Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 505–512

ISBN 978-83-60810-22-4

Configuring services regarding service environment and productivity indicators

Michael Becker, Stephan Klingner, Martin Böttcher
Department of Business Information Systems
Leipzig University, Germany
Email: {mbecker|klingner|boettcher}@informatik.uni-leipzig.de

***Abstract*—In course of the extensive changes in the service sector, methods and tools for modelling services, managing service-portfolios and optimising service-offers are required. This paper proposes an extension of a basic metamodel as described in various previous publications to be able to describe non-functional properties, global variables and the definition of configuration constraints.**

I. Introduction

DUE TO the widely acknowledged shift from the second to the third economic sector, services are of increasing importance. A growing market, driven, inter alia, by internationalisation, results in higher volumes of delivered services. At the same time complexity of provided services is increasing. This change in the quantitative as well as the qualitative dimension leads to the need for concepts, methods and models to conduct the engineering of services in a well-structured way [17]. Furthermore, both aspects in combination with a growing competitive environment require a stronger focus on productivity aspects of services.

This paper mainly presents an extension of various foundational concepts of a holistic metamodel for modelling, evaluating and optimising services, as described in multiple publications, such as [14]–[16]. Section II gives an aggregated overview of the different aspects of the metamodel described in the papers. Using a prototype, the scientific results were evaluated during several workshops. Subsequently this basic model was extended in accordance to the feedback of our industry partners. The extension comprises concepts that are required in real world applications but were not implemented in our service model so far. These include non-functional properties (*attributes*), global, system-wide definitions (*variables*) as well as the definition of rules as part of configuration queries (*constraints*).

In this work we present formalisations of these concepts and show their integration into the existing metamodel. Furthermore, we extend our metamodel to the ability of querying service components. To do so the remainder of this paper is structured as follows. Our method describes a two-step procedure, which consists firstly of modelling the service respectively the service portfolio (section II) and secondly of configuring the customised service offers based on that previously modelled portfolio (section III). These sections will recall necessary concepts for the formalisation of service components and extend our existing metamodel with attributes, variables, and constraints. In section IV we present initial ideas and applications of querying services. Finally, sections V and VI give an overview about related work in this field and conclude the paper.

II. Modelling Services

As mentioned in the introduction we presented initial ideas [15] and a formal approach [13] for modelling services based on components. Rather than recapitulating all concepts in detail we will briefly describe fundamental concepts that are necessary to understand the new concepts we introduce in this work. They are namely attributes to specify nonfunctional characteristics of components, external variables to detail the service environment, and constraints to refine valid service components during configuration. Using these extensions configurations can adapt customer wishes on specific aspects of components. Furthermore, they facilitate querying services.

In the course of this work we present the new modelling elements within the context of a small example portfolio shown in figure 1. In this example – based on a real world example of one of our industry partners – the described services are the provision of a call centre and the realisation of billing tasks. For this purpose, we define a component *CallCentre* encapsulating the general call centre provision process. This component is detailed by the subcomponents *CallCentre A*, *CallCentre B*, and *CallCentre C* defining specific steps for different call centre contractors. Due to comprehensibility reasons, the component *Billing* is not shown in full detail but only sketched. In the following section we describe the particular modelling concepts.

A. Foundations

To comprehend the newly introduced concepts for service modelling it is necessary to recall some of the existing foundations we lay in past work. In [15] we have defined service components as an offering of a well-defined functionality via precisely described interfaces. Thus, a component represents a part of a service provision process. The relationship between the so-called configuration graph containing the components and its processual representation can be seen in figure 2. This figure shows the general call centre provision process consisting of the common activities *apply call numbers* and *train products* encapsulated in component *CallCentre*. Furthermore, it has three subprocesses *CallCentre A*, *CallCentre B*,

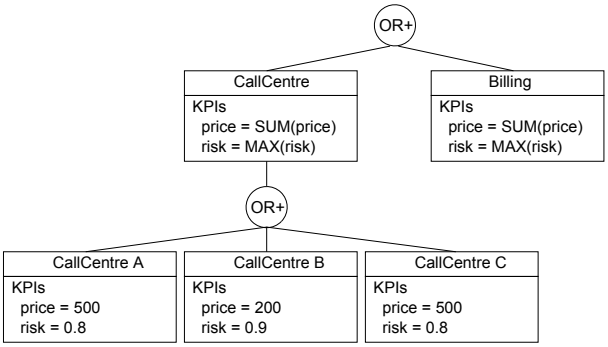

Figure 1. Example portfolio: of call centre provision and billing

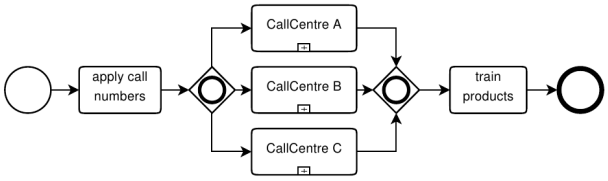

Figure 2. Call centre components represented as BPMN model

and *CallCentre C* containing specific activities for the three options.

Components can be used for composition (by combining components with other components) and decomposition (by separating existing components into distinct components). One application for composition is the creation of costumer specific combinations by choosing components from a portfolio. Decomposition can be used to structure and detail existing services. These two possibilities are mapped to a process model representation using e.g. BPMN subprocesses.

Basically, a portfolio consists of a set of service components. We relate components with each other using connectors. Based on these concepts the definition of hierarchical dependencies between components is possible, e.g. component A consists of components C and B. These dependencies can be extended by cardinalities enabling statements about the required amount of subcomponents. For example, in the call centre use case the specific call centre components are connected using a disjunctive-obligatory connector (OR^+) meaning that at least one of the child components must be selected. Since components represent processes, there are usually temporal dependencies between them. It is possible to integrate these dependencies using linear temporal logic enabling restrictions such as component A must be succeeded by component B. Furthermore, non-hierarchic dependencies between components are integrated using propositional logic. These dependencies are necessary to define additional constraints on service composition that cannot be displayed in the graph hierarchy, e.g. about mutually exclusive components. Additional details about connector semantics and both temporal and logical dependencies are presented in [16].

One goal of composition is to increase the productivity of services. Therefore, it is necessary to measure productivity in

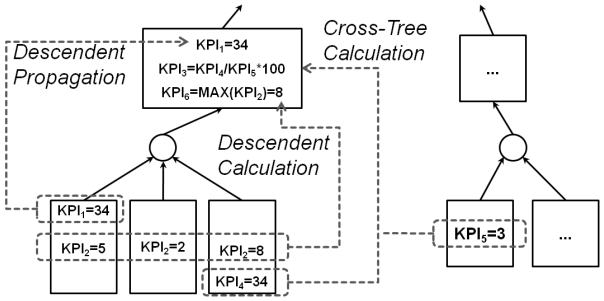

Figure 3. KPI inheritance strategies

some way. This is achieved using key performance indicators (KPIs) in components. In the call centre example we use the price (interpreted as costs to execute a component) and the risk (a lower number indicates a more mature process) as KPIs. Traditionally, productivity is defined as the fraction of output produced to inputs used. This approach is difficult for services since not only the input is hard to calculate but also the output [24]. However, this discussion is not in the focus of this work. In the component model, it is possible to use KPIs based on generic formulae.

KPIs are inherited through the service model in threefold manner. First, *Descendent Propagation* propagates a KPI to the parent component. Thus, the KPI characterises the parent component, too. Second, *Descendent Calculation* enables using KPIs of child components to calculate KPIs in parent components. Thus, in the use case in figure 1 the KPIs price and risk from the specific call centres can be accessed in the general call centre component. It is usually necessary to combine KPIs using arithmetic operators, e.g. the price of the general call centre provision is the sum of the specific prices. Another way to inherit KPIs is using *Cross-Tree Calculation* allowing to calculate KPIs based on KPIs from components that are not in hierarchical dependencies with each other. This allows for the representation of non-hierarchic component combinations on productivity. The different strategies for KPI inheritance are shown in figure 3.

B. External Variables

Service provision cannot be seen in isolation from the environment where services are to be provided. For example, in the call centre use case the amount of expected incoming calls per day plays an important role both for the price of the overall service and the calculation of the risk of the service. Since they depend on customer characteristics and preferences, the values for these environmental impact factors are not known during modelling. In our model we use variables to represent characteristics of specific service environments.

External variables affect the whole modelled service system. While we only show the application for call centre components in our example, the amount of incoming calls might also influence other components, e.g. the processing of reshipment (since every 10th call may be a reshipment enquiry). Figure 4 shows the integration of variables to calculate specific KPIs. To

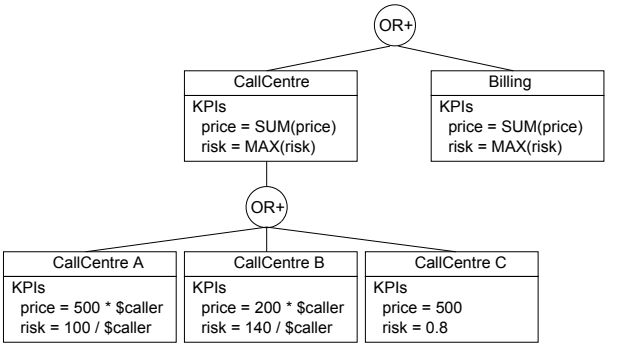

Figure 4. Example portfolio with variables

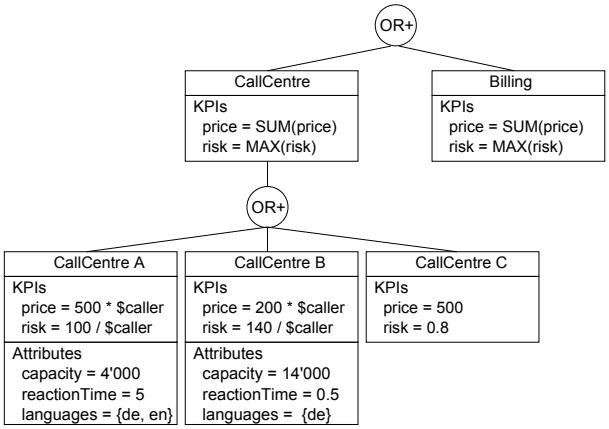

Figure 5. Example portfolio with variables and attributes

distinguish between variables and KPIs, variables are preceded by a dollar symbol. In the example the price and risk is calculated depending on the variable *caller*. Using variables facilitates the reusability of service components since KPIs do not need to be static and can also reference the service environment.

All used variables are formally represented in the set *VariableNames*. To allow greatest possible flexibility, variables are not restricted to specific data types. Since they represent the service environment, variables affect all components in the portfolio and have a global namespace. Thus, the variable *caller* in figure 4 has the same meaning both in component *CallCentre A* and in component *CallCentre B*. The values of variables are defined during configuration of customer specific services as shown in section III-B.

C. Attributes

Nonfunctional properties of services are of great importance to define service prospects and limitations. For example, a call centre might have a capacity limit of manageable calls per day. Though no restriction on the functionality itself, this is an important information about the capabilities of a component. Therefore, to enable meaningful configurations, it is necessary to specify nonfunctional properties in a consistent and formal way.

Especially in the domain of web services there exist a variety of different approaches to represent nonfunctional properties, e.g. [1], [36], [40]. However, for the sake of brevity we omit formal attribute definition and represent nonfunctional properties using simple free text attributes of service components. Based on these attributes, further constraints can be formulated to allow and disallow specific components, c.f. III-C. An extensive discussion about possible non-functional properties is conducted in [32]. Amongst others, the properties temporal and local availability, price, payment methods, penalties, right, and obligations are presented. Based on a given set of nonfunctional properties the restrictions of service components can be described very detailed.

The formal representation of attributes is similar to the representation of KPIs. They are determined by the sets *AttributeNames* and *AttributeValues* containing the names and values of used attributes. To connect a component with attributes we use the mapping *AttributeMapping* defined as follows.

$$AttributeMapping: Components \rightarrow P(AttributeNames \times AttributeValues) \quad (1)$$

To simplify attribute access during configuration we additionally define the mapping *AttributeValue*.

$$AttributeValue: Components \times AttributeNames \rightarrow AttributeValues \quad (2)$$

Attributes describe a specific component (and thereby characteristics of the underlying process). An attribute itself is represented by an arbitrary data type. The visualisation of attributes in the service model as well as its formal representation according to the definition of the mapping *AttributeMapping* is shown in figure 5. Both call centres are enriched with the attributes capacity and reaction time represented by numerical values and the attribute languages represented by a set of values. To clarify the difference between KPIs and attributes, we added a visual distinction between them in the components.

The formal representation of the attributes used in the call centre use case is as follows.

$$AttributeNames = \{capacity, reactionTime, languages\}$$
$$AttributeValues = \{4'000, 14'000, 5, 0.5, de, en\}$$
$$AttributeMapping(CallCentreA) = \{(capacity, 4'000), (reactionTime, 4), (languages, \{de, en\})\}$$
$$AttributeMapping(CallCentreB) = \{(capacity, 14'000), (reactionTime, 0.5), (languages, \{de\})\}$$

In contrast to KPIs, attributes have a fixed value defined during modelling phase. Furthermore, they are not automatically inherited through the service model. This restriction is

necessary because during modelling time it is not clear how attributes will be combined during configuration. Depending on customer preferences, different combinations are possible. In the call centre example, one customer might be able to combine different call centres to increase the capacity where other customers do not allow this combination. However, it is possible to use attributes in the calculation of KPIs, e.g. the monthly costs of a call centre may depend on the dimensions of a rented office space. Thus, attributes can be used in descendent and in cross-tree calculations as shown in figure 3 for KPIs but are not propagated.

It is not necessary to define attributes for every component in the model. For example, in figure 5 *CallCentre C* has no attributes. Missing attributes can occur when additional details about a specific service offer are not (yet) known. Therefore, missing attributes can be (but do not have to be) an indicator for underspecified and not well-known components in the portfolio.

III. Configuring services

Until now we have shown the integration of external variables representing the service environment and the description of nonfunctional properties of service components using attributes. Based on these concepts in this section we move on from modelling service portfolios to configuration of customer specific service offers considering KPIs of services. For comprehensibility we will recall configuration foundations in the next section.

A. Foundations

During service modelling, components and their dependencies between each other are defined. The formal definition of components can be used in twofold manner. First, it is possible to generate predefined service bundles consisting of different components. However, real benefit is achieved when customer specific configurations are generated. During configuration, components are selected and combined. Due to the formal representation of dependencies it is possible to verify the validity of service offers. The configuration is one approach to tackle the opposition between standardisation to reduce costs and customisation to offer a flexible portfolio. Due to aggregation of KPIs the performance of combined services can be assessed. We presented a formal definition of the configuration in [9].

If external variables are used during modelling it is necessary to define their values for a specific configuration. This is shown in section III-B. Furthermore, in section III-C we present a way to automatically constraint the selection of service components. Constraints can be used as restrictions on configurations.

B. Variable value definition

In section II-B, we stated that external variables have a global namespace. Therefore, the definition of their values is straightforward using the mapping *VariableValue* as follows.

$$VariableValue : VariableName \rightarrow VariableValues \tag{3}$$

The call centre example consists of only one external variable *caller* representing the expected amount of incoming calls per day. Using $VariableValue(caller) = 7000$ we set its value to 7000. After referenced variables have been set to a fixed value the calculation of KPIs for components using these variables is possible. Therefore, the price of component *CallCentre A* is set to 3'500'000 and its risk is set to 1/70. Analogously, *CallCentre B* has a price of 1'400'000 and a risk of 1/50.

C. Constraints

Using the above mentioned concepts of component attributes and environment variables as well as the already known performance indicators we are now able to formulate detailed constraints on service components. We consider constraints as a filter to choose only suitable components. For example, in the call centre scenario a typical constraint states that the capacity of a call centre must be greater or equal to the amount of incoming calls. This is an essential requirement for successful service provision.

During configuration it is possible to distinguish between hard and soft constraints. While members of the first one have to be satisfied because otherwise service provision is impossible, the latter ones should be satisfied because they are a potential risk during provision. However, we cannot define the type of a constraint in advance. For example, for a call centre provider it is of utmost importance to satisfy the capacity constraints mentioned above. In contrast to this, for a service provider offering but not focusing on call centre provision this constraints may not be as equally important. Finally, one service provider may have different clients emphasising service aspects in different ways. To overcome this challenge, all possible constraints on service models are specified in one set *Constraints* and during configuration it is possible to define which constraints are hard and which are soft by assigning them to the sets *HardConstraints* and *SoftConstraints* where unassigned constraints are automatically considered as being soft. In the next paragraphs, we show the definition and evaluation of constraints.

1) Definition of constraints: To allow for greatest possible freedom in constraint formulation the constraints are based on predicate logic. This is similar to the concepts used in feature models in software product lines, c.f. [29]. Constraints can be defined over values of attributes or KPIs. We use the mappings *AttributeValue* defined in section II-C and *KPIValue* analogously defined in [9] to access values of attributes and KPIs. The capacity constraint for components mentioned above combined with a price constraint is specified with the

following formula using the generic identifier *Components*.

$$\begin{aligned} &\forall c \in Components: \\ &AttributeValue(c, capacity) \geq 7000 \wedge \\ &KPIValue(c, price) \leq 1'100 \end{aligned} \tag{4}$$

To increase reusability of constraints, in addition to using fixed values it is also possible to use variables. This results in a simplified configuration since variables have to be entered only once and can be used both for constraint and KPI definition. The capacity constraint referencing the variable *caller* can be formulated as follows.

$$\begin{aligned} &\forall c \in Components: \\ &AttributeValue(c, capacity) \geq \$caller \end{aligned} \tag{5}$$

More often than not, it will be the case that different unrelated components are characterised by attributes with equal names. However, these attributes may not have the same meaning, e.g. the attribute capacity might also be used in different components that are not relating it to the expected amount of incoming calls. Therefore, developers need techniques to distinguish between these model parts. This is achieved by using a specific component name in the quantifier of a constraint. For example, the constraint

$$\begin{aligned} &\forall c \in CallCentre: \\ &AttributeValue(c, capacity) \geq \$caller \wedge \\ &KPIValue(c, price) \leq 1'100 * \$caller \end{aligned} \tag{6}$$

will only be evaluated for components that are in the transitive closure of child components of the component with the identifier *CallCentre*. The transitive closure of a component contains all its direct and indirect child components. As the service configuration graph is a tree and thus a specialisation of a directed graph, existing algorithms to calculate the transitive closure for graphs can be applied. A selection is, for example, shown in [26].

2) Evaluation of constraints: During constraint evaluation it is necessary to distinguish between constraints over attributes and constraints over KPIs. The former constraints are checked against the set of components contained in the transitive closure as defined above. The latter ones are checked against the complete configuration. Thus, different evaluation strategies are applied. In the course of this section we refer to the capacity constraint defined in equation 6 and set the variable *caller* to 7'000.

Constraints over attributes affect single components. For every component that is referenced in the constraint it is a) checked, whether the component contains the mentioned attribute and b) verified, whether the component fulfils the constraint. Therefore, it is possible to clearly identify components that violate attribute constraints. For example, the capacity constraint is violated in component *CallCentre A* since its capacity is only 4'000. On the other hand, it is fulfilled in *CallCentre B* because the capacity is more than 7'000 and in *CallCentre C* because this component does not have an attribute *capacity* and accordingly the constraints is not checked against this components. If an attribute constraint is violated it is violated throughout the whole configuration process regardless of what other components are selected. The only ways to fix a violated attribute constraint is to deselect the respective component. Depending on the type of constraint the configuration will either be invalid (hard constraint) or have warnings (soft constraints).

On the contrary, constraints over KPIs are evaluated at the topmost occurrence of the KPI. The KPIs are identified by a breadth-first search and comparison is based on their name. Thus, the capacity constraint is evaluated at the component *CallCentre*. Since KPIs are usually a combination of underlying KPIs it is not possible to clearly identify components that violate KPI constraints. The capacity constraint states that the price in the component *CallCentre* has to be less than or equal to 1'100 per caller. Selecting *CallCentre A* and *CallCentre B* this constraint is still fulfilled, while adding *CallCentre C* violated the constraint. However, it is not possible to determine the violating component since deselecting *CallCentre A* would fulfil the constraint. In summary, a violated KPI constraint can be fixed by different configurations even containing the last selected component that lead to the violation.

IV. Querying Services

Based on the previously developed formalisations for service modelling it is possible to structure extensive service portfolios. This is to a great extend focused on the viewpoint of service providers. However, from a customer point of view different considerations have to be taken into account. In provider-driven configuration of customer specific configurations, customers can only select from services of one provider. The configuration process is usually guided by the provider to support the component selection. But on electronic service markets customer are not aware what services exist and what services meet their specific requirements. Therefore, they need ways to identify suitable service components.

Finding suitable components is a problem with a long history in software engineering when existing software has to be reused. On this account, [28] has introduced four general techniques for reusing software artefacts that can be transferred to service engineering, too. First, *abstraction* is the basic reuse technique and enables developers to comprehend different artefacts. Because we only reuse service components, abstraction is inherently present in our model. Second, by *selection* developers are supported in the process of selecting different artefacts. Heretofore, we support selection by customer specific configuration. However, for the selection process itself there is no support since customers need to find suitable services on their own. By using queries over service components we are looking forward to improve the selection process. The two remaining techniques are *specialisation* to represent similar artefacts in a consolidated way and *integration* to support developers during assembling of reused artefacts to complete systems. While the latter one can be achieved using the formal metamodel of our service model,

the first one is possible by integrating predefined component types into the model.

Queries over service components are used to search through a collection of components and automatically identify components that match specific customer requirements. As stated in section III-C constraints are one approach to capture requirements and validate the adherence of components against requirements. A challenge to keep in mind is the often occurring gap between query formulation and component description identified in [25]. Generally, component descriptions focus on the functionality by answering the question how a component works. In our model this fact is typically represented by underlying process models. On the other hand, queries usually focus on the problem itself and ask what components are capable of solving a problem. It is possible to close (or at least reduce) this gap by making extensive use of the above presented concept of component attributes. Though it increases initial modelling effort, the detailed description of component prospects and limitations supports efficient and accurate component search.

The definition of queries focuses the ability of customers to find service components they can use. Therefore, it is necessary that query formulation is simple and not too complex. The structure of queries has to adhere to our underlying service component metamodel. Similar as in constraint definition based on predicate logic the query language must provide selectors to select the components that are affected by a query. Furthermore, it must be possible to define constraints that are matched against components. A source of inspiration for queries might be the well-known database query language SQL, e.g presented in [18].

V. Related Work

In response to a growing complexity, a more individualised market and an economically motivated stronger focus on productivity aspects, the division of monolithic tasks in manageable components is being utilised in various areas. Although in other scientific disciplines this process is called *modularisation* we refer to *components* as a synonym for *modules*. Particularly in the field of industrial engineering the concepts of modelling complex structures with the use of smaller components is quite established [5]. Similarly in software engineering the use of components is employed on a broad base [37]. A hybrid form of software and classical services are IT-services, where modularisation is applied likewise [10].

Based on the modularisation, service configuration is enabled. Configuration is well known in industrial engineering [38] and in software engineering [20]. In service engineering, configuration gains importance as well [3]. However, existing approaches like [2] do not focus dependencies between service components and customer-driven configuration but rather on formalising overall service systems.

In the course of this work we presented one approach to define constraints on service components. Boustil et al. stated that existing approaches for service selection do not integrate user requirements [12]. In the domain of web services, Tosic et al. present an approach to define constraints [39]. In order to so, they introduce an XML-based constraint language called Web Service Offerings Language. Yu et al. use a graph based approach to select web services adhering to specific QoS constraints [41]. The academic literature has produced a variety of approaches to formulate queries for services. However, most of them are tailored to web services and do not take the specialities of complex business services into account. We tackle this problem by using the underlying formal metamodel for the specification of service components. Most existing query approaches focus on one specific aspect of services. For example, Baraka developed a query language for mathematical services [6], [7]. It works in conjunction with a description language for mathematical services presented in [8]. The approach also uses SQL-like query statements. Another stream of researches focus on the query for resources [23], quality of service aspects [21], [30], and service mashup [22]. The integration of different service aspects is in the focus of the work of Pantazoglou et al. They present a query language [33], a matching algorithm [35], and an associated engine [34]. Though this approach targets more than one aspect its application is limited to web services. To facilitate optimal service selection, Bonatti and Festa have examined different approaches in [11].

Components in our service model represent process models. Therefore, query languages and constraint definition for process models can be an interesting object of study, too. Awad has presented a visual query language for BPMN models [4] and Jin et al. query process repositories using graph isomorphisms [27]. To specify requirements (e.g. constraints) on process models, Momotko et al. developed an integrated methodology. Therefore, they developed a metamodel for a process description language [31].

VI. Conclusion

In this research paper we presented the extension of a previously defined service component model with variables and attributes. These concepts can be used to formulate constraints on the customer specific configuration of components. Based on these constraints it is possible to create more meaningful configurations. We are currently applying the concepts contained in the extended service component metamodel together with our industry partner in a real world scenario. It has shown that proper tool support is of utmost importance to manage complex service portfolios. Therefore, we have developed a prototype supporting the modelling and configuration of service components.

Furthermore, we investigated challenges that arise when the service selection process is customer-driven without guidance by service providers. A great obstacle for realising service component markets is the existing gap between service descriptions and the representation of customer needs. In our approach this gap is reduced by the extensive use of attributes and the ability to query for service components based on constraints.

Using attributes and variables in constraints and queries depends to a great extent on a homogeneous taxonomy. Till now integration of existing components in a portfolio needs a careful check of used attributes and variables. This problem is aggravated when customers query marketplaces with components from different providers. Generally, different providers use different taxonomies what results in poor component recalls. One approach to tackle this problem is the Unified Service Description Language (USDL), presented in [19]. This would allow for a consistent definition of service components. Another approach might be predefining service characteristics for specific components and establish service types based on these characteristics. Selection of components can then be based on predefined service types.

In addition, increasing query capabilities requires integrating a holistic approach using various modelling concepts. In this work we presented queries based on attributes and KPIs. As components are based on processes, existing approaches from this domain can be examined for their applicability. Furthermore, integration of resources is an important point to keep in mind. More often than not it will be the case that components need specific inputs—querying for components producing this input as output will be challenging.

REFERENCES

[1] Stephan Aier, Philipp Offermann, Marten Schönherr, and Christian Schröpfer. Implementing non-functional service descriptions in soas. In Dirk Draheim and Gerald Weber, editors, *Trends in Enterprise Application Architecture*, volume 4473 of *Lecture Notes in Computer Science*, pages 40–53. Springer Berlin / Heidelberg, 2007.

[2] Hans Akkermans, Ziv Baida, Jaap Gordijn, Nieves Pena, Ander Altuna, and Inaki Laresgoiti. Value webs: Using ontologies to bundle real-world services. *IEEE Intelligent Systems*, 19:57–66, 2004.

[3] Luis Araujo and Martin Spring. Complex performance, process modularity and the spatial configuration of production. In N. Caldwell and M. Howard, editors, *Procuring Complex Performance: Studies in Innovation in Product-Service Management*. Routledge, London, 2010.

[4] Ahmed Awad. Bpmn-q: A language to query business processes. In *Enterprise Modelling and Information Systems Architectures - Concepts and Applications , Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures*, pages 115–128, 2007.

[5] Carliss Y. Baldwin and Kim B. Clark. Managing in an age of modularity. *Harvard Business Review*, 1997.

[6] Rebhi Baraka. Mathematical services query language: Design, formalization, and implementation. Technical report, Johannes Kepler University, Linz, Linz, Austria, September 2005.

[7] Rebhi Baraka and Wolfgang Schreiner. Querying registry-published mathematical web services. *Advanced Information Networking and Applications, International Conference on*, 1:767–772, 2006.

[8] Rebhi S. Baraka. *A Framework for Publishing and Discovering Mathematical Web Services*. Dissertation, Johannes Kepler Universität Linz, Linz, August 2006.

[9] Michael Becker. Formales metamodell für dienstleistungskomponenten. Technical report, Universität Leipzig, 2011. to appear.

[10] Tilo Böhmann, Richard Gottwald, and Helmut Krcmar. Towards mass customized it services: Assessing a method for identifying reusable service modules and its implication for it service management. In *AMCIS 2005 Proceedings*, 2005.

[11] P. A. Bonatti and P. Festa. On optimal service selection. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 530–538, New York, NY, USA, 2005. ACM.

[12] Amel Boustil, Nicolas Sabouret, and Ramdane Maamri. Web services composition handling user constraints: towards a semantic approach. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '10, pages 913–916, New York, NY, USA, 2010. ACM.

[13] Martin Böttcher and Klaus-Peter Fähnrich. Service systems modeling: Concepts, formalized meta-model and technical concretion. In Haluk Demirkan, James C. Spohrer, and Vikas Krishna, editors, *The Science of Service Systems*. Springer, New York et al., 2011.

[14] Martin Böttcher and Stephan Klingner. Der Komponentenbegriff der Dienstleistungsdomäne. In K.-P. Fähnrich and B. Franczyk, editors, *Informatik 2010—GI Jahrestagung*, volume 1, pages 59–66, Leipzig, 2010. Lecture Notes in Informatics (LNI).

[15] Martin Böttcher and Stephan Klingner. The basics and applications of service modeling. In *SRII Global Conference 2011*, 2011. to appear.

[16] Martin Böttcher and Stephan Klingner. Providing a method for composing modular b2b-services. *Journal of Business & Industrial Marketing*, 2011. To appear.

[17] Hans-Jörg Bullinger, Klaus-Peter Fähnrich, and Thomas Meiren. Service engineering - methodical development of new service products. *International Journal of Production Economics*, 85:275–287, 2003.

[18] S. Cannan and G. Otten. *SQL–The standard handbook: based on the new SQL standard*. McGraw-Hill, 1993.

[19] Jorge Cardoso, Alistair Barros, Norman May, and Uwe Kylau. Towards a unified service description language for the internet of services: Requirements and first developments. *Services Computing, IEEE International Conference on*, 0:602–609, 2010.

[20] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models, 2004.

[21] Giuseppe Damiano, Ester Giallonardo, and Eugenio Zimeo. onqos-ql: A query language for qos-based service selection and ranking. In Elisabetta Di Nitto and Matei Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, volume 4907 of *Lecture Notes in Computer Science*, pages 115–127. Springer Berlin / Heidelberg, 2009.

[22] Weilong Ding, Jing Cheng, Kaiyuan Qi, Yan Li, Zhuofeng Zhao, and Jun Fang. A domain-specific query language for information services mash-up. *Services, IEEE Congress on*, 0:113–119, 2008.

[23] Sebastian Günther, Claus Rautenstrauch, and Niko Zenker. Service-oriented architecture: Introducing a query language. In Martin Bichler, Thomas Hess, Helmut Krcmar, Ulrike Lechner, Florian Matthes, Arnold Picot, Benjamin Speitkamp, and Petra Wolf, editors, *Multikonferenz Wirtschaftsinformatik*. GITO-Verlag, Berlin, 2008.

[24] Christian Grönroos and Katri Ojasalo. Service productivity: Towards a conceptualization of the transformation of inputs into economic results in services. *Journal of Business Research*, 57(4):414–423, 2004. European Research in service marketing.

[25] Scott Henninger. Using iterative refinement to find reusable software. *IEEE Software*, 11:48–59, 1994.

[26] Yannis Ioannidis, Raghu Ramakrishnan, and Linda Winger. Transitive closure algorithms based on graph traversal. *ACM Trans. Database Syst.*, 18:512–576, September 1993.

[27] Tao Jin, Jianmin Wang, Marcello La Rosa, Arthur H.M. ter Hofstede, and Lijie Wen. Efficient querying of large process model repositories. Report 39060, Queensland University of Technology, December 2010.

[28] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24:131–183, June 1992.

[29] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[30] Delnavaz Mobedpour, Chen Ding, and Chi-Hung Chi. A qos query language for user-centric web service selection. *Services Computing, IEEE International Conference on*, 0:273–280, 2010.

[31] Mariusz Momotko and Kazimierz Subieta. Process query language: A way to make workflow processes more flexible. In András Benczúr, János Demetrovics, and Georg Gottlob, editors, *Advances in Databases and Information Systems*, volume 3255 of *Lecture Notes in Computer Science*, pages 306–321. Springer Berlin / Heidelberg, 2004.

[32] Justin James O'Sullivan. *Towards a precise understanding of service properties*. PhD thesis, Queensland University of Technology, Faculty of Information Technology, 2008.

[33] Michael Pantazoglou and Aphrodite Tsalgatidou. The unified service query language. Technicel report, National and Kapdistrian University of Athens, Athens, Greece, July 2009.

[34] Michael Pantazoglou, Aphrodite Tsalgatidou, and George Athanasopoulos. Discovering web services and jxta peer-to-peer services in a unified manner. In Asit Dan and Winfried Lamersdorf, editors, *Service-Oriented Computing—ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 104–115. Springer Berlin / Heidelberg, 2006.

[35] Michael Pantazoglou, Aphrodite Tsalgatidou, and George Athanasopoulos. Quantified matchmaking of heterogeneous services. In Karl Aberer, Zhiyong Peng, Elke Rundensteiner, Yanchun Zhang, and Xuhui Li, editors, *Web Information Systems—WISE 2006*, volume 4255 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin / Heidelberg, 2006.

[36] Stephan Reiff-Marganiec, Hong Yu, and Marcel Tilly. Service selection based on non-functional properties. In Elisabetta Di Nitto and Matei Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, volume 4907 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin / Heidelberg, 2009.

[37] Clemens Szyperski. *Component software - beyond object-oriented programming*. Addison-Wesley, London et al., 2002.

[38] J. Tiihonen and T. Soininen. Product configurators—inforamtion system support for configurable products. In T. Richardson, editor, *Using Information Technology During the Sales Visit*. Hewson Group, Cambridge, 1997.

[39] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. Wsol—web service offerings language. In Christoph Bussler, Richard Hull, Sheila McIlraith, Maria Orlowska, Barbara Pernici, and Jian Yang, editors, *Web Services, E-Business, and the Semantic Web*, volume 2512 of *Lecture Notes in Computer Science*, pages 57–67. Springer Berlin / Heidelberg, 2002.

[40] Hiroshi Wada, Junichi Suzuki, and Katsuya Oba. Modeling non-functional aspects in service oriented architecture. *Services Computing, IEEE International Conference on*, 0:222–229, 2006.

[41] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1, May 2007.