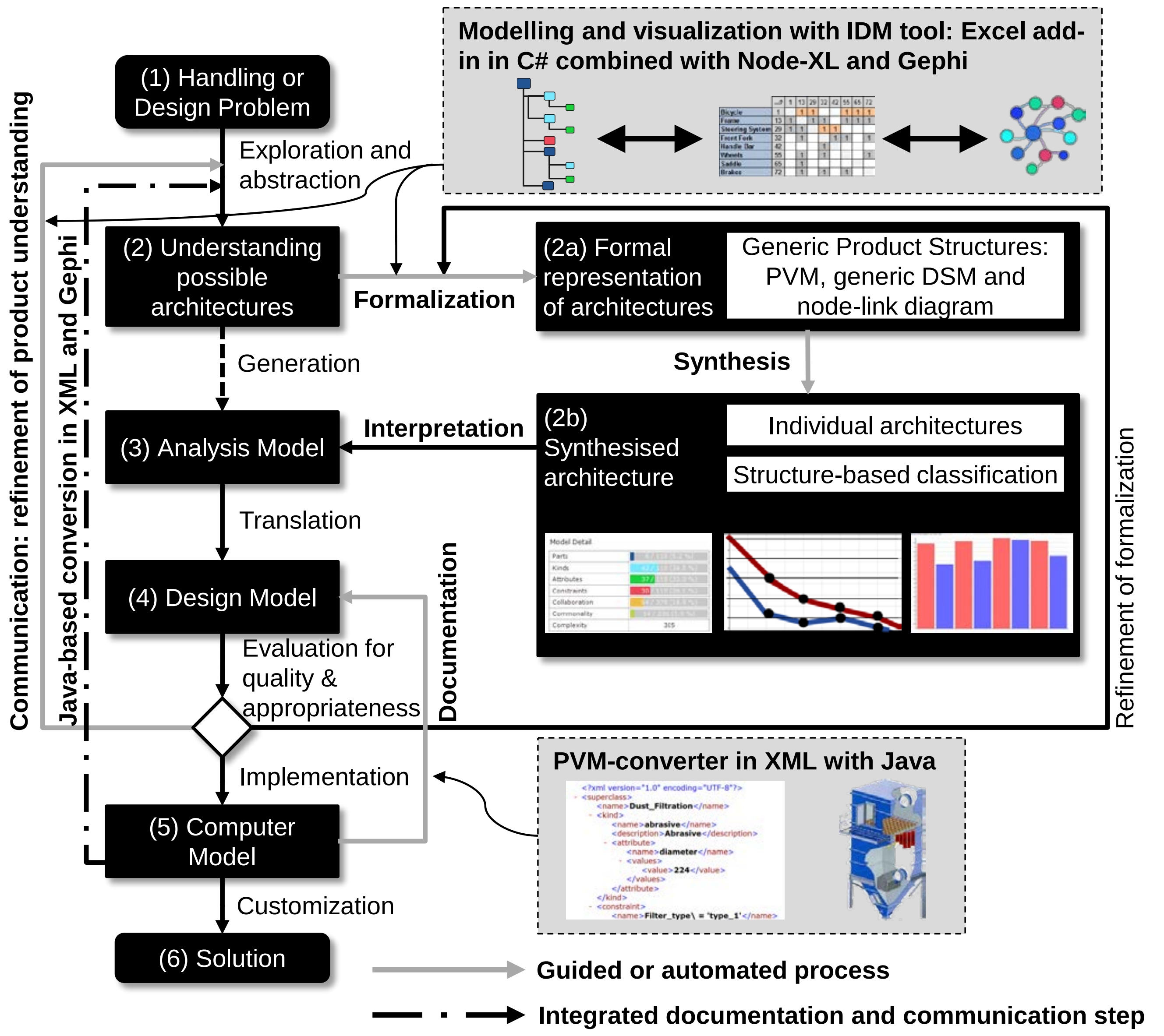
Table 2: Structural metrics in support of a quantitative architecture evaluation

Metric	Formula	Description	Normalized by
1 Parts	$n_p(A) = \sum_{i=1}^{n} p_i$	*More* parts require *more* design work (Hodbay, 1998): *the sum of all parts* p_i *in architecture (section) A containing n elements*	n
2 Kinds	$n_k(A) = \sum_{i=0}^{n} k_i$	*Higher* variety requires *more* design work (Martin et al. 2002): *the sum of all kinds* k_i *in architecture (section) A containing n elements*	n
3 Attributes	$n^{(u)}{}_a(A) = \sum_{i=1}^{n_p} \underbrace{(\sum_{j=0}^{n_g} a^{(g)}{}_j + \sum_{t=0}^{n_v} a^{(v)}{}_t)}_{a^{(u)}{}_i}$	*More* functionality requires *more* design work (Sinha et al., 2013): *the sum of all unique attributes* $a^{(u)}{}_i$ *in architecture (section) A containing* n_p *parts, with all generic attributes* $a^{(g)}{}_j$ *and all variant attributes* $a^{(u)}{}_t$	n
4 Collaborations	$Col(A) = \sum_{i=1}^{n_p} c_i$	*More* interfaces require *more* design work (Sosa et al. 2007): *the sum of all collaborations* c_i *in architecture (section) A containing* n_p *parts*	Maximum possible degree $Col(A)_{max}=n_p*(n_p-1)$
5 Communality	$Com(A) = \frac{\sum_{i=1}^{n_p} \overbrace{(p_i \sum_{j=0}^{n_g} a^{(g)}{}_j)}^{\alpha_i}}{\sum_{i=1}^{n_p}(p_i \sum_{j=0}^{n_g} a^{(g)}{}_j) + \sum_{i=0}^{n_k}(k_i \sum_{j=0}^{n_v} a^{(v)}{}_j)}$	*Higher* commonality requires *less* design work (Jiao et al., 2000): *the ratio between all common objects and their properties* a_i *(all parts times their generic attributes) compared to all objects and their properties (all parts times their generic attributes plus all kinds times their variant attributes) in architecture (section) A*	Maximum possible degree $Com(A)_{max}$ for $n_k=0$ or $n_v=0$
6 Complexity	$C(A) = n_p(A)+n_k(A)+Col(A)$	*Higher* (structural) complexity requires *more* design work and communication effort (Prasad, 1998); it is based on number of components, their variety and interdependence (Geraldi et al., 2011): *the sum of all parts, kinds and collaborations in architecture (section) A*	-
7 Active sum	$As(A) = \sum_{i=0}^{n} l^{(a)}{}_i$	Parts with *high* active sum are *more* significant in design (Lindemann et al., 2009): *the sum of all interfaces l(a) that emerge from a part*	-
8 Passive sum	$Ps(A) = \sum_{i=0}^{n} l^{(p)}{}_i$	Parts *with* high passive sum are *more* influenced in design (Lindemann et. Al, 2009): *the sum of all interfaces l(p) that affect a part*	-
9 N-modularity	$N_m(A) = \frac{\sum_{i=1}^{n_m} l^{(m)}{}_i}{\sum_{j=1}^{n_o} l^{(o)}{}_j}$	*Higher* modularity facilitates variety and concurrent design and maintenance; modules are tightly connected components inside a cluster and loosely connected to others (Sosa et al., 2007): *the normal ratio* $N_m(A)$ *between all interfaces l(m) within a selected section n(m) compared to it's interfaces to other parts l(o) in architecture (section) A containing n(o) parts*	-
10 W-modularity	$W_m(A) = \frac{\sum_{i=1}^{n_m} w^{(m)}{}_i}{\sum_{j=1}^{n_o} w^{(o)}{}_j}$	Modularity may be *weighted*, to account for *multiple* connections between two components (Gershenson et al., 2004): *the weighted ratio* $W_m(A)$ *between all interfaces w(m) within a selected section n(m) compared to it's interfaces to other parts w(o) in architecture (section) A containing n(o) parts*	-
11 Constraint active sum	$Cas(A) = \sum_{i=0}^{n} l^{(c)}{}_i$	Constraints with *high* active sum are *more* significant in design (after Lindemann et al., 2009): *the sum of all constraint links that emerge from a constraint l(c)*	-

4 Case study: applying a proposed formal computational synthesis method

4.1 Company background and data collection

As Sect. 2.2 discussed, the conventional informal approach to architecture design implies a number of challenges influencing the quality of the implemented computer model. Drawing on the established insight from Sect. 3, a formal synthesis method addressing these challenges has been developed. To validate it's applicability for a real case, the method has been tested in an empirical architecture design problem at a major provider of plant and machinery applications. The case study was established throughout 2014 and involved several semi-structured interviews lasting between 1 and 2 hours as well as half-day workshops with a team of domain experts, one group leader, one engineer and one IT expert. The three domain experts are part of a larger physically disconnected team, which is responsible for the coordination of the architecture design and its implementation in a configuration system. In addition, full access was given to architecture models of selected product families and their development over a period of 12 months. The objectives for the study were (1) to identify the major concerns for the architecture design and implementation process and (2) to address them with a new approach for generating architectures through the formal synthesis method in Fig. 5.


Modelling and visualization with IDM tool: Excel add-in in C# combined with Node-XL and Gephi
(1) Handling or Design Problem
Exploration and abstraction
(2) Understanding possible architectures
Formalization
(2a) Formal representation of architectures
Generic Product Structures: PVM, generic DSM and node-link diagram
Synthesis
Generation
(3) Analysis Model
Interpretation
(2b) Synthesised architecture
Individual architectures
Structure-based classification
Translation
(4) Design Model
Evaluation for quality & appropriateness
Documentation
Implementation
(5) Computer Model
Customization
(6) Solution
PVM-converter in XML with Java
Communication: refinement of product understanding
Java-based conversion in XML and Gephi
Refinement of formalization
Guided or automated process
Integrated documentation and communication step

```
<?xml version="1.0" encoding="UTF-8"?>
- <superclass>
    <name>Dust_Filtration</name>
  - <kind>
      <name>abrasive</name>
      <description>Abrasive</description>
    - <attribute>
        <name>diameter</name>
      - <values>
          <value>224</value>
        </values>
      </attribute>
    </kind>
  - <constraint>
      <name>Filter_type\ = 'type_1'</name>
```

Fig. 5: Proposed formal method to support product family architecture design and customization

4.2 Documenting and communicating implemented architectures

4.2.1 Consequences of the informal approach

At the beginning of the case study, the company employed an informal approach as described in Sect. 2.2. In response to an increased market demand for rapid and robust customization, a growing part of the product line was incorporated into the commercial configuration system Tacton (Tacton Systems, 2014). By the beginning of 2014, more than 30 different product families of highly customizable industrial applications, e.g. conveyors, pumps and valves, were used internally by product managers and technical salesmen to support customers in specifying their product requirements. Comparable to other modern configurators, the software provides an object-oriented development environment as described in Sect. 2.4.3 for the design of generic architectures. A representative product family architecture consists of several thousand interconnected elements and may include components that are produced internally or sourced externally by sub-suppliers. The architecture design is generally organized as an incremental process with regular iterative steps and requires the latest architecture to be used as a starting point for the new solution. The objective of the design work typically involves considerations for how increase the reuse of *common* parts, while maintaining the necessary product variety or simply for how to *document* the implemented architectures in order to comply with changing legal requirements.

The lack of a formal and/or integrated computer support forced the organization to use a considerable amount of resources for designing and coordinating developed architectures. Since the computer models per se can neither be extracted nor visually displayed, design initiatives have to be compared manually against the implemented architectures within the configurator, making it exhausting to focus on the *significant* parts of the relevant scope. Moreover, both product managers and engineers find it difficult to verify if a committed design objective, e.g. to increase *modularity* of certain sub-assemblies, has actually been obtained. Even if substantial rework may be done to achieve this goal, the informal approach provides no method to demonstrate any positive evidence pointing towards the obtained result. As the group leader comments *"we are forever bound to a system with which we cannot document or coordinate our work properly, especially if we want to discuss our [architecture] designs with our developers or external suppliers"*. In consequence, the insufficient control mechanism of the informal approach increases the risks for delayed product launches and inconsistent architectures.

4.2.2 Applying a proposed documentation and communication strategy

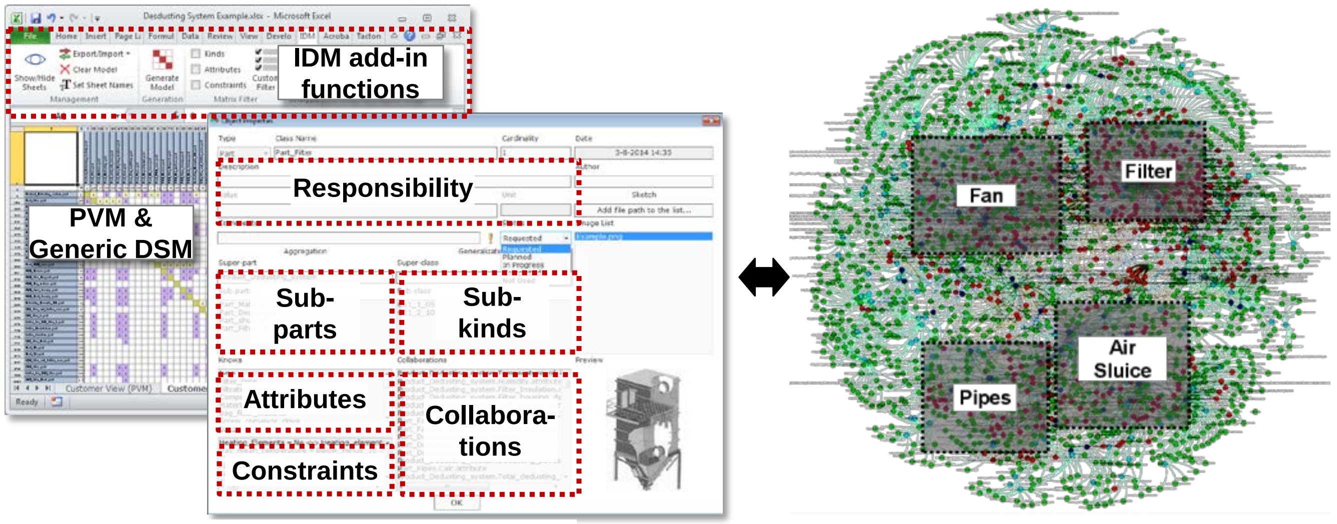
Being aware of this challenging situation, the responsible engineers and configuration software experts are pressured from several directions. They have to improve their productivity when designing and implementing architectures and to provide more transparent planning reports to the product management about their progress, which is further *communicated* to the board of directors. This may be achieved by the method illustrated in Fig. 5. The displayed scheme proposes a pragmatic solution which could be implemented and tested within the limited time of the study. Comparable to many other computer systems, the employed configurator allows by default to save the computer files in the Extensible Markup Language (XML). The XML standard is a text-based format which is frequently used to represent machine-readable structured information, such as documents, configuration status and invoices (XML Working Group, 2010). The XML files created by the configurator contain the encrypted product family architecture of the computer model along with other program specific information. This suggests that the computer models created in Tacton can with relatively little development effort be decoded or converted in a legible modelling format using XML. However, as no XML standard per se is capable of representing generic architectures, a format was created which resembles the discussed PVM notation in Sect. 2.4.3. This was done using a self-developed Java-based application called 'PVM converter'. The application utilizes simple data mining techniques to decode the relevant information within the configurator files and to restructure them in the PVM format. An example of an XML-based PVM file is indicated in Fig. 5. The illustrated XML syntax uses the integrated identifiers from the XML language to express part-of-structures, kind-of-structures, attributes and constraints. Apart from documenting the architecture of the computer model alone, the application complements the architecture with comments and path references to drawings which have been included within the knowledge base of the configuration system.

To obtain consistent design models and to communicate them effectively to various stakeholders, the created XML-based PVM models need to be expressed graphically with the discussed grammar methods. Since no dedicated software tools hitherto exist for creating the required generic architectures, a modest solution presented in Fig. 5 is to employ of the capabilities of existing open source software and to adjust it to the context specific requirements. This has been realized through an IDM application, a Microsoft Excel add-in which has been developed in C#. The IDM software is used to generate semantically correct PVM and generic DSM models out of the previously created XML-based PVM model. The software has been further combined with the freeware visualization software NodeXL and Gephi, which are two very frequently used tools for studying social networks with node-link graphs (The Gephi Consortium, 2014; The Social Media Research Foundation, 2014). An export function within the IDM software has been developed to ensure the consistency of the generic structure. It converts the XML-file into the discussed convention of node-link diagrams and exports it automatically into the relevant freeware formats, e.g. csv or .gephi. A major advantage of utilising widely accepted standard software is that the obtained solution can be established with relatively low development costs. Furthermore, as only little changes are made to the existing IT infrastructure in the company, the software is more likely to be accepted by the stakeholders.

The documentation and communication process may alternatively be combined into one integrated step so that any user of the configurator can directly share and discuss the latest version of a particular architecture. As shown in Fig. 5 this process has been realised at the case company by integrating a web-based function within the UI of the configurator, which when selected encodes the underlying computer model first into the described XML-based PVM file and subsequently decodes it into a node-link graph in the form of a svg- or pdf-based Gephi model. For an industrial company offering a variety of custom tailored products this automatic visualization of the entire generic structure proved to be very valuable in praxis. As the group leader reports *"product managers and technical salesmen [using the configurators] are typically very experienced with the [provided] products. Having a method which allows them to communicate instantly the [architecture] model in use graphically [through e.g. a web browser] increases significantly the transparency of the achieved solution and [eventually] enables the consideration of a larger amount of product experience into our design [process]"*. Thus, if used externally, the method may facilitate companies to engage their customers in co-creating new product functionalities and thereby to utilize external resources to drive their innovation processes (Martínez-Torres, 2013).

4.3 Formalization and synthesis of the architecture design

With the described documentation and visual communication of the computer model, the graphs were used to create an understanding of the design problem and to narrow the development effort to the relevant aspects. The design process was supported by the IDM software. Part (a) in Fig. 6 shows the modelling environment of the IDM tool, where equivalent to the guided knowledge base editor of modern configurators (Liao, 2005), the user is assisted in creating valid architectures inter alia by following the generic syntax discussed in Sect. 3. Data mining techniques have been further implemented in the software to guide the user in formulating feasible constraints and to consider the described aspects of encapsulation. To abstract the model towards the particular design problem, domain experts may choose to collapse or filter out unessential elements. CRC cards are automatically generated and include the implemented drawings, comments and implementation status of the computer model. In large design projects the latter feature can be very useful, as project experts are supported in keeping track of the development work and managing the responsibilities of tasks. Depending on the user's preferences, architectures can be designed within either the PVM or the generic DSM notation, where furthermore the user can switch dynamically between the IR/FAD and IC/FBD convention of the matrix. Eventually, to synthesize feasible architectures within a wider physically disconnected team of domain experts, each architecture was communicated using the generic models in any of the three grammar graph techniques. Part (b) in Fig. 6 displays an example of an automatically created generic node-link diagram of a dust filtration system. The graph shows a major section of the entire family, which in total consists of roughly 2000 elements. The product is installed in production environments exposed to extreme dust and dirt to keep critical manufacturing areas clean during operation and maintenance. In praxis this is achieved by creating a negative pressure in the production equipment in order to prevent that generated dust disperses to the surroundings. The major building blocks are illustrated by the shaded areas in the model and include a fan and a filter system, several pipes, as well as an air sluice.



(a) Modelling editor and CRC card

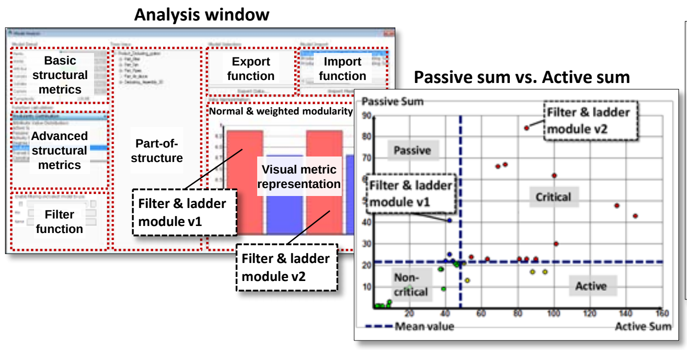
(b) Generic node-link diagram

Fig. 6: IDM software tool showing (a) a collapsed generic DSM model and a CRC card, and (b) the generic node-link diagram on the example design problem

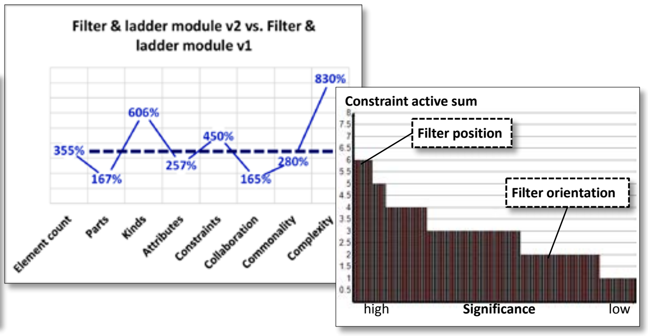
4.4 Interpretation and refinement of synthesized architectures

Architectures may be modified either through changing the structure of the model, i.e. by redefining the connections between elements, or through altering the objects as such. The letter may for example mean to add new components and/or functionalities to a product. To support domain experts in comparing architectures of different products or selecting between alternative ones of the same product, each model was evaluated quantitatively using the set of metrics introduced in Sect 3. Fig. 7 illustrates with an example how the structural measures can be included by domain experts to explore the synthesized results towards a preferred solution. To explore structural patterns or 'interesting' architecture areas, the information gained from a metric can in general be presented visually with commonly used chart formats, e.g. bar charts. This exploration process may be further assisted either by listing the values unsorted within the charts in the sequence of the index numbers in the model (see Sect. 2.4.1), or by showing them with an ascending or descending order against an absolute scale, e.g. time. This method was realised within a developed analysis tool for the IDM add-in.

Fig. 7 displays an extract of the method applied on an example problem of the dust filtration system, where two kinds of a 'filter and ladder module' were compared with each other. To evaluate their architecture quality and their impact on the remaining elements, the two alternative modules were assessed based on measures introduced in Section 3. As Part (a) of the graph indicates, both module types come with a similar overall modularity, yet since modularity is a relative measure of connectivity, their influence on the remaining architecture can be significantly different. In this case, compared to 'module v1', 'module v2' possesses both a higher active and passive sum, making it a relatively critical component for the overall architecture design. The reason for this effect can be explained by the particular structural difference illustrated in Part (b) of Fig.7. While the total number of elements of ‘module v2’ has increased by 355%. Moreover, the disproportional increase in part variety (kinds) and constraints even rises in structural complexity by 830%. This increase in complexity makes related design work more substantial to the overall architecture quality. Next, to identify the most significant interfaces to the connected modules, the related constraints can be ordered according to their active sum. This interpretation technique for example explicitly reveals that the individual ‘filter position’ creates considerably more interfaces than the related ‘filter orientation’. The different filter solutions from suppliers may for example be evaluated based on their alternative position within the module. To reduce the structural impact of ‘module v2’, the suggested design changes can then be evaluated iteratively with regard to their consequence on the overall architecture quality. Alternatively, if the obtained architecture satisfies the requirements of the domain experts, the hereby gained insight may help to establish a more transparent cost-benefit estimation of the synthesised solution.



(a) Normal and weighted modularity compared to active and passive sum



(b) Relative increase in basic structural measures and the related constraint active sum

Fig. 7: IDM software tool showing (a) a collapsed generic DSM model and a CRC card, and (b) the generic node-link diagram on the example design problem

5 Conclusions

Mass customization provides a promising concept to respond rapidly to individual customer needs. It requires from manufacturers to design effectively, implement and maintain suitable product family architectures in configuration systems, to support the customization process. Drawing on requirements as defined by researchers and industry, this paper evaluates the application of related modelling methods and formal computer-based approaches to facilitate this process. In particular, the paper argues that architectures can be presented explicitly through appropriate grammar graphs which consider common generic modelling standards of the UML language. This systematic documentation and communication of architectures allows the integration of a widespread internal product expertise as well as stronger customer engagement. Moreover, the quality of architectures may be evaluated objectively through computational structural analysis methods, making any assumptions about the obtained solution transparent and thereby accessible. The usability of the presented methods is demonstrated on an industrial case study of a major plant and machinery provider. The capabilities of a state-of-the-art configurator utilized at the case company are complemented with automatically generated grammar graphs using a developed XML-standard and a Java-based converter. Besides, the architecture design process is assisted through a computer-based modelling and analysis method termed IDM, consisting of modelling guidelines and visually represented structural metrics. The method integrates generic DSMs, node-link diagrams and PVMs in a coherent modelling environment. Further, a set of structural measures (e.g. modularity, communality, complexity, active sum) are used to evaluate the quality of a particular design problem and to classify the significance of the relevant parts and interfaces.

While the proposed formal computational approach supports the architecture documentation, communication and synthesis at the case company, the applied methods have been specifically designed to fit the particular needs of the studied industrial praxis. Future research may consider addressing these limitations and thereby extending the relevance of the presented methods. Specifically, the discussed documentation techniques may be applied to a variety of commercial configuration systems, for which the created XML-based generic modelling standard was developed. In addition, a dedicated modelling and analysis system may be developed to obtain a more stable and scalable software solution, which can be connected to various commercial configurators and visualization tools.

Moreover, for more assessable design problems, domain experts may not necessarily rely on any structural analysis support, but may rather trust their experience towards an improved cost-benefit result of the architecture. In this case, a structural analysis of the synthesised solution may in addition help to establish a more accurate estimation of the related design work and to communicate better a particular design progress. Related research may for example investigate the correlation between architecture complexity and the related lifecycle cost of a product family, module or component. Finally, depending on the clarity of the design objective, e.g. minimizing the complexity, and the computational capability, supplementary mathematical multi-objective optimization models may be developed. Such methods may help to increase the scalability of the discussed formal approach, by automating aspects of the architecture synthesis.

References

Aberdeen. (2008), *Tailoring Products to Customer Preference: Configuring Products to Order*, Aberdeen Group, Boston, MA, pp. 1–40.

Abuthawabeh, A., Beck, F., Zeckzer, D. and Diehl, S. (2013), “Finding structures in multi-type code couplings with node-link and matrix visualizations”, *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, Ieee, pp. 1–10.

Acatec. (2014), “Acatec Software GmbH”, available at: http://www.acatec.de.

Åhlström, P. and Westbrook, R. (1999), “Implications of mass customization for operations management: An exploratory survey”, *International Journal of Operations & Production Management*, Vol. 19, pp. 262–275.

Aldanondo, M., Rouge, S. and Veron, M. (2000), “Expert configurator for concurrent engineering: Cameleon software and model”, *Journal of Intelligent Manufacturing*, Vol. 11, pp. 127–134.

AlGeddawy, T. and ElMaraghy, H. (2013), “Optimum granularity level of modular product design architecture”, *CIRP Annals - Manufacturing Technology*, CIRP, Vol. 62 No. 1, pp. 151–154.

Andreasen, M.M., Duffy, A.H.B. and Mortensen, N.H. (1995), “Relation Types in Machine Systems”, *WDK Workshop on Product Structuring*, Delft University of Technology, Delft.

Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G. and Schäfer, R. (2003), “A Framework for the Development of Personalized , Distributed Web-Based Configuration Systems”, Vol. 24 No. 3, pp. 93–110.

Battista, G. Di, Eades, P., Tamassia, R. and Tollis, I.G. (1994), "Algorithms for drawing graphs: an annotated bibliography", *Computational Geometry: Theory and Applications*, Vol. 4, pp. 235–282.

Bodein, Y., Rose, B. and Caillaud, E. (2014), "Explicit reference modeling methodology in parametric CAD system", *Computers in Industry*, Elsevier B.V., Vol. 65 No. 1, pp. 136–147.

Booch, G. (1986), "Object-oriented development", *IEEE Transactions on Software Engineering*, Vol. SE-12, pp. 211–221.

Bounova, G. and de Weck, O. (2012), "Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles", *Physical Review E*, Vol. 85 No. 016117, pp. 1–11.

Brière-Côté, A., Rivest, L. and Desrochers, A. (2010), "Adaptive generic product structure modelling for design reuse in engineer-to-order products", *Computers in Industry*, Vol. 61 No. 1, pp. 53–65.

Cagan, J., Campbell, M.I., Finger, S. and Tomiyama, T. (2005), "A Framework for Computational Design Synthesis: Model and Applications", *Journal of Computing and Information Science in Engineering*.

Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N. V and Wood, K.L. (2011), "Computer-based design synthesis research: An overview", *Journal of Computing and Information Science in Engineering*, Vol. 11, doi:10.1115/1.3593409.

Clarkson, P.J., Simons, C. and Eckert, C. (2004), "Predicting Change Propagation in Complex Design", *Journal of Mechanical Design*, Vol. 126 No. 5, p. 788.

Cross, N. (2008), *Engineering Design Methods: Strategies for Product Design*, *Design*, Vol. 1, p. 230.

Duffy, A.H.B. and Andreasen, M.M. (1995), "Enhancing the evolution of design science", in Hubka, V. (Ed.),*Proceedings of ICED'95*, Zürich: Heurista, Praha, pp. 29–35.

Elmqvist, N.. and Yi, J.S. (2013), "Patterns for visualization evaluation", *Information Visualization*, Vol. 12, pp. 1–20.

Eppinger, S.D. and Browning, T.R. (2012), *Design Structure Matrix Methods and Applications*, MIT Press, Cambridge MA, p. 352.

Felfernig, A., Friedrich, G. and Jannach, D. (2000), "UML as domain specific language for the construction of knowledge-based configuration systems", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 10 No. 4, pp. 449–469.

Forza, C., Nicola, S.S. and Salvador, F. (1994), "Product configuration and inter-firm coordination : an innovative solution from a small manufacturing enterprise".

Forza, C. and Salvador, F. (2008), "Application support to product variety management", *International Journal of Production Research*, Vol. 46 No. 3, pp. 817–836.

Freeman, L.C. (2004), *The development of social network analysis*, *Document Design*, Vol. 27, p. 205.

Funke, M. and Ruhwedel, R. (2001), “Product variety and economic growth: empirical evidence for the OECD countries”, *IMF Staff papers*, Vol. 48 No. 2, pp. 225–242.

Gershenson, J.K., Prasad, G.J. and Zhang, Y. (2004), “Product modularity: measures and design methods”, *Journal of Engineering Design*, Vol. 15 No. 1, pp. 33–51.

Ghoniem, M., Fekete, J.-D. and Castagliola, P. (2005), “On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis”, *Information Visualization*, Vol. 4 No. 2, pp. 114–135.

Haug, A., Hvam, L. and Mortensen, N.H. (2010), “A layout technique for class diagrams to be used in product configuration projects”, *Computers in Industry*, Elsevier B.V., Vol. 61 No. 5, pp. 409–418.

Haug, A., Hvam, L. and Mortensen, N.H. (2012), “Definition and evaluation of product configurator development strategies”, *Computers in Industry*, Elsevier B.V., Vol. 63 No. 5, pp. 471–481.

Heslin, P. a. (2009), “Better than brainstorming? Potential contextual boundary conditions to brainwriting for idea generation in organizations”, *Journal of Occupational and Organizational Psychology*, Vol. 82, pp. 129–145.

Huang, G.Q. (1996), *Design for X—concurrent engineering imperatives*, Chapman & Hall, London, p. 508.

Hvam, L., Bonev, M., Denkena, B., Schürmeyer, J. and Dengler, B. (2011), “Optimizing the order processing of customized products using product configuration”, *Production Engineering*, Vol. 5, pp. 595–604.

Hvam, L., Malis, M., Hansen, B. and Riis, J. (2004), “Reengineering of the quotation process: application of knowledge based systems”, *Business Process Management Journal*, Vol. 10, pp. 200–213.

Hvam, L., Pape, S., Jensen, K.L. and Riis, J. (2005), “Development and maintenance of product configuration systems: requirements for a documentation tool”, *International Journal of Industrial Engineering*, Vol. 12 No. 1, pp. 79–88.

Jiao, J. and Tseng, M. (2000), “Understanding product family for mass customization by developing commonality indices”, *Journal of Engineering Design*, Vol. 11 No. 3, pp. 225–243.

Jiao, J. and Tseng, M.M. (1999), “A methodology of developing product family architecture for mass customization”, *Journal of Intelligent Manufacturing*, Vol. 10 No. 1, pp. 3–20.

Jiao, J., Tseng, M.M., Duffy, V.G. and Lin, F. (1998), “Product family modeling for mass customization”, *Computers & Industrial Engineering*, Vol. 35 No. 3-4, pp. 495–498.

Jiao, R.J., Xu, Q., Du, J., Zhang, Y., Helander, M., Khalid, H.M., Helo, P., et al. (2008), "Analytical affective design with ambient intelligence for mass customization and personalization", *International Journal of Flexible Manufacturing Systems*, Vol. 19 No. 4, pp. 570–595.

Keller, R., Eckert, C.M. and Clarkson, P.J. (2005), "Multiple Views to Support Engineering Change Management for Complex Products", *Coordinated and Multiple Views in Exploratory Visualization (CMV'05)*, Ieee, pp. 33–41.

Keller, R., Eckert, C.M. and Clarkson, P.J. (2006), "Matrices or node-link diagrams: which visual representation is better for visualising connectivity models?", *Information Visualization*, Vol. 5 No. 1, pp. 62–76.

Kimmance, A.G.., Anumba, C.J.., Bouchlaghem, D.M. and Baldwin, A.N. (2004), "The application of information modelling methodologies: the HIPPY approach to integrated project modelling", *International Journal of Computer Applications in Technology*, Vol. 20, p. 62.

Klenow, J. and Bils, J. (2001), "The Acceleration in Variety Growth", *American Economic Review*, Vol. 91 No. 2, pp. 274–280.

Kreimeyer, M. and Lindemann, U. (2011), "Complexity Metrics in Engineering Design", Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 401–403.

Kurtoglu, T. and Campbell, M.I. (2009), "Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping", *Journal of Engineering Design*, Vol. 20 No. 1, pp. 83–104.

Li, B.M., Xie, S.Q. and Xu, X. (2011), "Recent development of knowledge-based systems, methods and tools for One-of-a-Kind Production", *Knowledge-Based Systems*, Vol. 24 No. 7, pp. 1108–1119.

Liao, S.H. (2005), "Expert system methodologies and applications-a decade review from 1995 to 2004", *Expert Systems with Applications*, Vol. 28, pp. 93–103.

Lindemann, U., Maurer, M. and Braun, T. (2009), *Structural Complexity Management: An Approach for the Field of Product Design*, Springer, Berlin, Heidelberg, p. 248.

Magro, D. and Torasso, P. (2003), "Decomposition strategies for configuration problems", *Ai Edam*, Vol. 17 No. August 2003, pp. 51–73.

Maier, A.M.., Kreimeyer, M.., Lindemann, U.. and Clarkson, P.J. (2009), "Reflecting communication: a key factor for successful collaboration between embodiment design and simulation", *Journal of Engineering Design*, Vol. 20 No. 3, pp. 265–287.

Malmqvist, J. (2002), "A classification of matrix-based methods for product modeling", *7th International Design Conference*, Dubrovnik, pp. 1–10.

Martin, M. V and Ishii, K. (2002), "Design for variety : developing standardized and modularized product platform architectures", *Research in Engineering Design*, Vol. 13, pp. 213–235.

Martínez-Torres, M.R. (2013), “Application of evolutionary computation techniques for the identification of innovators in open innovation communities”, *Expert Systems with Applications*, Vol. 40, pp. 2503–2510.

Oracle. (2014), “Oracle Configurator”, available at: http://www.oracle.com/us/products/applications/ebusiness/scm/051314.html.

Orsvärn, K. and Bennick, M.H. (2014), “Tacton: Use of Tacton Configurator at FLSmidth”, in Felfernig, A., Hotz, L., Bagley, C. and Tiihonen, J. (Eds.),*Knowledge-based Configuration – From Research to Business Cases*, Morgan Kaufmann Publishers, Waltham, MA, pp. 211–218.

Osborn, A.F. (1963), *Applied Imagination: Principles and procedures of creative problem solving*, *Oxford*, p. 317.

Pahl, G. and Beitz, W. (1996), *Engineering design: a systematic approach*, *Springer*, p. 544.

Prasad, B. (1998), “Designing products for variety and how to manage complexity”, *Journal of Product & Brand Management*, Vol. 7 No. 3, pp. 208–222.

Purcell, A.T. and Gero, J.S. (1996), “Design and other types of fixation”, *Design Studies*, Vol. 17, pp. 363–383.

Salehi, V. and McMahon, C. (2011), “Development and Application of an Integrated Approach for Parametric Associative CAD Design in an Industrial Context”, *Computer-Aided Design and Applications*, Vol. 8 No. 2, pp. 225–236.

Schmidt, L.C. and Cagan, J. (1997), “GGREADA: A graph grammar-based machine design algorithm”, *Research in Engineering Design*, Vol. 9, pp. 195–213.

Sinha, K. and de Weck, O.L. (2013), “A network-based structural complexity metric for engineered complex systems”, *2013 IEEE International Systems Conference (SysCon)*, Ieee, pp. 426–430.

Sosa, M.E., Eppinger, S.D. and Rowles, C.M. (2007), “A Network Approach to Define Modularity of Components in Complex Products”, *Journal of Mechanical Design*, Vol. 129 No. 11, p. 1118.

Speel, P.-H.A., Schreiber, W. and Joolingen, G.B. (2001), “Conceptual models for knowledge-based systems”, *Encyclopaedia of Computer Science and Technology*, Marcel Dekker Inc., New York.

Steward, D. V. (1981), “Design Structure System: A Method for Managing the Design of Complex Systems”, *IEEE Transactions on Engineering Management*, Vol. EM-28, pp. 71–74.

Tacton Systems. (2014), “Tacton”, available at: http://www.tacton.com/products/modeling/.

Tang, D.., Zhu, R.., Dai, S.. and Zhang, G. (2009), “Enhancing Axiomatic Design with Design Structure Matrix”, *Concurrent Engineering*, Vol. 17 No. 2, pp. 129–137.

The Gephi Consortium. (2014), “Gephi”, available at: http://gephi.github.io/.

The Social Media Research Foundation. (2014), “NodeXL Network Graphs”, available at: http://nodexl.codeplex.com/.

Thomas, L.D. (1992), “Functional implications of component commonality in operational systems”, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, pp. 548–551.

Tiihonen, J., Soininen, T., Männistö, T. and Sulonen, R. (1996), “State-of-the-practice in Product Configuration - A survey of 10 cases in the Finish Industry”, *Knowledge Intensive CAD*, pp. 95–114.

Trentin, A., Perin, E. and Forza, C. (2011), “Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence”, *Computers in Industry*, Elsevier B.V., Vol. 62 No. 3, pp. 260–268.

Ulrich, K. (1995), “The role of product architecture in the manufacturing firm”, *Research Policy*, Vol. 24 No. 3, pp. 419–440.

Verhagen, W.J.C., Bermell-Garcia, P., van Dijk, R.E.C. and Curran, R. (2012), “A critical review of Knowledge-Based Engineering: An identification of research challenges”, *Advanced Engineering Informatics*, Elsevier Ltd, Vol. 26 No. 1, pp. 5–15.

Wielinga, B. and Schreiber, G. (1997), “Configuration-design problem solving”, *IEEE Expert-Intelligent Systems and their Applications*, Vol. 12 No. 2, pp. 49–56.

Wyatt, D.F., Wynn, D.C., Jarrett, J.P. and Clarkson, P.J. (2011), “Supporting product architecture design using computational design synthesis with network structure constraints”, *Research in Engineering Design*, Vol. 23 No. 1, pp. 17–52.

XML Working Group. (2010), “XML Standard”.

Yassine, A. and Wissmann, L. (2007), “The implications of product architecture on the firm”, *Systems Engineering*, Vol. 10 No. 2, pp. 118–137.

Ziv-Av, A. and Reich, Y. (2005), “SOS - Subjective objective system for generating optimal product concepts”, *Design Studies*, Vol. 26, pp. 509–533.