Stephan Klingner and Michael Becker

Formal Modelling of Components and Dependencies for Configuring Product-Service-Systems

The increasing entwinement of products and services in combined offers – so called Product-Service Systems (PSS) – leads to various requirements regarding the modelling of those systems. Due to these requirements, software support is needed for customer-specific configuration of complex PSS. To provide such software a formal description of the structure of products and services is required. Furthermore, complex logical interdependencies between products and services need to be described. Based on an existing service modelling notation the following paper develops a holistic notation for PSS. To describe the interdependencies within PSS, dependency rules are formally specified.

1 Introduction

Promoted by a growing economical relevance of services in recent years (Hildebrand and Klostermann 2007), there has been an increasing convergence of products and services. Therefore, the concept of combining products and services in so-called Product-Service Systems (PSS) as a bundled offer (Morelli 2002) has increasingly gained relevance (Knackstedt et al. 2008; Mont 2002; Stille 2003). Drivers are for instance a focus-shift of customers from the purchase of single products or services to the more abstract purchase of solutions respectively functionalities (Baines et al. 2007; Isaksson et al. 2009). Likewise, services are becoming more important as unique distinguishing features compared with the competition (Knackstedt et al. 2008).

Although the concept of PSS already has its place in the scientific discussion, the level of pervasion in industrial applications is limited so far (Baines et al. 2007; Meier and Uhlmann 2012). This is due to a variety of challenges, which are the result of a more holistic and integrating view on products and services. Thus, the interdependencies between products and services in PSS require new approaches regarding the development and design of business processes, since new departments need to be established respectively the collaboration between existing departments needs to be intensified (Mont 2002).

The interdependencies between products and services also increase complexity, since design, management, and composition of PSS is much more sophisticated than the separate handling of products and services (Isaksson et al. 2009; Mont 2002). For example, modifications in the product portfolio may require correspondent modifications in the service portfolio and vice versa. Furthermore, customers also demand individualised PSS offers, so that the challenge of customer-individual configured offers is extended on both products and services.

To be able to keep PSS manageable – despite their high complexity – adequate software support is required (Dietze 2008). Besides processual aspects, a precise description of the interdependencies between products and services is needed (Becker et al. 2008). Therefore, the following paper focuses research questions regarding the configuration and individualisation of PSS. To be able to offer software support for configuration, a complete, all-embracing modelling of PSS is required, incorporating both product and

service model. This aim can be achieved using three different approaches:

1. Using existing modelling approaches of the respective domain and describing interdependencies between these heterogeneous models separately.
2. Modelling products, services, and interdependencies in an all-encompassing model.
3. Reusing existing models of services and products by transformation into a holistic modelling method.

Due to the vast amount of interdependencies between products and services it does not seem reasonable to use the first approach. Furthermore, modelling should be conducted by using a homogeneous modelling environment (Weber et al. 2004). Thus, it is necessary to establish an all-encompassing model. In this paper we propose an approach for modelling PSS based on an existing notation for modelling services. The approach focuses formalising dependencies between products and services and allows for reuse of existing product models.

Therefore, the remainder of this paper is structured as follows. In Sect. 2 we present the existing method for modelling services. The basic concept of this method is component-based modelling. Since the conceptual origin of component-based modelling lies in the industrial domain, the application of this approach for modelling PSS seems feasible. This application is shown in Sect. 3 by modelling a PSS example. Section 4 introduces various interdependencies between products and services found in academic literature. As a basis for a software implementation these dependencies are classified and formalised, with the aim to provide a lightweight description of interdependencies between components. To emphasise the practical applicability of the approach, different methods of describing structure and variability of products and software were analysed regarding the possibility to transform them into the holistic PSS notation. Accordingly, in Sect. 5 we show whether and how the various already established models can be reused. Related work about service and PSS modelling is presented in Sect. 6. Finally, Sect. 7 concludes the paper and gives some directions for future research.

2 Modelling Services

By modelling services, various objectives can be addressed. A service model may support the description of process flows, the allocation of resources, or the creation of customer-specific configurations of service offers. Driven by the increasing economical relevance of customer-specific offers, previous contributions already focused on methods, models, and tools for configuring services (Becker et al. 2011; Böttcher and Fähnrich 2009; Böttcher and Klingner 2011). By also supporting the assignment of so-called key performance indicators (KPI), the evaluation of configurations regarding productivity aspects becomes feasible as well (Böttcher et al. 2011a).

Since customers demand individualised PSS offers, it has to be studied to which extent the presented method for modelling services is applicable for describing PSS. Therefore, subsequently an aggregated overview of the previous work is given.

2.1 Concepts

In this section we introduce and interrelate various concepts necessary for defining the metamodel for describing services. The metamodel was originally developed to cover four dimensions for the description of services, as introduced by Böttcher and Fähnrich (2009). Those dimensions comprise a *component model* for describing the functionality of service elements, a *resource model* to describe the components' resources and their interdependencies, a *product model* to describe hierarchical dependencies between components, and a *process model* to specify the possible order of component execution.

The presented metamodel as of now focuses the description of components and logical dependencies between components. Thus, the resource

model is not within the focus of the following remarks. The presented concepts of the metamodel are formalised using propositional as well as first-order logic. Although this might lead to additional modelling effort at the beginning, using formal logic has two main advantages. On the one hand, a description based on formal logic allows for a precise and non-ambiguous definition of concepts used in the metamodel. On the other hand, creating extensions and adaptations of the metamodel will be simplified. This is especially relevant in the context of this paper, since one aim was the analysis of the extensibility of the metamodel to support PSS.

The concepts for component-based description of services can be summarised as follows:

- *Components* define the functionality of single service steps respectively service elements. The component model is described by the set of all components.
- *Hierarchical and non-hierarchical dependencies* describe structural connections between components as well as temporal and logical dependencies. Using those relations, the aforementioned product and process model can be represented.
- *Cardinalities* allow for a precise description of logical relations between hierarchically directly connected components. In this way, the product model can be enriched with further semantics.
- *Configurations* are the composition of separate, standardised components into customer-specific service offers.

The structured compilation of various service components describes the service portfolio. A portfolio is formally defined using a tuple

$$P = (C, K, E, card, L, T, kpi, kpiV, att, attV, var)$$

with

- C is a finite, non-empty set of *components*,
- K is a finite set of *connectors*,
- $E \subseteq (C \times K) \cup (K \times C) \cup (K \times K)$ is a set of edges establishing an acyclic *configuration graph* representing *hierarchical dependencies* between components,
- $card : K \to \mathcal{P}(\mathbb{N} \times \mathbb{N})$ is a set of *cardinalities* to assign additional semantics to connectors,
- L is a finite set of *logical dependencies*,
- T is a finite set of *temporal dependencies*,
- kpi is a finite set of *key performance indicators* with possible values $kpiV$,
- att is a finite set of *attributes* with possible values $attV$,
- var is a finite set of *external variables*.

In the following description of the metamodel, we focus components and their hierarchic dependencies. Thus, temporal dependencies, KPI, attributes, and external variables will not be addressed in this paper. Logical dependencies are used to integrate products and services. Therefore, they are separately covered in Sect. 4.

2.2 Components

A service component represents a well-defined, limited functionality, which uses and modifies resources. The functionality is provided by precisely defined interfaces. Components represent subprocesses of a complete service (Böttcher and Fähnrich 2009). Analogous to software components, service components should describe logical and functional related activities. Thus, components should have a high cohesion and a low coupling to be used efficiently (Brocke et al. 2010).

The customer-specific configuration of components is provided by assembling a set of components into a new complete service offer (*composition*). To structure an existing portfolio components can be *decomposed* into smaller units. These units can be managed independently. Having a structured set of decomposed services already offers advantages, e.g., in terms of reduced effort, increased reusability, and increased clarity (Böttcher et al. 2011b). The functionality of a component consisting of other subcomponents

is the sum of the functionalities of all composed subcomponents (Böttcher and Klingner 2011). A portfolio is composed of several components represented by the set C.

Using KPI is a widely-used approach for assessing productivity (Diewert and Nakamura 2005). However, due to the nature of services, the measurement of productivity is very complex. For example, as Maroto and Rubalcaba (2008) state increasing productivity might decrease customer perceived service quality. In addition, customers can influence service productivity to a great extent (Ojasalo 2003). As Harmon et al. (2006) state, service variability adds to the complexity of measuring productivity, too. Using the approach presented here, we do not tackle the first two challenges. However, decomposing services in more fine-grained components reduces their complexity. Using these more focused components simplifies identification of relevant KPI. Therefore, we introduce the assignment of KPI to single components using the mapping *KPIValue*.

$$KPIValue : C \times kpi \rightarrow kpiV$$

Hierarchical dependencies between components as shown in the next section allow for the combination of KPI (*aggregation*). In doing so, productivity of more complex components consisting of more fine-grained components can be assessed by aggregating existing KPI. Therefore, the value of a KPI (*kpiV*) assigned to a component might either be a constant number or a calculation referencing KPI of other components. Since productivity considerations are not the focus of this work, the interested reader may find additional details in (Böttcher and Klingner 2011).

2.3 Hierarchical Dependencies

As stated above, decomposing services into more fine-grained components has advantages itself. Previously monolithic and very complex services are split into less complex subparts. The resulting benefits include an efficient way to offer high flexibility regarding customer requirements. Particularly in view of an increasing competitive pressure, defining components is a suitable approach to handle the dichotomy of standardisation and individualisation (Pine 1999; Sundbo 1994). However, defining components alone is not sufficient to describe complex services. Additionally, to allow for configuration of services, it is necessary to define the structure of components.

To define composition of complex services of fine-grained components, hierarchical dependencies between components need to be established. In industrial engineering, using so-called gozinto-graphs (Vazsonyi 1954) is a common approach for representing these dependencies. In software engineering, using feature models (Mendonca et al. 2009) is a widespread approach. Therefore, it seems reasonable to adapt these approaches for service modelling. Opposing to software respectively product components, service components represent parts of processes. Executing corresponding activities results in realising a well-defined functionality (Geum et al. 2011). Therefore, the decomposition of service components in fine-grained subcomponents can be seen as an application of process refinement.

Hierarchical dependencies are represented using a directed, acyclic configuration graph. The set of nodes of the graph is established by the union of components and connectors. An example for such a configuration graph and, thus, for the decomposition of a complex service is depicted in Fig. 1. The left-hand side shows a component *PV Service* representing a portfolio containing services for photovoltaic installations. On the right-hand side, this component is expanded to depict its hierarchical dependencies with other components. In the graphical notation provided by the metamodel, components are represented using rectangles. Hierarchical dependencies in the configuration graph are represented by paths between components, i.e., in the example the subcomponents of PV Service are *Installation*, *Monitoring*, and *Maintenance*.

In addition to components, Fig. 1 contains one connector, too. Being aware of the definition of

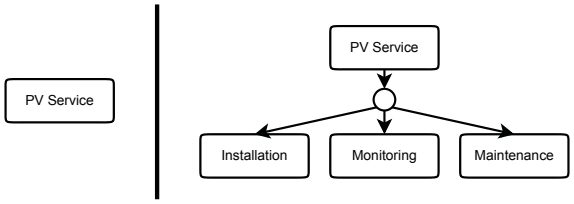

Figure 1: Expanding a complex service

edges in the service portfolios, components must not be connected directly with each other. Instead, they must be connected using connectors. This constraint is necessary to assign semantics to the hierarchical dependencies between components. Additional details about the semantics are presented in the next section.

The configuration graph is restricted by several other constraints besides prohibition of direct connection between components. We mentioned above that it needs to be acyclic. This is necessary because a component must not contain itself. To define additional constraints, we use the two mappings *prenodes* and *postnodes* representing predecessors and successors of a node.

$$prenodes : C \cup K \rightarrow \mathcal{P}(C \cup K)$$
$$\forall v_1 \in C \cup K : prenodes(v_1) =$$
$$\{v_2 \in C \cup K : \exists e \in E : e = (v_2, v_1)\}$$

$$postnodes : C \cup K \rightarrow \mathcal{P}(C \cup K)$$
$$\forall v_1 \in C \cup K : postnodes(v_1) =$$
$$\{v_2 \in C \cup K : \exists e \in E : e = (v_1, v_2)\}$$

First, connectors need to have at least one succeeding node. This is necessary for a reasonable configuration. Leaves of the configuration graph should consist only of components. These components can be seen as atomic services that cannot be detailed any further.

$$\forall k \in K : |postnodes(k)| \geq 1$$

To facilitate traceability of configuration decisions, connectors must have exactly on preceding node.

$$\forall k \in K : |prenodes(k)| = 1$$

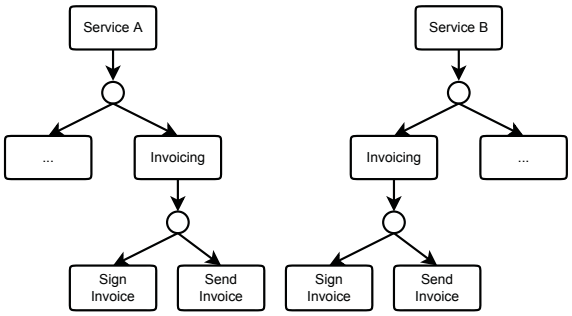

Figure 2: Standardised components without clones

Contrary, this restriction does not hold for components. Decomposing services results in fine-grained components that are candidates for standardisation. To facilitate reuse it should be possible to use standardised components in more than one case. For example, imagine the (highly simplified) case shown in Fig. 2. On the left-hand side the component *Service A* consists of a subcomponent *Invoicing*. Due to several legal regulations, electronic invoicing is a highly standardised service. However, if we were not able to reuse the invoice component in any way, it is necessary to design this component over and over again. This is shown on the right-hand side of Fig. 2. Not only does this result in additional initial effort. Furthermore, changes in the invoicing component (e.g., adding new subcomponents or adapting KPI) need to be propagated everywhere it is used.

To address this problem, we allow components to have more than one incoming edge. This is depicted in Fig. 3. The component Invoicing is a subcomponent of both Service A and Service B. We call components with more than one incoming edge *clones*. A clone facilitates reuse of a standardised component because it needs to be modelled only once. Using a clone simplifies modifications since modifications in the respective component need to be conducted only once. During configuration (see Sect. 2.6) clones are copied throughout the configuration graph, i.e., the example in Fig. 3 has an equal configuration graph to the example in Fig. 2. This is necessary

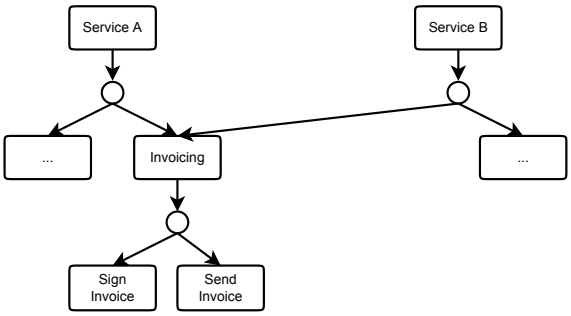

Figure 3: Reuse of standardised components with clones

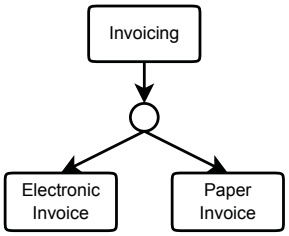

Figure 4: Motivation for cardinalities

to distinguish between the specific components that are selected during configuration.

2.4 Cardinalities

Using simple hierarchical dependencies is not enough to define complex relations between components. For example, it is not possible to define whether succeeding components of a component are optional or mutual exclusive. An example is depicted in Fig. 4. Here, invoicing might either be conducted electronically or paper-based. However, simple hierarchical dependencies do not support distinguishing between different types of composition.

Additional semantics for composition can be assigned using *cardinalities* to connectors. Cardinalities allow for the specification of the number of necessary succeeding nodes selected during configuration. A cardinality is represented using a tuple (min, max). The first value, min, specifies the number of nodes that needs to be selected at least for a configuration to be valid. Furthermore, max specifies the maximum number of nodes that are allowed to be selected during configuration. To ensure satisfiability of cardinalities, several constraints as shown in Tab. 1 restrict their usage.

The constraint `MINMAX` specifies that the minimum number of nodes necessary to select must not be greater than the maximum number of nodes that are allowed to select. Defining a cardinality violating this constraint would result in a configuration graph without any valid configuration. Another constraint to ensure satisfiability of cardinalities is `POSTNODES`. It defines that the maximum number of allowed nodes to select during configuration must not be greater than the amount of succeeding nodes. Together with the `MINMAX` constraint, `POSTNODES` does also restrict the number of minimum nodes necessary to select during configuration.

As can be seen from the definition of the mapping *card*, it is possible to assign an arbitrary number of cardinalities to a single connector. In doing so, cardinalities are connected using logical disjunctions, i.e., only one of the specified cardinalities needs to be satisfied during configuration. To provide clarity, cardinalities of a connector must not overlap. This is ensured by the `OVERLAPS` constraint.

To increase usability of connectors and cardinalities in practice, we provide a set of predefined connectors as shown in Tab. 2. Every of these connectors consists of exactly one cardinality. Connector K_{ALL} specifies that during configuration every succeeding node of the connector needs to be active. When using K_{ONE} exactly one succeeding node needs to be active. Finally, for highest configuration freedom, the K_{ANY} connector can be used. It specifies that an arbitrary number (including none) of succeeding nodes can be active during configuration.

2.5 Extensions

Based on formalised description of components and their interdependencies, further extensions are possible. To fulfil additional requirements

Table 1: Constraints for specifying cardinalities

Name	Formalisation
MINMAX	$\forall k \in K, \forall (m,n) \in card(k) : m \leq n$
POSTNODES	$\forall k \in K, \forall (m,n) \in card(k) : n \leq \lvert postnodes(k) \rvert$
OVERLAPS	$\forall k \in K, \forall (m,n) \in card(k) : \nexists (m^\star, n^\star) \in card(k) : m \leq m^\star \leq n \wedge m \leq n^\star \leq n$

Table 2: Predefined connectors

Name	Formalisation
K_{ALL}	$\forall k \in K_{ALL} : card(k) = \{(\lvert postnodes(k) \rvert, \lvert postnodes(k) \rvert)\}$
K_{ONE}	$\forall k \in K_{ONE} : card(k) = \{(1,1)\}$
K_{ANY}	$\forall k \in K_{ANY} : card(k) = \{(0, \lvert postnodes(k) \rvert)\}$

gathered from practice, the model was extended with *attributes* and *variables* (Becker et al. 2011). Variables represent certain aspects of the service environment. They describe characteristics which may influence the validity or efficiency of customer-specific configurations but are not part of the service model itself. For instance, certain service components (or their characteristics) of a call centre service might depend on the expected number of incoming calls per day. Those characteristics are customer-specific and must be adapted for each customer. By using variables, individual characteristics can be edited independently from the model. Values need to be set only during configuration. Thus, contrary to pre-defined customisation points within the model, the flexible integration of variables allows for a higher abstraction regarding the modelled environment respectively domain.

Further characteristics of components can be described using non-functional properties. For example, a call centre has a limited capacity of processed calls per hour. Therefore, non-functional properties have great impact on the selection of components. Non-functional properties are represented as attributes of components.

More extensions, e.g., for describing resource consumption of components, are possible. Since they are defined based on a formal structure, they can be integrated easily into the existing metamodel.

2.6 Configuration

Specifying portfolios using the above defined concepts is the basis for establishing customer-individual configurations of services. The configuration is carried out to generate service variants best matching customer requirements. Due to the formalisation of the portfolio, it is possible to evaluate configurations regarding their validity. Furthermore, by aggregating KPI, different valid configurations can be compared based on their productivity.

During configuration, components are selected according to the constraints introduced by the portfolio definition. A configuration is established by the set of selected components. Hitherto, we do only consider hierarchical dependencies, i.e., the configuration needs to be valid regarding connectors. To evaluate the validity, we use the set of *activated* nodes. A component is activated when it is selected and vice versa.

$$\forall c \in C : c \in selected \leftrightarrow c \in activated$$

It is possible to establish configurations using a top-down approach, a bottom-up approach, or a combination of both approaches. Top-down approaches start with the topmost component. Accordingly, succeeding components are selected to fulfil cardinalities of the connectors. Therefore, top-down configurations refine components. To support top-down configurations, succeeding

connectors of an active component need to be activated automatically.

$$\forall c \in C, \forall k \in \mathit{postnodes}(c) : \\ c \in \mathit{activated} \rightarrow k \in \mathit{activated}$$

On the other hand, bottom-up configurations start with selecting specific services, i.e., selecting leaf components in the configuration graph. Thus, customers having detailed requirements are able to select services they need and the overall configuration is established by application of dependencies between components. We support this type of configuration by activating nodes with at least one activated direct succeeding node.

$$\forall n \in C \cup K : \\ \exists n^{\star} \in \mathit{postnodes}(n) : n^{\star} \in \mathit{activated} \rightarrow \\ n \in \mathit{activated}$$

Figure 5 depicts a top-down configuration example. The left-hand side shows a situation where no component is selected, i.e., before the beginning of the configuration. In the centre of the figure, the topmost component *Service B* is selected. Consequently, the succeeding connector is selected, too. To establish a valid configuration, the connector needs to be satisfied according to its defined cardinalities. This may be achieved by selecting component *Invoicing* as shown on the right-hand side in Fig. 5. By selecting this component, again the succeeding connector is activated and has to be satisfied by selecting the correct combination of *Sign Invoice* and *Send Invoice*.

A connector is valid if one of its cardinalities is satisfied, i.e., the amount of activated succeeding nodes must be tested against the cardinalities.

$$\forall k \in K \cap \mathit{activated} : \exists (m, n) \in \mathit{card}(k) : \\ m \leq |\mathit{postnodes}(k) \cap \mathit{activated}| \leq n$$

3 Modelling PSS

Heretofore, the presented metamodel is used to model complex services. However, due to its origins in industrial and software engineering, it is feasible to use the metamodel for representing products, too. Especially in industrial engineering, modularisation based on components is a widespread approach. Therefore, we model products as we model services presented above. However, it is necessary to keep in mind that decomposition of services means process refinement while decomposition of products means decomposing physical product parts (e.g., screws, barrels etc.). With this approach, it is possible to manage complex products using the configuration graph as shown above. Furthermore, the same constraints for validity apply.

A complex PSS consists of a product portfolio, a service portfolio, and integration of these two parts (Becker et al. 2008). To evaluate adaptability of our approach we next give examples for both portfolio types situated in the area of a photovoltaic (PV) installation. Building on this, Sect. 4 gives some insights about the integration of both portfolios by specifying dependencies.

Above, we defined a service portfolio as a tuple. Analogously, product portfolios are represented using the same definition. However, the set of temporal dependencies between product components is always empty since temporal dependencies only make sense in process terms. To define a complete PSS we use a tuple $P = (P, S, D)$ with the product and service portfolios P and S, respectively and the set D of dependencies between these portfolios.

3.1 A Product Example

The product portfolio of the PV installation consists of several physical parts. In our example, an installation has four *PV Modules*. A real system would allow for different types of panels, e.g., by giving a selection between various panel materials or sizes. However, for comprehensibility reasons we do not elaborate on that. A panel

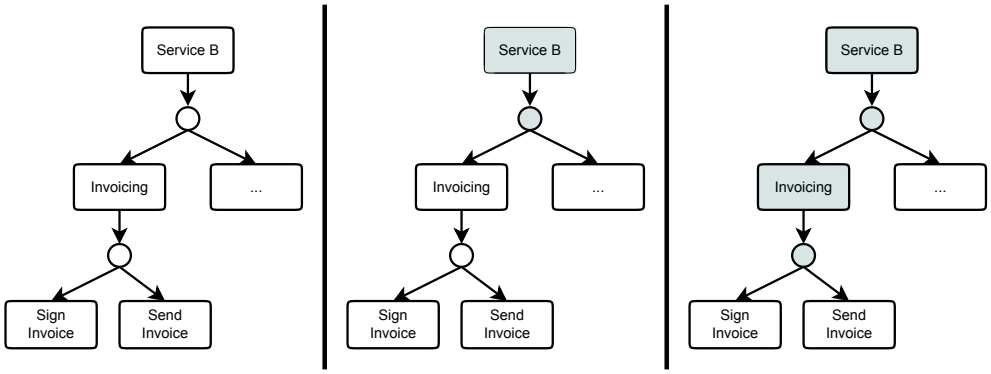

Figure 5: Automated activation of nodes during configuration

has several cells that convert light energy into electrical energy.

Since panels produce direct current and electrical grids use alternating current a *PV Inverter* is necessary for converting. Furthermore, PV inverters provide special functionality like maximum power point tracking. In our example, customers can optionally install a *LAN Interface* and an *RS485 Interface* for monitoring the performance of the installation.

Besides necessary technical components, the provided PV installation also needs to be attached to a roof. This is achieved by using the component *Substructure*. According to the roof characteristics, different substructures are necessary. For example, installations on flat roofs have different requirements than installations on pitched roofs. Each substructure has eight *Roof Hooks*. Hooks are used for mounting panels to the roof. They are an essential and very important part, since hooks support the whole installation. Finally, four *Mounting Profiles* are necessary to attach PV panels. In our example, we only use one profile. However, in reality profiles with different characteristics are available, e.g., different materials (usually steel or aluminium) or different shapes. Figure 6 is a graphical representation of the product portfolio.

Modelling the product portfolio makes use of clones as described above. Thus, we can assure that the components are equal and modifications need to be done only once. For comprehensibility reasons, we depict clones as overlapping nodes. Besides using clones it would also be possible to annotate components that are necessary multiple times. For example, the component roof hook could be annotated with an amount of eight. However, using clones allows for greater flexibility, since a cloned component has a real world counterpart while annotated components do not. Using the definition from above the product portfolio P can be formalised as follows. For the sake of brevity we use component identifiers rather than full component names. Furthermore, since we are focusing dependencies between products and services and not performance evaluation, we omit KPI and attributes.

$$
\begin{aligned}
P =& (C_P, K_P, E_P, card_P, L_P, T_P)\\
C_P =& \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}\\
K_P =& \{K_{1_P}, K_{2_P}, K_{3_P}\}\\
E_P =& \{(P_1, K_{1_P}), (K_{1_P}, P_2), (K_{1_P}, P_3), (K_{1_P}, P_3),\\
& (K_{1_P}, P_3), (K_{1_P}, P_3), (K_{1_P}, P_4), (P_2, K_{2_P}),\\
& (P_4, K_{3_P}), (K_{2_P}, P_5), (K_{2_P}, P_5), (K_{2_P}, P_5),\\
& (K_{2_P}, P_5), (K_{2_P}, P_5), (K_{2_P}, P_5), (K_{2_P}, P_5),\\
& (K_{2_P}, P_5), (K_{2_P}, P_6), (K_{2_P}, P_6), (K_{2_P}, P_6),\\
& (K_{2_P}, P_6), (K_{3_P}, P_7), (K_{3_P}, P_8)\}\\
card_P =& \{(K_{1_P}, (6,6)), (K_{2_P}, (12,12)),\\
& (K_{3_P}, (0,2))\}\\
L_P =& \emptyset, T_P = \emptyset
\end{aligned}
$$

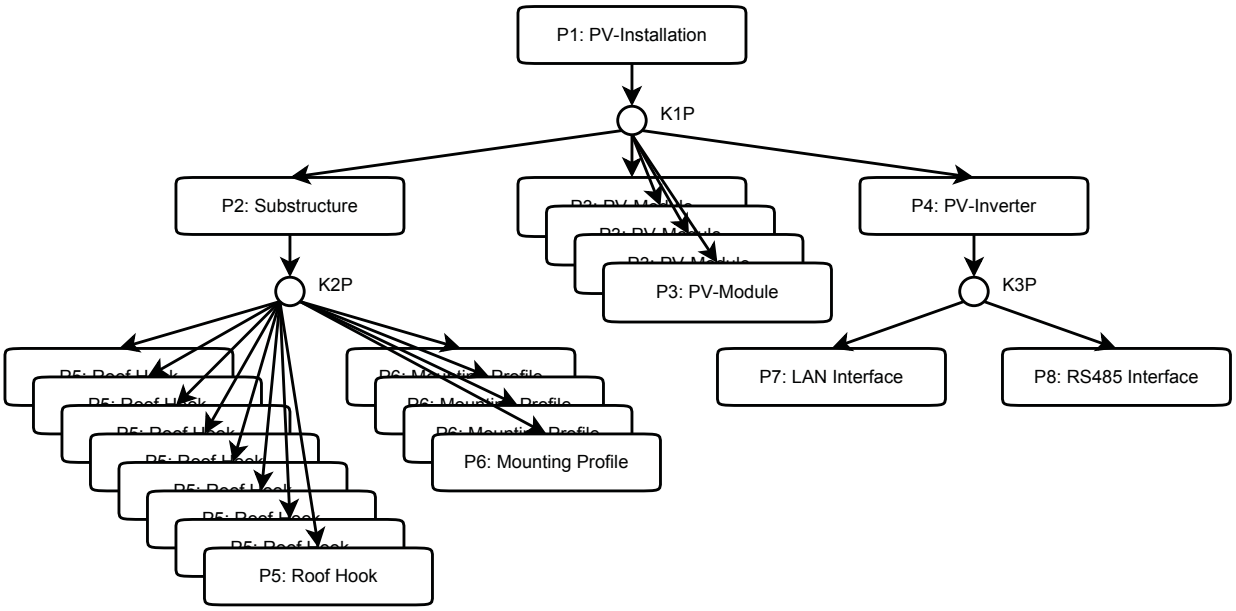

Figure 6: Product portfolio of the PSS PV Installation

3.2 A Service Example

The service portfolio for the PV installation can be distinguished between installations at customer and at provider site. For installations at customer site the provider offers services for *installation*, *maintenance*, and *monitoring*. Installation includes *delivery* and *construction*, i.e., either customers or service technicians from the provider construct the PV installation.

Maintenance has several child components that can be selected. In *on-site maintenance* service technicians conduct a visual inspection of the installation. This is necessary to ensure security of the installation and to provide its efficiency. Furthermore, the installation is analysed for shading effects and the proper function of the inverter is verified. Since PV installations are exposed to wind and weather, *cleaning* of the installation on a regular basis is necessary. Otherwise, the degree of efficiency might decrease due to dirt on the modules. To increase maintenance efficiency, customers can select *remote maintenance*. In doing so, the provider has the ability to conduct several services without sending technicians. Accordingly, this should result in reduced maintenance costs.

Besides maintenance, monitoring is an important activity to ensure installation efficiency. First of all, *recording* data of the installation allows for analysing the long term behaviour. Defining critical thresholds that must not be exceeded allows for sending a signal in case of unexpected behaviour of the installation. Based on the recorded data, it is possible to conduct an *evaluation*. For example, changing the inclination angle of a PV installation might increase its efficiency. Availability of recorded data also allows for the possibility of an *efficiency comparison* with other PV installations. Therefore, the performance of an installation can be compared resulting in gaining additional information.

If it is not possible to construct a PV installation at the customer site, the provider offers the possibility for an investment in off-site installations. In doing so, customers receive the energy they invested for. Though different investment models are possible, we do not go into detail. Since the investment is a completely different offer, customers can only select between on-site or off-site

installations. Thus, it is not possible to combine the component *external installation* with other components from the service portfolio.

We can describe the service portfolio S (as seen in Fig. 7) using the above presented formalisations as follows.

$$
\begin{aligned}
S &= \{C_S, K_S, E_S, card_s, L_S, T_S\} \\
C_S &= \{S_1, S_2, S_3, S_4, S_5, S_6, \\
&\quad S_7, S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}\} \\
K_S &= \{K_{1_S}, K_{2_S}, K_{3_S}, K_{4_S}, K_{5_S}\} \\
E_S &= \{(S_1, K_{1_S}), (K_{1_S}, K_{2_S}), (K_{1_S}, S_{13}), \\
&\quad (K_{2_S}, S_2), (S_2, K_{3_S}), (K_{3_S}, S_5), (K_{3_S}, S_6), \\
&\quad (K_{2_S}, S_3), (S_3, K_{4_S}), (K_{4_S}, S_7), (K_{4_S}, S_8), \\
&\quad (K_{4_S}, S_9), (K_{2_S}, S_4), (S_4, K_{5_S}), (K_{5_S}, S_{10}), \\
&\quad (K_{5_S}, S_{11}), (K_{5_S}, S_{12})\} \\
card_S &= \{(K_{S_1}, (1,1)), (K_{S_2}, (1,3)), (K_{S_3}, (1,2)), \\
&\quad (K_{S_4}, (0,3)), (K_{S_5}, (0,3))\} \\
L_S &= \emptyset, T_S = \emptyset
\end{aligned}
$$

4 Dependencies in PSS

An important requirement regarding the notation for modelling PSS is the representation of interdependencies between elements. Thus, this section elaborates the structure and types of dependencies. Based on that, specific dependency rules presented in literature are introduced. As extension of the existing literature, the rules are aggregated and formalised. Due to formalisation the (semi)automated evaluation of configurations' validity is possible. Thus, each dependency rule is described both textual (supported by different examples) and formal (corresponding to the introduced metamodel).

In PSS different types of dependency relationships are possible. According to Böttcher and Klingner (2011), *descendent dependencies* (Type I) describe hierarchical relationships within a single configuration graph. Since Böttcher focuses only the description of relationships within service models, we suggest an adapted definition of the term *cross-tree dependencies* for the description of non-hierarchical dependencies in PSS.

Within the context of PSS non-hierarchical dependencies can be divided into two different types. Relationships within product or service portfolios can be referred to as intra-tree dependencies (Type II). These dependencies are represented in the portfolios as the set of logical dependencies L_P and L_S respectively. Contrary, relationships between the service and the product portfolio are described as inter-tree dependencies (Type III). Thus, descendent as well as intra-tree dependencies describe relationships between elements of one portfolio while inter-tree dependencies always include both product and service portfolios. Figure 8 provides an overview of the different types of dependencies. In the following, we describe dependencies according to the PSS PV installation of Sect. 3. All presented dependencies are of type III, i.e., they include product and service elements. However, they are valid for dependencies of type I and II as well.

Furthermore, dependencies can be classified regarding their configuration impact. Existing dependency rules in literature are mainly limited to the aspect of restriction, defined by so called constraints (Faltings and Weigel 1994; Gelle and Weigel 1996; Jinsong et al. 2005). The complex dependencies between elements of PSS suggest a wider understanding of rules. This includes the introduction of new rules, such as recommendations (Te'eni and Shufer 2006) or alternatives (Baines et al. 2007). Thus, it is possible to identify three distinct classes of rules.

- *Suggesting* rules describe loose dependencies between PSS elements. Since the application of those rules is optional, the number of valid configuration options is not limited.
- *Restricting* rules describe dependencies, which reduce the number of valid configurations.
- *Modifying* rules describe changes of PSS elements.

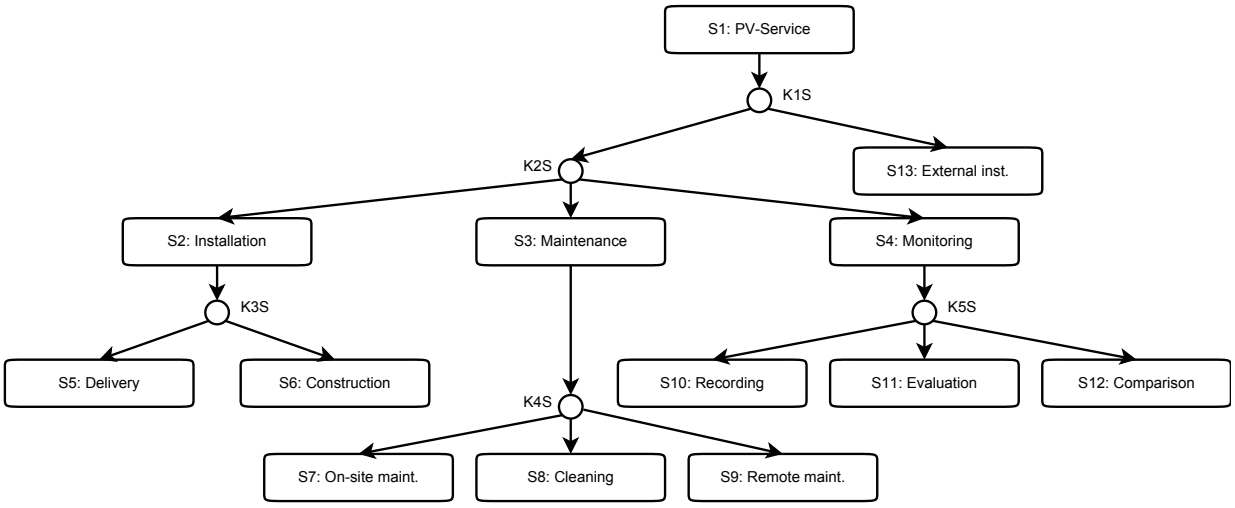

Figure 7: Service portfolio of the PSS PV Installation

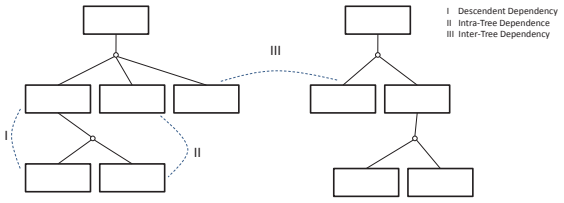

Figure 8: Types of dependencies of PSS

During configuration several rules may have interrelations with each other. To ensure satisfiability, it is necessary to define these relations. For example, elements requiring each other must not prohibit each other, too. For this reason, we formalised the rules as defined in (Becker and Klingner 2012a). For practical application, we developed a Prolog listing adhering to this formalisation.[1] This allows for automated validation of configurations.

Using this formalisation, it is possible to establish an order for evaluating rules. First, restricting rules are validated in background during configuration, i.e., combining elements that are restricted is not possible. Furthermore, modifying rules are applied. Contrary to restricting and modifying rules, suggesting rules are not applied automatically. This is due to the fact that their application is optional. Therefore, these rules are proposed and customers can decide whether or not they apply them.

[1]The listing and its application for the PV example are to be found at `http://sourceforge.net/projects/kpstools/`

4.1 Rule Structure

Though the rules cover a variety of application areas, they all adhere to a specific structure. In the next section we present a selection of rules. However, at this point their general structure is introduced. Based on this structure, it is possible to define additional rules. These rules might be necessary in several areas of application. In doing so, defining new rules allows for establishing a domain specific set of notational elements and, accordingly, simplified modelling. In general, a rule is a mapping from a domain D to a codomain C:

$$rule : D \rightarrow C$$

The domain of the rules is established by a component or a comparison of variables, attributes, or KPI. Furthermore, elements can be combined by a logic operation. Using the comparison, it is possible to compare the respective elements according to a previously defined value. Therefore, the set $comp = \{<, >=\}$ with the respective

meanings of the operators is used. Thus, a rule states that a variable, attribute, or KPI needs to be less than, greater than, or equal to a specific number. The following equations for the formalisation of comparators show that the values of attributes and KPI are compared with respect to a specific component.

$$\begin{aligned} varComp &= (var, comparator, \mathbb{R}) \\ attComp &= (att, C, comparator, \mathbb{R}) \\ kpiComp &= (kpi, C, comparator, \mathbb{R}) \end{aligned}$$

Using these definitions, it is possible to formalise the domain of rules.

$$D = \{C, varComp, attComp, kpiComp\}$$

To be able to also use non-atomic domains, it is necessary to extend and combine the elements of the domain. Logic operations are used for an inductive extension. This is necessary for practical application, since often a combination of various elements has impacts on other elements.

$$\begin{aligned} a \in D \wedge b \in D &\rightarrow (a \wedge b) \in D \\ a \in D \wedge b \in D &\rightarrow (a \vee b) \in D \\ a \in D &\rightarrow \neg a \in D \end{aligned}$$

By means of the established definitions it is possible to define several domains. For example, a domain could be as follows. The components *A* and *B* are selected and the variable *incomingCalls* is greater than 10'000 or the KPI *price* of component *C* is greater than 200. This domain can be formalised as follows.

$$\begin{aligned} &A \wedge B \wedge (incomingCalls, >, 10'000) \\ &\vee (price, C, >, 200) \end{aligned}$$

Contrary to the domain, the codomain cannot be defined in advance since it is too heterogeneous. Valid elements of the codomain depend on the semantics of the specific rule. To illustrate this, we establish various rules for the above specified rule classes. A comprehensive overview about rule classes, analysed rules of this work, resulting constraints for the codomain, and academic references is presented in Tab. 3. For a better understanding we explain the rules according to the PV example given above.

4.1.1 Suggesting Rules

In the following, we present the suggesting rules *alternative* and *recommendation*. Alternatives represent functional equivalent components that can be interchanged with each other. Recommendations define elements that supplement a specific configuration.

Alternative

According to Baines et al. (2007) one motivation for providing PSS is to shift the focus from selling products to selling functionalities. This statement is based on the assumption that customers do not care whether their requirements are satisfied by products or by services.

We use the predicate *surrogates* to define alternative components. The domain and codomain of alternatives is restricted to a logic operation of components. This is reasonable considering that this rule is used to state alternative product and service parts satisfying identical customer requirements by providing equal functionalities.

In the PV example, the service *external installation* fulfils the same requirements as the product *PV Installation* - energising customers. This results in the following rule.

$$surrogates(\text{PV-Installation}) = \text{External Installation}$$

Recommendation

Recommendations are based on the assumption that providers offer their customers additional services when selecting specific products. For example, Uhlmann et al. (2008) argue the productivity of a machine does not only depend on its physical characteristics but also on the qualification of employees operating the machine and

Table 3: Rule classes and rules for defining PSS dependencies

Rule Class	Rule	Source	Codomain
Suggesting	Alternative	Baines et al. (2007)	component, logic operation
	Recommendation	Uhlmann et al. (2008)	component, comparators, logic operation
Restricting	Requirement	Böhmann and Krcmar (2007); Sun (2010)	component, comparators, logic operation
	Prohibition	Faltings and Weigel (1994); Gelle and Weigel (1996); Jinsong et al. (2005)	component, comparators, logic operation
Modifying	Modification	Mont (2002)	KPI modifier

on its general maintenance state. Recommending specific components can furthermore be generalised by recommending components with particular characteristics. Thus, it is possible to leave out too fine-grained details. Finally, the environment of a service might influence the selection of components. Based on these dependencies, organisations are able to compile innovative portfolios. Consequently, by recommending value-adding services or products it is possible to exceed customer requirements (Chase and Hayes 1991). We use the predicate *recommends* to represent a recommendation rule.

In our PV example, the service component *Cleaning* is always recommended when customers select the product component *PV Installation*. This guarantees constant performance of the installation. Selecting the service component *On-site maintenance* recommends the product component *RS485* because it allows for fast and stable readout of installation data by service technicians. Additionally, it is recommended that customers should not construct the installation on their own if the roof pitch is less than 20 or more than 50 degrees, i.e., in these cases the service component *Construction* should be selected and performed by specialised service technicians. These rules can be formalised as follows.

$$\textit{recommends}(\text{PV-Installation}) = \text{Cleaning}$$
$$\textit{recommends}(\text{On-site maintenance}) = \text{RS485}$$
$$\textit{recommends}((\text{roofPitch}, <, 20) \vee (\text{roofPitch}, >, 50)) = \text{Construction}$$

4.1.2 Restricting Rules

Restricting rules reduce the amount of valid configuration variants. The rules *requirement* and *prohibition* are restricting rules. Requirements define elements or characteristics that are necessary for a specific set of selected elements. By using prohibitions, it is possible to identify elements that must not be included in one configuration.

Requirement

Requirements are one of the most frequently mentioned dependencies in literature. Sun (2010) defines an interface that provides a service as *service carrier*. This might either be a product or a combination of products and different services, e.g., the PSS medical treatment contains the service examination and the product pharmaceuticals. An example for external variables in the domain of a requirement rule is given by Böhmann and Krcmar (2007) with integrating services and products into the value creation process of customers. We represent a requirement using the predicate *requires*.

The PV example has two requirements. First, for selecting the service component *Remote maintenance*, the inverter needs a LAN interface. Otherwise, providers would not be able to access the installation from a distance. Furthermore, the service component *Efficiency comparison* requires an installed base of 100 (installed base is represented by the variable *installedBase*). Thus, it ensures that enough comparison data is available.

$$\begin{aligned} &\textit{requires}(\text{Remote maintenance}) = \text{LAN} \\ &\textit{requires}(\text{Efficiency comparison}) = \\ &(\text{installedBase}, >, 100) \end{aligned}$$

Prohibition

The rule prohibition is used to define elements that must not occur together in a single configuration. This is often the case for alternative elements that cannot be combined with each other (i.e., for mutual exclusive alternatives). It is possible to define prohibitions using the configuration graph, too. This is supported by using K_{ONE} connectors. However, non-hierarchic prohibitions must be defined explicitly using the predicate *prohibits*.

The service component *External installation* in the PV example is mutually exclusive to the product component *PV Installation*. Thus, customers cannot select to construct a PV installation at their site together with the participation in an external installation.

$$\begin{aligned} &\textit{prohibits}(\text{External Installation}) = \\ &\qquad\qquad \text{PV Installation} \end{aligned}$$

4.1.3 Modifying Rules

As the name states, modifying rules modify values of elements. Particular configurations might result in changes of KPI of other components. These modifications can be quantified using the rules.

Modification

Nowadays, production companies shift towards offering of PSS because this allows for value-adding of existing products by offering additional services (Mont 2002). For a detailed definition of this relation it is possible to define on component level which service components increase the value of which product components and vice versa. This value modification can reflect in terms of quality or in terms of price, too. We use the predicate *changedValue* to define generic modifications using a modifier.

The quality of the product component *PV Installation* is increased when the construction is conducted by trained service technicians, i.e., the service component *Construction* is selected. The increased quality is represented by increasing the quality of the product component *Substructure*.

$$\begin{aligned} &\textit{changesValue}(\text{Construction}) = \\ &\qquad (\text{Substructure}, \text{quality}, 1.2) \end{aligned}$$

5 Transformation of Product Models

In industrial and software engineering the modelling and the configuration of products are already well established (Hegge and Wortmann 1991). Therefore, it can be assumed that in companies various models already exist. In this section we analyse two common approaches for modelling regarding the possibility of reusing existing models. Based on these findings, transformations between existing models and our presented PSS metamodel are introduced. A corresponding open source application was implemented.[2] Due to the open project structure, the integration of additional modelling formats is possible.

In the following, we analyse the notations *Bill of Material* (BOM) and *feature modelling*. Both approaches are compared with respect to their applicability for transformation. For a better understanding, we model the product portfolio of the PV example in both notations.

[2]The application is publicly available at `http://sourceforge.net/projects/kpstools/`

5.1 Bill of Material

In industrial engineering using BOM is a common method to represent logical and quantitative product data (Jiao et al. 2000; Zhu et al. 2007). BOM describe product components and their relations (Guoli et al. 2003; Stonebraker 1996). Jiao et al. (2000) specify three mandatory aspects, which need be considered in the model. Based on these requirements, the essential elements of a BOM can be derived:

- *Items* for defining components for structuring a product.
- *Goes-into relationships* defining a connection between parent and child component. This connection is complemented with the number of child components needed to create the parent component.
- *Employment* describing the influence of the area of application. Depending on the environment respectively the position in the product life cycle different variants of the creation of BOM exist.

In this paper we restrict the analysis to standard BOM. For a more detailed description of variants, e.g., modular BOM (MBOM), engineering BOM (EBOM), generic BOM (GBOM) or variant BOM (VBOM) the interested reader may refer to the literature (Hegge and Wortmann 1991; Jiao et al. 2000; Veen and Wortmann 1992; Zhu et al. 2007).

A practical example illustrating the above mentioned elements of a BOM can be found in Fig. 9. Analogue to the previous examples a PV installation is modelled. It is important to note that the depicted BOM represents only one possible configuration. The creation of further configurations would result in various, very similar BOM with a high level of redundancy (Hegge and Wortmann 1991).

To evaluate the ability to transform BOM into the PSS metamodel based on a practical example, we modelled a BOM of the PV installation using the web-based application Arena[3]. Arena produces

[3]http://www.arenasolutions.com

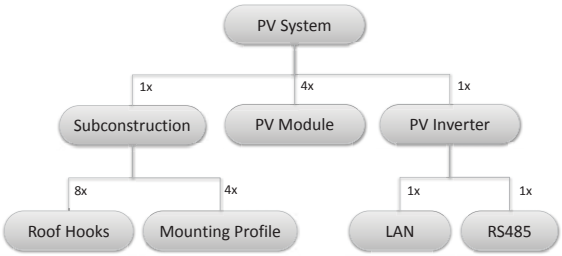

Figure 9: BOM representation of a PV installation

Table 4: Mapping of BOM concepts to concepts of the PSS metamodel

BOM	PSS Metamodel
Item	Component
Parent/Childnodes	ALL-connector
Quantity	Clones

an XML file describing the model which we used as the source for transformation.

The main part of the transformation is the extraction of the items, which are transformed into components. The structure of the BOM's items is mostly identical to the structure of components of the transformed model. Since BOM directly connect product components, we have to add connectors. All added connectors are of type K_{ALL} because every modelled component is mandatory. To represent the BOM-specific quantities the concept of clones are to be used in the metamodel. The limited expressiveness of BOM inhibits the description of complex logical relationships. Therefore, the expressiveness of the PSS metamodel can be mapped only partially, as Tab. 4 indicates.

5.2 Feature Models

Feature models are widely-used in the area of software engineering to describe so-called product lines. Czarnecki and Eisenecker (2000) define the concept of a feature as a function of the system, which is visible for end users and a perceivable characteristic of a concept, which is relevant for certain stakeholders.

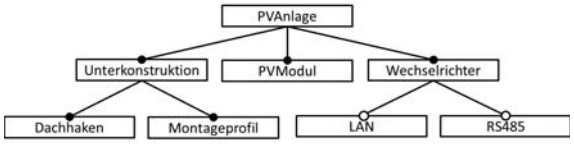

Figure 10: Feature model representation of a PV installation

Similar to the PSS metamodel or the BOM presented above, features are arranged in a hierarchical structure. The following three different types of compositions can be distinguished:

- *Mandatory* features have to be chosen if the respective parent feature is chosen.
- From a set of *alternative* features exactly one has to be chosen to create a valid configuration.
- *Optional* features can be chosen, but do not have to.

Additionally to these hierarchical relationships also non-hierarchical dependencies can be defined. Batory (2005) introduces *simple inclusion* as well as *simple exclusion*. Simple inclusion defines that choosing feature *A* results in an automatic selection of feature *B*. Simple exclusion determines the automatic deactivation of feature *B* as consequence of choosing feature *A*. Those basic dependencies can be extended by using propositional formula. Czarnecki et al. (2004) present further extensions of feature models, such as cardinalities and annotations of features. However, this papers focuses only basic feature models. Figure 10 shows the representation of the product portfolio of the PV installation as feature model. The model was created using the Eclipse plug-in FeatureIDE (Kastner et al. 2009). Mandatory features are depicted by a filled circle, whereas non-filled circles represent optional features.

Similar to BOM, the concepts of feature models can be transformed to elements of the PSS metamodel as shown in Tab. 5. The expressiveness of feature models is comparable to the PSS metamodel, so that transformations in either models can be conducted without losing expressive power. Only cardinalities as represented by connectors cannot be modelled in feature models without using aforementioned extensions. Therefore, the components roof hooks, mounting profile, and PV module are represented by only one feature. Missing data regarding quantities can be added using feature attributes.

Table 5: Mapping of feature model concepts to concepts of the PSS metamodel

Feature Model	PSS Metamodel
Concept	Configured Product
Feature	Component
Relationships	Connectors
Mandatory Features	Connector K_{ALL}
Optional Features	Connector K_{ANY}
Alternative Features	Connector K_{ONE}

6 Related Work

Since our presented metamodel has its origins in the domain of service engineering, in this section we first discuss various approaches for service modelling. This is followed by an overview of different methods for describing PSS. A deep analysis of product modelling approaches is not in the focus of this paper. A detailed discussion of various approaches for modelling products is carried out by Yang et al. (2008).

For describing complex services, Barros et al. (2011) introduced the Unified Service Description Language (USDL). Besides describing service component interaction, the focus of the USDL is the holistic depiction of services including the organisation, responsibilities etc. Contrary to our presented metamodel the pricing and possible discounts for services are explicitly supported. In the PSS metamodel prices could be indirectly represented using KPI. USDL lacks an extensive support for the configuration of services.

In addition to defining dependencies between components, Dong et al. (2011) also focus configuration supported by an ontology. Using the ontology it is possible to define constraints. For

practical application, they provide interfaces to the Java Expert System Shell.

In addition to a structural description of services, Brocke et al. (2010) as well as Emmrich (2005) analyse necessary adaptations of services in regard to different phases of their life cycle. While Emmrich (2005) focuses product centred services as well as the states and state changes of products, Brocke et al. (2010) focus the continuous need for customer-specific adaptation in the area of IT-based services.

Contrary to research about services, PSS in general and modelling PSS in particular did emerge in academic literature only in recent years. Especially in the German-speaking world the term *hybrid service bundles* is commonly used (Becker et al. 2009; Böhmann and Krcmar 2007; Langer et al. 2009). Early definitions of PSS were established by Goedkoop et al. (1999) and Mont (2002). Both works argue for a view on PSS not only consisting of products and services but also of dependencies between these two parts. This statement is used as a foundation for a couple of comparable approaches to model PSS.

Becker et al. (2009) present an approach to describe PSS. They focus ex ante configuration by providers and customer-individual configuration based on Entity-Relationship-Diagrams. A distinctive feature of their approach is assessing different configurations using financial schemes. Therefore, they allow for a fine-grained comparison of established configurations. Contrary to this, our model focuses using KPI dynamically calculated during configuration.

Originally developed for modelling products, Weber et al. (2004) present their so-called Property-Driven Design/Development model. They distinguish between characteristics to define the constituent parts and the structure of a PSS and properties to describe the product behaviour. Additionally, they provide a process model for systematic PSS development. A process model is also described by Uhlmann et al. (2008) where the authors focus consideration of customer requirements. We can reflect parts of both approaches using external variables (as customer requirements) and attributes and KPI (as characteristics). General requirements and specific characteristics of PSS have been established by Morelli (2002).

Further approaches analyse relations between products and services (Sun 2010). However, they focus relations between organisations offering and using PSS. Immediate practical relevance is shown by Walter (2009) with defining a PSS in the domain technical services.

7 Conclusion

In this paper we presented a holistic modelling method for PSS, including product and service components as well as interdependencies between them. Based on the postulation of similarity of structuring products and services, an existing metamodel for describing services was extended for the representation of products. Therefore, various dependency rules were identified, structured, and formally defined. Together with the component-based modelling of the structure these rules define the basis for the configuration of complex PSS. By the use of these models the management of a high number of variants becomes economically feasible, due to economies of scale (Anthonysamy et al. 2011; Böttcher and Klingner 2011).

To ensure the possibility of reusing existing models and to validate the expressiveness of the presented metamodel, approaches for transforming two common product modelling notations to the PSS metamodel were analysed. This was illustrated using an example of a PV installation, which was modelled as BOM respectively feature model.

It was shown that transformation of product models into the PSS metamodel is feasible. On this basis, product modelling can be improved by extending the PSS metamodel with product-specific details. For example, explicit definition of quantities makes indirect and extensive modelling of clones obsolete. This also allows for the

integration of quantitative dependency rules, e.g., *if product component A is chosen more than ten times, service component B has to be chosen.* These rules are mentioned by Jinsong et al. (2005) under the term *cardinality constraints.* Additional necessary extensions for the metamodel can be identified by analysing more product modelling approaches. For example, Yang et al. (2008) present STEP-based methods and Felfernig et al. (2001) use configuration graphs modelled with UML.

The presented PSS metamodel defines the formal basis for the development of software tools supporting the modelling and configuration of PSS. For the subdomain of service modelling a prototypical tool was already developed, based on previous scientific findings and industrial requirements (Klingner et al. 2011). The extension for PSS needs to be implemented in the future. A first step in that direction is the integration of dependency rules within the service model. These rules allow for generation of business process models based on customer-specific configurations (Becker and Klingner 2012b).

Using software tools, the increased complexity of formalised models becomes manageable. Nevertheless, creating models still involves a lot of effort. The lack of a process model for creating models corresponding to the PSS metamodel adds to this fact. Possible negative impacts are a suboptimal granularity of elements of the PSS. This might lead to higher efforts in case of adaptations or to erroneous implications regarding KPI. Particular attention needs to be paid on how to define dependencies between PSS elements. The methodology presented by Abramovici and Schulte (2005) and by Thomas et al. (2008) can be used as a feasible starting point for establishing this process.

The practical applicability of the proposed PSS metamodel can be further increased by defining domain-specific components and common PSS elements in advance. On-demand usage of these elements could decrease the necessary effort for creating new models. Therefore, existing best practices and reference models of organisations can be reused.

An additional requirement from the practice is the representation of changes in the PSS within a specific period of time. For example, imagine a webshop with seasonal fluctuations, e.g., peak load during Christmas time. As of yet, the metamodel does not give any support for this. To be able to comprehensively evaluate behaviour and costs of such a system, the description of these time-variable effects need to be included in the model. Brocke et al. (2010) and Emmrich (2005) present a few approaches in this direction. However, extensive changes are necessary to integrate these into the metamodel.

Of particular interest in complex PSS is the interaction between participating organisations (Sun 2010). Currently, we only support representing PSS elements of a single organisation, since services provided cooperatively by two or more organisations are not in the focus. However, collaboration of organisations can be facilitated by sharing PSS models. For example, product models of a manufacturer can be imported into the service portfolio of service provider. This allows for definition of dependencies between portfolio elements.

Acknowledgements

The German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) partially supported this work in the context of the DLR projects ‘Concept and Implementation of an Information Production System (IPS) for Precision Farming’ (support code 01IS12013B) and EUMONIS (support code 01IS10033D).

References

Abramovici M., Schulte S. (2005) Lifecycle Management von Produkt-Service-Systemen (PSS) für einen maximierten Kundennutzen. In: Grote K.-H. (ed.) Tagungsband 3. Gemeinsames Kolloquium Konstruktionstechnik 2005 am 16. und 17.06.2005 im Herrenkrug Parkhotel Magdeburg. Shaker, pp. 1–11

Anthonysamy P., Rashid A., Rummler A. (2011) Taming Unbounded Variability in Service Engineering. In: Muehlen M., Su J., Aalst W., Mylopoulos J., Rosemann M., Shaw M. J., Szyperski C. (eds.) Business Process Management Workshops. LNBIP Vol. 66. Springer, Berlin, pp. 615–619 http://dx.doi.org/10.1007/978-3-642-20511-8_56

Baines T. S., Lightfoot H. W., Evans S, Neely A, Greenough R, Peppard J, Roy R, Shehab E, Braganza A, Tiwari A, Alcock J. R., Angus J. P., Bastl M, Cousens A, Irving P, Johnson M, Kingston J, Lockett H, Martinez V, Michele P, Tranfield D, Walton I. M., Wilson H (2007) State-of-the-art in product-service systems. In: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 221(10), pp. 1543–1552

Barros A., Kylau U., Oberle D. (2011) Unified Service Description Language 3.0 (USDL) Overview. SAP Research. http://www.internet-of-services.de/index.php?id=570

Batory D. (2005) Feature Models, Grammars, and Propositional Formulas. In: Obbink H., Pohl K. (eds.) Software Product Lines. LNCS Vol. 3714. Springer, Berlin, pp. 7–20

Becker J., Beverungen D., Knackstedt R. (2008) Reference Models and Modeling Languages for Product-Service Systems Status-Quo and Perspectives for Further Research. In: Hawaii International Conference on System Sciences, pp. 105–105

Becker J., Beverungen D., Knackstedt R., Müller O. (2009) Konzeption einer Modellierungssprache zur softwarewerkzeugunterstützten Modellierung, Konfiguration und Bewertung hybrider Leistungsbündel. In: Thomas O., Nüttgens M. (eds.) Dienstleistungsmodellierung. Physica, Heidelberg, pp. 53–70

Becker M., Klingner S. (2012a) Formalisierung von Regeln zur Darstellung von Abhängigkeiten zwischen Elementen von Product-Service-Systems. Abteilung für Betriebliche Informationssysteme, Universität Leipzig. Leipzig, Germany. http://koproserv.uni-leipzig.de/wp-content/uploads/2012/02/report.pdf

Becker M., Klingner S. (2012b) Towards Customer-Individual Configurations of Business Process Models. In: Aalst W., Mylopoulos J., Rosemann M., Shaw M. J., Szyperski C., Bider I., Halpin T., Krogstie J., Nurcan S., Proper E., Schmidt R., Soffer P., Wrycza S. (eds.) Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, Berlin and Heidelberg, pp. 121–135

Becker M., Klingner S., Böttcher M. (2011) Configuring services regarding service environment and productivity indicators. In: Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on, pp. 505–512

Böhmann T., Krcmar H. (2007) Hybride Produkte: Merkmale und Herausforderungen. In: Bruhn M., Stauss B. (eds.) Wertschöpfungsprozesse bei Dienstleistungen. Gabler, Wiesbaden, pp. 239–255

Böttcher M., Fähnrich K.-P. (2009) Service Systems Modeling. In: Alt R., Fähnrich K.-P., Franczyk B (eds.) Proceedings First International Symposium on Services Science. Logos, Leipzig

Böttcher M., Klingner S. (2011) Komponentisierung zur Steigerung der Dienstleistungsproduktivität. In: Bruhn M., Hadwich K. (eds.) Dienstleistungsproduktivität. Gabler, pp. 59–80

Böttcher M., Klingner S. (2011) Providing a Method for Composing Modular B2B-Services. In: Journal of Business & Industrial Marketing 26(5), pp. 320–331

Böttcher M., Klingner S., Becker M. (2011a) Komponentenbasiertes Produktivitätscontrolling komplexer Dienstleistungsportfolios. In: Controlling 23(10), pp. 509–513

Böttcher M., Swialkowski R., Fähnrich K.-P. (2011b) Produktivitätsbetrachtung bei der Komponentisierung von Dienstleistungen. In: Gatermann I., Fleck M. (eds.) Mit Dienstleistungen die Zukunft gestalten – Impulse aus Forschung und Praxis. Campus, pp. 207–216

Brocke H., Uebernickel F., Brenner W. (2010)

Zwischen Kundenindividualität und Standardisierung – Konzept und Referenz-Datenstruktur eines konfigurierbaren IT-Produktmodells. In: Thomas O., Nüttgens M. (eds.) Dienstleistungsmodellierung 2010. Physica, Heidelberg, pp. 231–253

Chase R. B., Hayes R. H. (1991) Beefing Up Operations in Service Firms. In: Sloan Management Review 33(1), pp. 15–26

Czarnecki K., Eisenecker U. W. (2000) Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA

Czarnecki K., Helsen S., Eisenecker U. (2004) Staged Configuration Using Feature Models. In: Nord R. (ed.) Software Product Lines. LNCS Vol. 3154. Springer, Berlin, pp. 162–164

Dietze V. (2008) Festigung zwischenbetrieblicher Kollaboration durch den Einsatz von Group Decision Support Software – ein strategischer Planungsansatz. Grin, München

Diewert W. E., Nakamura A. O. (2005) Concepts and Measures of Productivity: An Introduction In: Lipsey R., Nakamura A. (eds.) University of Calgary Press chap. 2, p. 20

Dong M., Yang D., Su L. (2011) Ontology-based service product configuration system modeling and development. In: Expert Systems with Applications 38(9), pp. 11770 –11786

Emmrich A. (2005) Ein Beitrag zur systematischen Entwicklung produktorientierter Dienstleistungen. PhD thesis, University of Paderborn, Paderborn

Faltings B., Weigel R. (1994) Constraint-based knowledge representation for configuration systems. Technical Report TR-94/59. Department d'Informatique, Laboratoire d'Intelligence Artificielle, Ecole Polytechnique Federale de Lausanne. Lausanne

Felfernig A., Friedrich G., Jannach D. (2001) Conceptual modeling for configuration of mass-customizable products. In: Artificial Intelligence in Engineering 15(2), pp. 165–176

Gelle E., Weigel R. (1996) Interactive Configuration using Constraint Satisfaction Techniques. In: In Second International Conference on Practical Application of Constraint Technology, PACT-96. Menlo Park, AAAI Press, London, pp. 37–44

Geum Y., Kwak R., Park Y. (2011) Modularizing services: A modified HoQ approach. In: Computers & Industrial Engineering 62(2), pp. 579 –590

Goedkoop M., van Halen C., Riele T. H., Rommens P. (1999) Product Service systems, Ecological and Economic Basics. Pre. The Hague

Guoli J., Daxin G., Tsui F. (2003) Analysis and implementation of the BOM of a tree-type structure in MRPII. In: Journal of Materials Processing Technology 139(1-3), pp. 535–538

Harmon E., Hensel S. C., Lukes T. E. (2006) Measuring Performance in Services. In: The McKinsey Quarterly

Hegge H., Wortmann J. (1991) Generic bill-of-material: a new product model. In: International Journal of Production Economics 23(1-3), pp. 117–128

Hildebrand W.-C., Klostermann T. (2007) Dienstleistungsverkehr in industriellen Wertschöpfungsprozessen. In: Bruhn M., Stauss B. (eds.) Wertschöpfungsprozesse bei Dienstleistungen. Gabler, Wiesbaden, pp. 215–236

Isaksson O., Larsson T. C., Rönnbäck A. Ö. (2009) Development of product-service systems: challenges and opportunities for the manufacturing firm. In: Journal of Engineering Design 20(4), pp. 329–348

Jiao J., Tseng M. M., Qinhai Ma, Yi Zou (2000) Generic Bill-of-Materials-and-Operations for High-Variety Production Management. In: Concurrent Engineering 8(4), pp. 297–321

Jinsong Z., Qifu W., Li W., Yifang Z. (2005) Configuration-oriented product modelling and knowledge management for made-to-order manufacturing enterprises. In: The International Journal of Advanced Manufacturing Technology 25(1-2), pp. 41–52

Kastner C., Thum T., Saake G., Feigenspan J., Leich T., Wielgorz F., Apel S. (2009) FeatureIDE: A tool framework for feature-oriented software development. In: Proceedings of the 31st International Conference on

Software Engineering. ICSE '09. IEEE Computer Society, Washington DC, pp. 611–614

Klingner S., Böttcher M., Becker M., Döhler A. (2011) Managing complex service portfolios. In: Ganz W., Kicherer F., Schletz A. (eds.) RESER 2011 Productivity of Services NextGen – Beyond Output/Input. Conference Proceedings.

Knackstedt R., Pöppelbuß J., Winkelmann A. (2008) Integration von Sach- und Dienstleistungen – Ausgewählte Internetquellen zur hybriden Wertschöpfung. In: WIRTSCHAFTSINFORMATIK 50 (3), pp. 235–247

Langer S., Kreimeyer M., Müller P., Lindemann U., Blessing L. (2009) Entwicklungsprozesse hybrider Leistungsbündel – Evaluierung von Modellierungsmethoden unter Berücksichtigung zyklischer Einflussfaktoren. In: Thomas O., Nüttgens M. (eds.) Dienstleistungsmodellierung. Physica, Heidelberg, pp. 71–87

Maroto A., Rubalcaba L. (2008) Services productivity revisited. In: The Service Industries Journal 28(3), pp. 337–353

Meier H., Uhlmann E. (2012) Hybride Leistungsbündel – ein neues Produktverständnis. In: Meier H., Uhlmann E. (eds.) Integrierte Industrielle Sach- und Dienstleistungen. Springer, Berlin, pp. 1–21 http://dx.doi.org/10.1007/978-3-642-25269-3_1

Mendonca M., Wasowski A., Czarnecki K. (2009) SAT-based analysis of feature models is easy. In: Proceedings of the 13th International Software Product Line Conference. SPLC '09. Carnegie Mellon University, Pittsburgh, pp. 231–240

Mont O. K. (2002) Clarifying the concept of product-service system. In: Journal of Cleaner Production 10(3), pp. 237 –245

Morelli N. (2002) Designing Product/Service Systems: A Methodological Exploration English. In: Design Issues 18(3), pp. 3–17

Ojasalo K. (2003) Customer Influence on Service Productivity. In: SAM Advanced Management Journal (07497075) 68(3), pp. 14 –19

Pine B. J. (1999) Mass Customization: The Frontier in Business Competition. Harvard Business School Press

Stille F. (2003) Produktbegleitende Dienstleistungen gewinnen weiter an Bedeutung. In: Wochenbericht 70(21), pp. 335–342

Stonebraker P. W. (1996) Restructuring the bill of material for productivity: A strategic evaluation of product configuration. In: International Journal of Production Economics 45(1-3), pp. 251–260

Sun H. (2010) Product service relationship: defining, modelling and evaluating. In: International Journal of Internet Manufacturing and Services 2(2), pp. 128–141

Sundbo J. (1994) Modulization of service production and a thesis of convergence between service and manufacturing organizations. In: Scandinavian Journal of Management 10(3), pp. 245 –266

Te'eni M., Shufer I. (2006) Component upgrading with dependency conflict resolution, knowledge based and rules pat. 7140013 http://www.freepatentsonline.com/7140013.html

Thomas O., Walter P., Loos P. (2008) Product-Service Systems: Konstruktion und Anwendung einer Entwicklungsmethodik. In: WIRTSCHAFTSINFORMATIK 50(3) (3), pp. 208–219

Uhlmann E., Meier H., Bochnig H., Geisert C., Sadek K., Stelzer C. (2008) Customer-driven development of product-service-systems. In: Pham D. T., Eldukhri E. E., Soroka A. J. (eds.) Innovative production machines and systems: Fourth I*PROMS Virtual International Conference. Whittles Publ.

Vazsonyi A. (1954) The Use of Mathematics in Production and Inventory Control. I. In: Management Science 1(1), pp. 70–85

van Veen E., Wortmann J. C. (1992) New developments in generative BOM processing systems. In: Production Planning & Control 3(3), pp. 327–335

Walter P. (2009) Modellierung technischer Kundendienstprozesse des Maschinen- und Anlagenbaus als Bestandteil hybrider Produkte. In: Thomas O., Nüttgens M. (eds.) Di-

enstleistungsmodellierung. Physica, Heidelberg, pp. 129–145

Weber C., Steinbach M., Botta C., Deubel T. (2004) Modelling of product-service systems (PSS) based on the PDD approach. In: D. M. (ed.) Proceedings of the 8th International Design Conference DESIGN 2004. Dubrovnik, pp. 547–554

Yang W. Z., Xie S. Q., Ai Q. S., Zhou Z. D. (2008) Recent development on product modelling: a review. In: International Journal of Production Research 46(21), pp. 6055–6085

Zhu S., Cheng D., Xue K., Zhang X. (2007) A Unified Bill of Material Based on STEP/XML. In: Shen W., Luo J., Lin Z., Barthès J.-P. A., Hao Q. (eds.) Computer Supported Cooperative Work in Design III. Springer, Berlin, pp. 267–276

Stephan Klingner, **Michael Becker**

Department of Business Information Systems,
University Leipzig
Augustusplatz 10
04109 Leipzig,
Germany
{klingner | mbecker}@informatik.uni-leipzig.de