Context-based Modeling: Introducing a Novel Modeling Approach

Martin Juhrisch[1], Gunnar Dietz[2]

[1]Technische Universität Dresden
Lehrstuhl für Wirtschaftsinformatik, insb. Systementwicklung
martin.juhrisch@tu-dresden.de

[2]Dongbei University of Finance and Economics
International Education Centre
mail@gdietz.de

Abstract: Despite the fact that researchers agree on the importance of enterprise models to an organization's success, the knowledge about how to handle problems where models have to be compared or integrated is still fuzzy and vague, and there is little agreement regarding compositional facets. Highly interesting is the interaction between models in shared modeling projects — e. g. between requirement models and service implementations in a service-oriented architecture (SOA). This article highlights an approach that allows to prevent integration conflicts in conceptual models already during the modeling phase. The influence of this approach on conceptual modeling and its use in intra-organizational collaborations is investigated. We show the inherent complexities of model-mediated interactions between domain experts and IT-service developers. It is suggested that at an early stage of the modeling process the use of guidelines has an substantial benefit for avoiding integration conflicts in conceptual models. Furthermore, due to the way how the approach bridges the semantic gap, changes of business requirements as well as technical implementation restrictions influence each other. This results in an ongoing system development process that can be interpreted as a permanent management of application systems. Our results contribute to model-based management theories that have so far neglected the distributed construction of conceptual models.

1 Introduction

In [23] a new approach was introduced, called the Description Kit Approach (DKA), that allows the development and configuration of service compositions. One focus of that article was the development of the inter-level architecture of the DKA that can't be established using classical methods of meta-modeling. Even if the modeling of Description Kits in principle quite similar as the creation of meta-models, and therefore fits technically into the field of meta-modeling, the "philosophical" background is quite different. In contrast to meta-modeling the goal is not the creation of new modeling languages.

One new aspect of the DKA is the embedding of guidelines that at least influence the pro-

cedure model of the modeling method. Therefore guidelines — restrictions that the DKA establishes for the modeling methods — an influence on the ordinary modeling process by influencing or even determining the procedure model. Since the DKA is a generic approach, it can be even used as a method for (indirectly) creating procedure models. These result in annotated models that used domain spanning concepts, which again allow an automized evaluation. A second aspect is the inclusion of the adaption process of the domain specific modeling language into the problem solving process by weakening the barrier between language creation and language use. This is of great importance, since the guidelines result from a labile consensus, which simplifies the model comparability and only by this allows the establishment of problem solving methods.

Another focus was the description of the different model types that can be used and the derivation of language concepts for each of the three modeling levels. The abstract syntax has been discussed, since this is the foundation for an automated model use. The fundamental notation of the elements has been described and shown in examples. A problem that should be solved in a concrete situation can be translated for model-based problem solving methods into the combination or comparison of different domains. Examples are here analysis models and design models, or to compare actual (as-is) domains with target domains, or to combine or compare models from a distributed modeling project. The model for describing guidelines — the description kit language (DKL) — is generic, so that each way of describing a certain situation can be restricted relatedly to the aim of solving the concrete problem. Therefore a general methodology has been created that simultaneously meets the requirements of a certain domain.

2 Theoretical Background

2.1 Method Engineering

The following chapter explains some fundamental definitions from the areas of the present area of research. Starting with the dimensions of a language, classes of languages are discussed. This discussion is necessary, since languages plays a major role when reformulating conceptual content with the intention of creating service compositions.

2.1.1 Classification of Language

A language that is characterized by a precise syntax, a formal semantic and a well-defined usage of the linguistic terms is called a formal language [10, pp. 9]. A precise syntax only allows a limited set of valid and well-formed terms. Semantically formal means that each linguistic term has a unique interpretation [10, pp. 9]. Therefore also the usage of that term is well-defined.

Natural languages are the opposite of formal languages. Their linguistic terms come from a real-world (material) domain, and their syntax and semantic have been evolved historically within a linguistic community [28, pp. 56]. Typically for natural languages is the diversity

of linguistic terms, which results in a high semantic power. However, this also results in several linguistic defects due to the contrariness of their lexicographical and structural possibilities [33], [32].

All previous attempts to describe a material semantic in the sense of a final set of syntactical rules in a complete formal way have been fruitless [37, pp. 97]. Therefore it remains a big challenge, to formally specify a material domain with sufficient preciseness, such that models within that domain can be used for an automatic transformation [35, p. 6].

Another way of classifying languages is to differentiate the phases language creation and language use. Language creation means to the description of the syntax and semantic of a language, e.g. a modeling language, by referring to a meta-language. In this case one refers to the object language that is the language used to describe real-world phenomena within a certain domain, and the meta-language that describes the object language [11]. If the meta-language is again a modeling language, one refers to a meta-modeling language [11, p. 696]. The term language use refers to the usage or application of the object language to describe a certain domain. One application field of Business Informatics (german "Wirtschaftsinformatik") research that derives from distinguishing object languages and meta-languages is the development of methodologies.

2.1.2 Conceptual Modeling Languages

Since formal languages are not appropriate for complete descriptions of a material semantic, their applicability for analyzing social systems is doubted [10, p. 10], [31, pp. 26]. Semiformal languages have been established in the field of systems development for an understanding of material domains by modeling problems that are not well-structured, cannot be structured objectively, but that are bound to subjects and goals [17, pp. 7], [18, p. 125]. Their linguistic instruments has to cover informal aspects that allow a deep understanding of the operational domain and its potential for reorganization by information and communication technology (ICT). Furthermore, the implementation of future information systems has to be prepared by aspects of formal language [11, p. 696]. In this context one speaks of conceptual modeling languages. A dominant property of these languages is a combination of a formally defined syntax and an extensively material semantic of the linguistic terms [34, p. 44], [12, p. 34]. The semantic is normally derived from a concept of a technical language. This is done by annotating linguistic terms by natural language terms. Typically the terms of the application domain are not contained in the language, for which a grammar is described with the help of a meta-language [26, p. 53]. A conceptual modeling language only comes to live by combining the modeling grammar and a set of technical linguistic terms. Conceptual modeling grammars have a high relevance within the communication process between the individuals involved in the analysis phase of system development [48, pp. 104]. By combining a technical language with a modeling grammars one gets a conceptual modeling language, which can be used to create conceptual models, see Figure 1.

In the following we assume that modeling grammars always base on a diagrammatic notation, whose application results in graphical representations [40, pp. 159], [15, pp. 510], [11, p. 696], [47, p. 363], [12, p. 84].

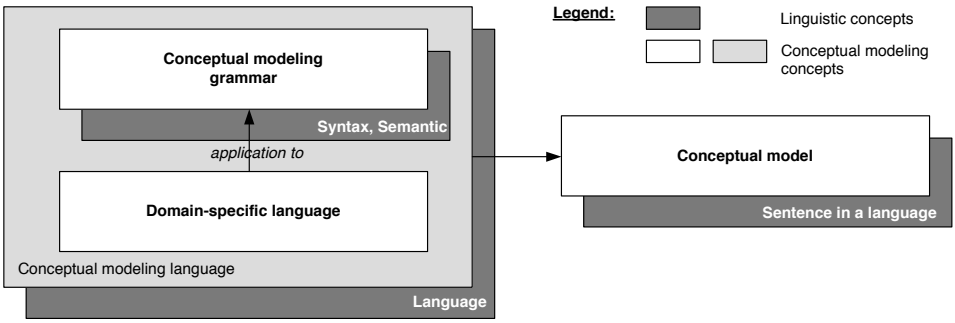

Figure 1: Conceptual modeling

One primary goal of business informatics ("Wirtschaftsinformatik") is the "... analysis, development and use of informatioin systems" [49, p. 2]. An information system (IS) is a system that can process information, where "process" means collect, transfer, transform, save and supply [9, p. 65]. The utilization of models as an instrument for handling complexity by abstraction allows a well-planned procedure when designing information systems, if one succeeds in systemizing the analytic and creational character of the modeling [9, p. 119], [3, pp. 27], [36, p. 63].

For this reason, methods on the base of using models have been established [43, p. 33], [50, p. 371], [39, p. 27] — so-called model-based methods. In the context of information systems design a method is normally understood as a specification in accordance to aims and available resources, which describes a well-planned procedure for the solution of a certain type of problem [5, p. 5], [27, p. 876], [41, p. 239], [51, p. 34], [7, p. 36], [13, p. 32]. The aim is to successfully achieve the required goals. The method should be intersubjectively comprehensible, which means it should systematically lead to the solution of a problem [51, p. 34]. Methods have to be designed for the problem solution [13, p. 33]. To derive concrete tasks and stakeholders, the method must declare certain elements as obligatory and restrict the users' freedom of decisions in this area, e.g. by giving detailed demands and specifications [43, p. 33], [13, p. 34]. On the other hand, a method can also leave a certain freedom for deciding how to use the method.

GUTZWILLER analyzes approaches from information systems research in the domain of method development and approaches for a standardization in this area and extracts some commonly used concepts [16, pp. 15]. From this he derives general method components [16, p. 12]: The procedure for the problem solution is determined by a series of activities, whose course of action has to be described by the procedure model. Within each activity some method products (results) are generated or used, e.g. documents and artifacts. How to perform an activity is determined by techniques. Roles describe stakeholders who use these techniques to generate results. A meta-model is specified as a conceptual data model for all method products. A modeling method contains rules that describes how to use the linguistic concepts of a modeling language and in which order to do that [39, p. 27] (and similar [50, p. 371], [43, p. 33]). Furthermore the modeling method contains minimal

requirements for the models to create. If one transfers GUTZWILLER's method components to modeling methods, the following components are essential: The meta-model of the method, the activities within the modeling process and the result of the model creation. A focus clearly is on the meta-model for the modeling grammar [25, p. 377]. If modeling methods again are developed in a model based way, one gets models for the linguistic concepts and models for the procedure of the model creation.

Language Model: Conceptual data model of the method [16, pp. 12–14].

Procedure Model: Model of the activities of the model creation [21, p. 116].

2.2 Service-oriented System Design

For the development of a process oriented information architecture the paradigm of service orientation has been established. The literature contains several variants of the definition of a service-oriented architecture (SOA). All these definition have in common that a SOA describes a system architecture that offers business services in combination with the application system functionality in the form of (electronic) services [8, p. 3 and p. 50].

In contrast to component orientation the service orientation underlines a high autonomy of a service and the extent of the offered functionality [2, pp. 634]; [29, p. 19]. The conceptual change from objects or components to services is driven by the functional requirements to a service, its technical implementation, and its economic importance.

A service encapsulates as set of functions that encompass a closed (self-contained) process activity. The data a service processes corresponds to complete business objects.

In contrast, object orientation means to encapsulate data and functionality in the concept of a class [4, p. 156]. A component contains the functionality of several classes and therefore a larger amount of data and functions (or better said methods) [42, p. 153]. The functions, however, are still fine-granular. They serve the purpose of the integration of components into an application [19, p. 50]. An example for this are getter methods. They just return the value of an attribute of an object to another object that is associated to the first object. This functionality does not correspond to a process step in a business process, the exchanged data does not correspond to business objects, as they are depicted within the conceptual modeling of cross-application processes [19, p. 50]; [1, p. 24].

Services encapsulate functions that correspond to functional requirements [8, pp. 280]. A service interface has to handle complete business objects instead of single attributes [30, p. 50]. Therefore services have to adapt their granularity on the level of the application system to the process requirements [42, p. 153]; [1, p. 24].

Services and service users are developed independently and at different time. Any composition of service functions is done after developing the services first. Different to objects or components, services and service users are developed by organizationally distributed teams, at different times, and using different technologies [38, p. 45]; [19, p. 50]; [45, p. 16].

When considering reusability of services, the focus lies on a process oriented composition. In object orientation a system is composed into objects, which are created mirroring real world objects and concepts. In contrast, services are created by bundling tasks that follow common actual and formal goals and operate on common business objects. An example would be the service "Inventory" with functions like "ordering spare parts", "check availability", "reservation", etc. The aim is a high cohesion of the functions within one service and a weak coupling between different services. The reuse of a service function and the creation of complex functions is done by composition [24, p. 327 and p. 74]; [52]. This distinguishes services from objects, where the reuse of classes is done on the level of the source code by inheritance [14, p. 32].

Services are, in contrast to objects, deployed. That means they are installed externally and made available externally. A service user binds a service during runtime [6, p. 10]. Service and service user are compiled independently. During runtime an application system constitutes of a certain set of service functions as a so-called Composite Application [19, p. 191]. This distinguishes services from components as well as classes, which are typically made available as libraries. A library must be integrated into a program and must be compiled together with the software. On the other hand, the communication between a service and the service user has to be stateless [8, p. 332]; [46, p. 62]). There is no mechanism for object state or life cycle management for services [19, p. 51]; [46, pp. 61].

3 Context-based Modeling

This section presents the procedure model of the DKA, which guides systematically through the modeling on every modeling layer and describes algorithms for the model comparison. In [22] the concept of the different modeling layers M^1, M^{0*}, M^0 have been described, as well as the language artifacts for the different layers and their relationships. These relationships include different kinds of instantiation relationships, which implicitly describe how to navigate between models within the method.

The DKA combines two different progress models: First for the methodology of the DKA itself, second for specific problem solutions done by using a model-based approach following the DKA. The following section describes the procedure how to put information methodically into models in the sense of the DKA (see Figure 2).

The method engineer has to develop a recommendation for a set of Description Kit Types (DescKitTypes). This has to be done in a consensus with the domain expert based on the concrete problem to be solved. In the progress model this happens directly after the definition of the meta-models for the modeling language(s) that should be used, i.e. as soon as the object language has been determined. For this the necessary modeling language has to been derived from the problem and the meta-model on M^1M has to been described. The development of a recommended set of DescKitTypes is a creative task for the method engineer that cannot be automized. Several DescKitTypes, however, may be fixed for certain scenarios. For example, in the case of service identification or composition the DescKitTypes *Interface*, *Input*, *Output*, *AttributeType*, *Attribute*, *Object*, and *Service* are

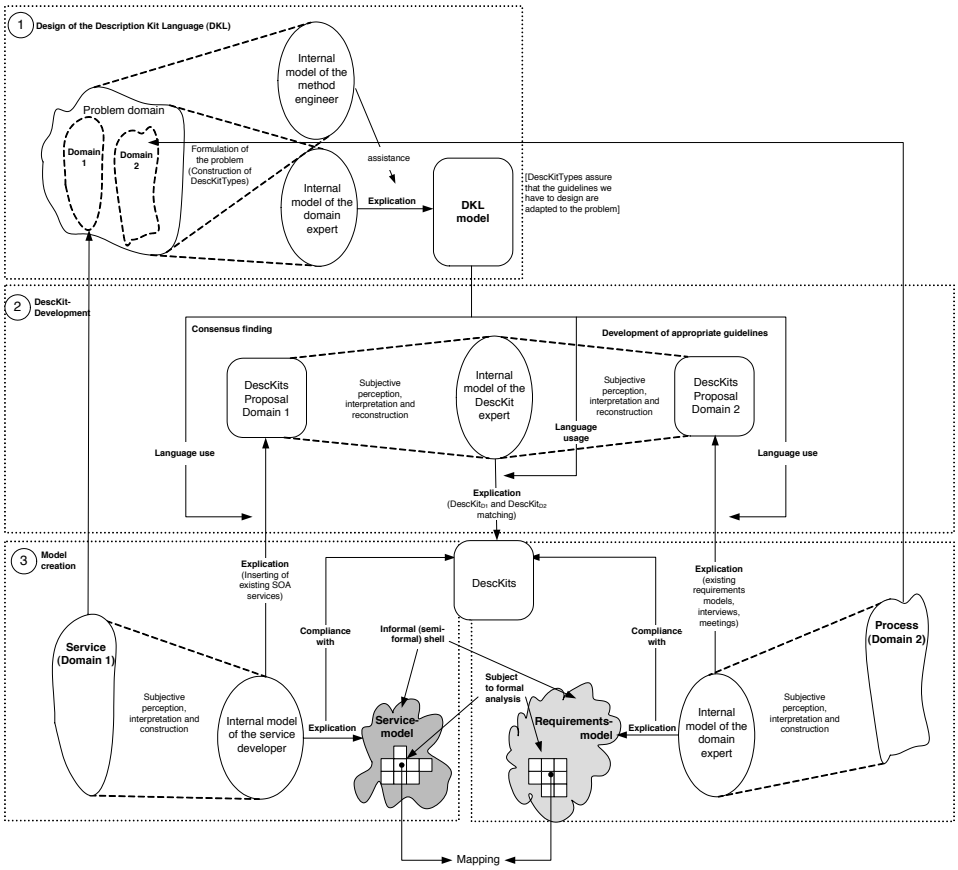

Figure 2: Role model of the DKA

predefined. They are the result of an analysis of the domain conflict [22]. Their aim is the description of objects and object states within analysis and service models. As a result of this activity one gets a set of DescKitTypes, that has to be used for the model creation in both domains simultaneously (see Figure 3).

The set of DescKitTypes and the set of Relation Types has to be added to the meta model layer of the DKL model. The creation of the DKL model includes the hierarchization of the different DescKitTypes and the definition how to use them. The latter means to create links from the DescKitTypes to the concepts of the modeling language. By this, the DKA establishes two generic ways for a restriction of the freedom of modeling:

- The first possibility is to integrate additional information into analysis models in a restricted way. For this a link between the meta-model of a conceptual modeling language (which has to be determined previously) and the meta-model for the DKL. This link determines, which DescKits are obligatory or optional for which model

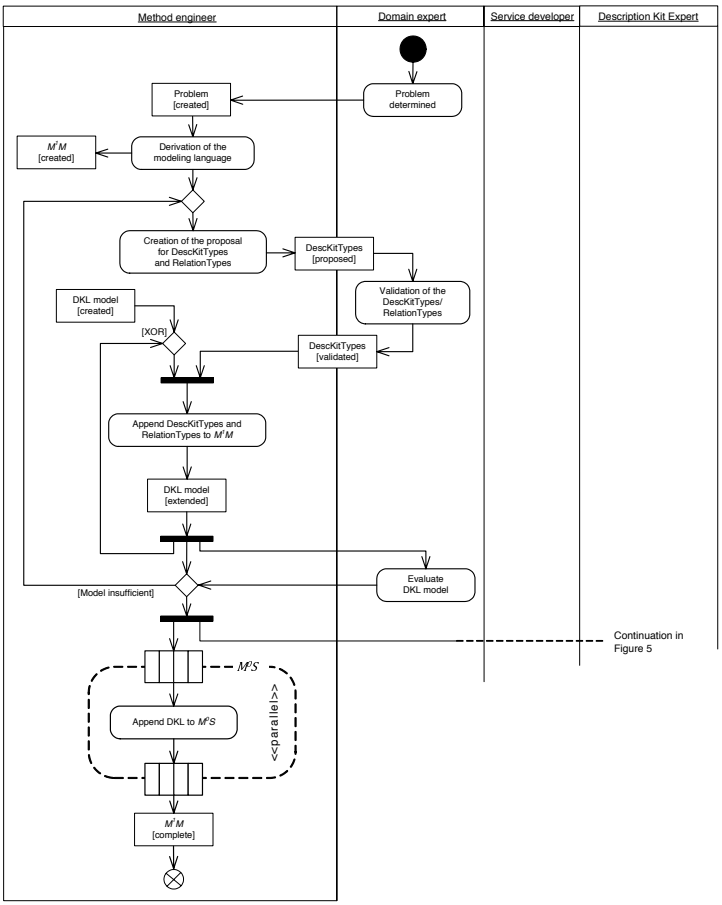

Figure 3: Procedure model for the model creation

elements.

- The second possibility is to combine several DescKitTypes to a complete modeling language. The model creation is then done by only using DescKits and Relations.

A restricted or enforced use of certain DescKits for certain modeling elements not only means a restriction of the modeling freedom with regards to the DescKits, but also a restriction of the (ordinary) modeling process itself. The meta-model of the DKA then is complete and allows the method engineer with help of the model of the DKL to ensure that the created guidelines are comply with the problem to be solved.

The tasks for the determination of DescKits on the layer M^{0*} and their instances in the models (Descriptions on the layer M^0) are done by different persons. The determinations on layer M^{0*} require consolidation processes that must involve both domain experts and

technical experts who want to model in a functional and technical way, respectively. The establishment of a consensus between those persons and the modeling of the result of the consensus are task that are not included in classical modeling projects. For the engineering of the DescKits therefore a new role *description kit expert* has to be established as a specialization of the role modeling expert. An approval process starts with the recommendation of a set of DescKitTypes by the description kit expert. Iteratively this recommendation has to be approved or declined by the domain experts and technical experts (service developers) and replaced by an improved recommendation, until a consensus has been found.

Domain experts and technical experts first have to have a rough idea about what the possible DescKits could be. That means the environment, in which one wants to create models, has to be clear. The group of domain experts then describes certain concepts that match the DescKitTypes and therefore are adapted to the problem to be solved. These concepts then have to be put into a hierarchy to derive DescKits. These DescKits then have to be created and added to the modeling layer M^{0*}.

If requirements models, e.g. conceptual data models, already exist, one can derive certain information from these to ease the above described process. Of course such an extraction cannot be done completely automatically. However, the results of the extraction represent a series of suggestions for the approval process. Figure 4 shows exemplarily the translation of a conceptual data model (here a UML class diagram) into suggestions for DescKitTypes and DescKits.

Note that suggestions for both DescKitTypes and DescKits that should be used for the DKL can be derived from a UML class diagram. Abstract classes e.g. are candidates for DescKitTypes. The same is true for "fundamental" classes, which however cannot be defined precisely. The number of parameters (class elements that represent primitive types) could give a hint for this decision. Other "not so fundamental" classes on the other hand could be candidates for DescKits. The description kit expert has to sort and rate all suggestions, but as a first suggestion this procedure could be very helpful.

Furthermore the group a service developers can input the existing SOA services into a service catalog. The WSDL descriptions include the ingoing and outgoing messages, which furthermore include a list of complex object types. These objects again represent candidates for concepts that may be represented by DescKits.

These two different ways to generate suggestions then have to be combined using best practices to come to common concepts. The result of the approval process should be then a consensus about which of these concepts should be represented by DescKitTypes and DescKits. With the help of the resulting DKL the description kit expert has then to design the DescKits on layer M^{0*} (see Figure 5). At this point these candidates for DescKits have to be put into a hierarchy that matches the hierarchy of DescKitTypes. This is done by a concretization on DescKit layer M^{0*}.

After the previous steps are done, the DescKits have to be implemented into the actual modeling project. The description kit expert again bears the responsibility for this. This is done iteratively accordingly to the procedure model: The first use of the DescKits to create Descriptions is done by persons, who also would perform the modeling within a

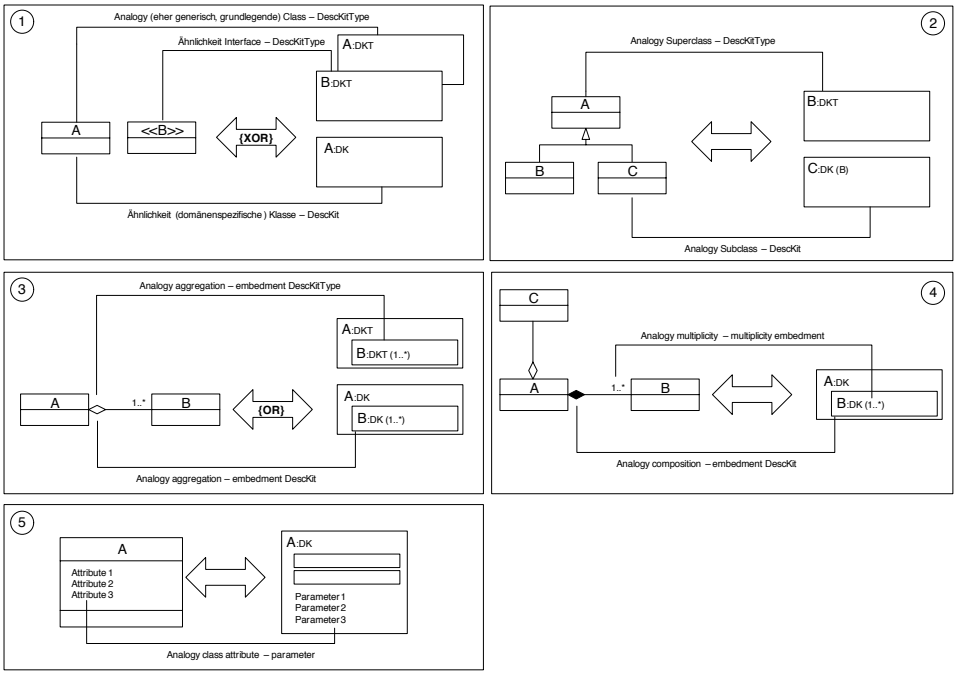

Figure 4: Transformation of a UML class diagram to DescKitTypes/DescKits

certain domain in classical modeling projects. These are the domain experts (perhaps with help of the modeling experts) and the service developers. A new aspect is, however, that the modeling process includes adaption processes: A DescKit can be modified during the modeling process.

On the meta-model layer M^1 no domain specific language constructs are modeled, but the language for the DescKits for the domain specific language constructs. Since the modeling layer M^{0*} is near to the layer M^0, an iterative procedure allows to work on both layers alternately or even simultaneously. This means that the method supports an iterative adaptation process between layers M^{0*} and M^0 clearly and transparently and allows for example to "promote" modeling information from M^0 to M^{0*}.

At each time of the modeling on layer M^0 exactly on configuration of a DescKit of layer M^{0*} is active. Modifications of a DescKit, e.g. by defining new parameters, transfer a DescKit from one configuration to another. The adaptation operations should allow a modification that does not destroy the integrity of existing conceptual models. That means that a completely free modification of DescKits that have been used in models already is not possible without further ado. Not critical is always the addition of new elements, as far it is not obligatory. More difficult is the modification and deletion of elements. A change management is necessary for the DescKit, so that dependencies can be solved.

During the modeling process change requests may occur. These have to communicated

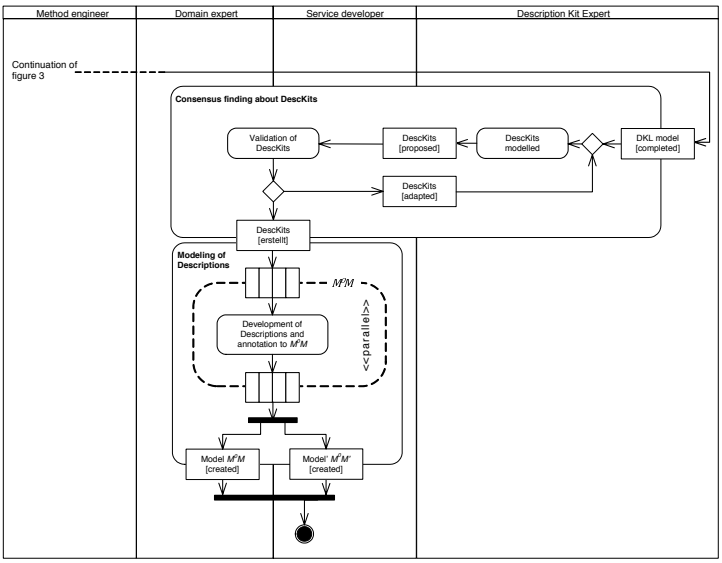

Figure 5: Procedure model to model creation 2

to the description kit expert. Using some feedback from the domain experts and technical experts, he plans modifications to the model on M^0*. If the modeling process is done in a team, clear responsibilities are important. It must be prevented that each method developer just creates DescKits as just needed by him. Certain persons perhaps should have no access to modifications on M^0*.

On layer M^0 DescKits now can or must — depending on how the DKL has been linked with the modeling language on M^1 — be used in concrete models. The modeling of Descriptions and within Descriptions is done hierarchically and therefore in another way as e.g. classical process modeling (see Figure 6). The hierarchic top-down approach starts with a highly aggregated view on a certain fact and then step-by-step refines the description with more detailed information in a highly structured and well-defined way. A hierarchy of descriptions with increasing level of detail is the result.

This hierarchical way of modeling supports the modeler in distinguishing certain types of information about a real world phenomena from other types of information, to hide some part of the information or to handle different types of information differently. This is done without modifying language, but merely by an adaption of the descriptive structure of the DescKits. To use a DescKit, it has to be instantiated in a classical (linguistic) way. The description gets a unique name and is part of the ordinary modeling layer M^0. At the same time a Description represents (in another sense of instantiation) a certain DescKit, which sets the framework (or guideline) for the further proceeding in the sense of "shaping" the description by the following tasks:

- Selection of a certain set of parameters and embedded DescKits (sub-DescKits) of

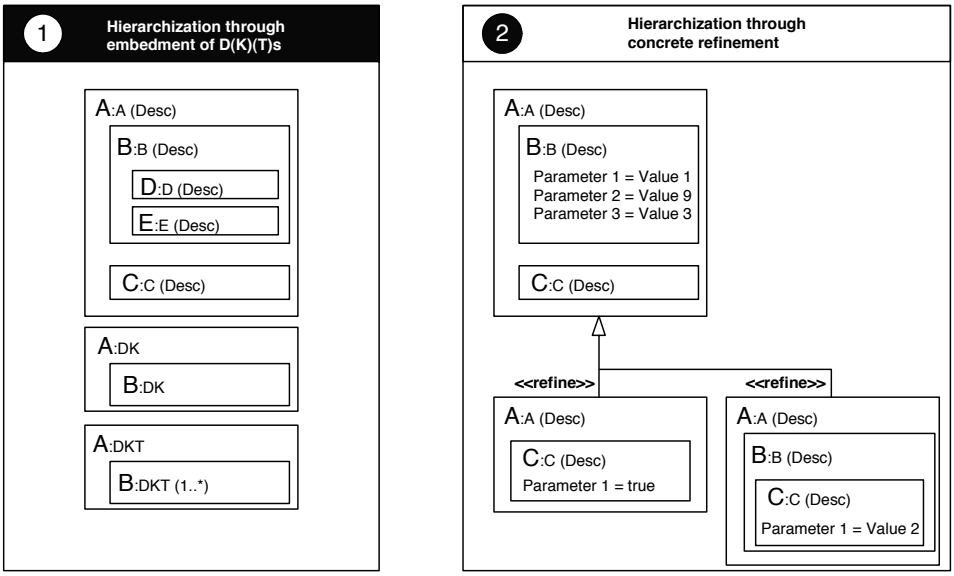

Figure 6: Way of hierarchization

the corresponding DescKit

- Setting values or constraints for the selected parameters
- Proceeding iteratively with the sub-DescKits

The set of Descriptions that are compliant with a certain DescKit is therefore determined by possible allocations of parameters, constraints and embedded Descriptions of a lower hierarchy. A certain fact can be modeled by the domain expert either completely from scratch, or by using pre-modeled Descriptions that can be refined. The DescKit also determines when the process of refining has to come to an end. This is dependent from the individual aim of the modeling project and can be fixed implicitly on layer M^{0*}. The hierarchic approach allows to describe real-world phenomena is models in a restricted way by obeying some guidelines, and furthermore offers the possibility to describe in a domain specific way certain phenomena that are hard to describe in an ordinary way.

4 Exemplary Application of the DKA

The aim of this section is to evaluate the applicability and usability of the DKA. This analysis is an important part of the present research methodology [20, p. 85], [44, p. 734]. To test the DKA in a realistic application one not only need a software implementation of the parts that can be formalized (language model, parts of the procedure model and the

algorithms), but also an organizational implementation of task that cannot be automized, like e.g. the creation of guidelines or the identification of a service.

A major role within the algorithms coming with the DKA plays the model comparison. This task is composed into two steps: A 1:1-mapping algorithm for simple descriptions and the so-called convolution algorithm, which is responsible for preparing complex models for an analysis by "folding" the different parts of a model along the relations into a single (in many cases artificial) description. The convolution results are then used to invoke the 1:1 comparison algorithm. This allows the comparison of arbitrarily complex models. We refer to [22] for an introduction into the algorithms and details on their implementation.

The exemplary scenario used for testing purposes (part of it presented here) contains — untypical for real-world applications — all possible process steps for a service, but also services that bundle the given process steps (by internally calling the finer granular services internally). See Figure 7 for part of the scenario.

It should be checked if the algorithm is able to find a certain service within a set of candidates, which were described using the DKA, and what probably suggested alternatives would be. The first step is a simple test that only uses the 1:1 -mapping algorithm. Then as a second step the convolution algorithm is used within a scenario based on whole process chains.

4.1 Exemplary Application of the Convolution Algorithm

Here we just show a very simplified example of the convolution algorithm. The test process consists of two process steps, described as interfaces using the DKA: "Create Document" (1) and "Send Document" (2), see the left side of Figure 7. The process is modeled in a way that step (1) generates a document for step (2).

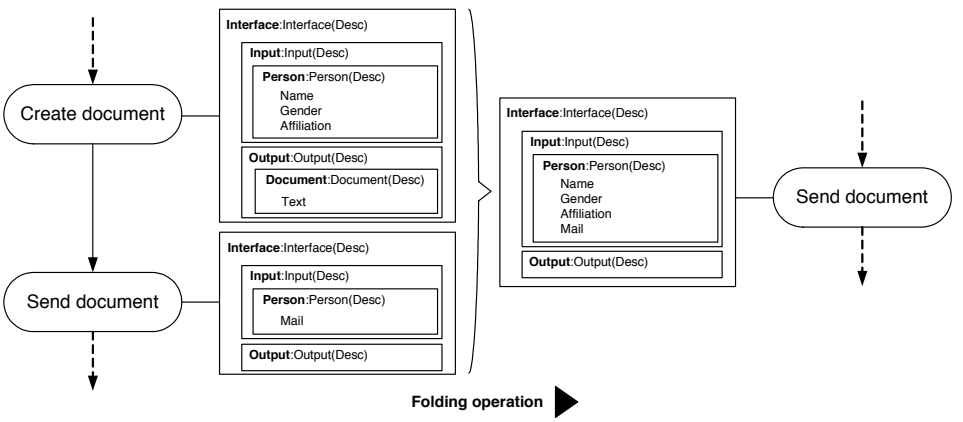

Figure 7: A simple convolution example

The folding algorithm now takes the two Descriptions (that use the DescKit "Interface") and folds them along the relation (process flow) between them. The folding operation operates on the content of the Descriptions (especially embedded Descriptions, parameters, and values). It uses the 1:1-mapping algorithm for all embedded information and is doing the following:

- If two embedded contents match, combine them to one part of the result
- If two embedded contents do not match, embed them separately (as neighbors) into the result
- Intermediary content like here the document that step (1) produces, but step (2) then consumes, are removed
- These steps are done iteratively by proceeding into the depths of the embedding hierarchy

For the actual case the result is shown on the right hand side of Figure 7. All input and output objects are "collected", while intermediary objects vanish from the convolution result. The result resembles now exactly the fact, that the complete process (step (1) and (2) together) "philosophically" simply represents a technical fulfillment of the business requirement of informing a person. It therefore should match similar processes that also fulfill similar requirements. The convolution result is the best foundation for that. It combines the "essence" of all process steps together and represents a single Description "Interface", which can be used then in the 1:1-mapping algorithm to compare it with actual services.

Altogether, the prototypical implementation has shown in this and similar scenarios good results and therefore demonstrates the applicability and usability of the algorithm.

4.2 Exemplary Application of the 1:1-Mapping Algorithm

This section should demonstrate now the central part of the algorithms, namely the 1:1-mapping algorithm. It is used both within the convolution operation (for deciding how to "collect") and after the convolution operation (to compare the convolution results), and therefore essential for realizing good results.

Using some parts of the previous example, Figure 8 shows how the 1:1-mapping algorithm works. The algorithm works in several steps:

1. Comparison of the exterior DescKitType: Matches (both are of DescKitType "Interface")
2. Comparison of the exterior DescKit: matches
3. Comparison of the exterior Description: no match

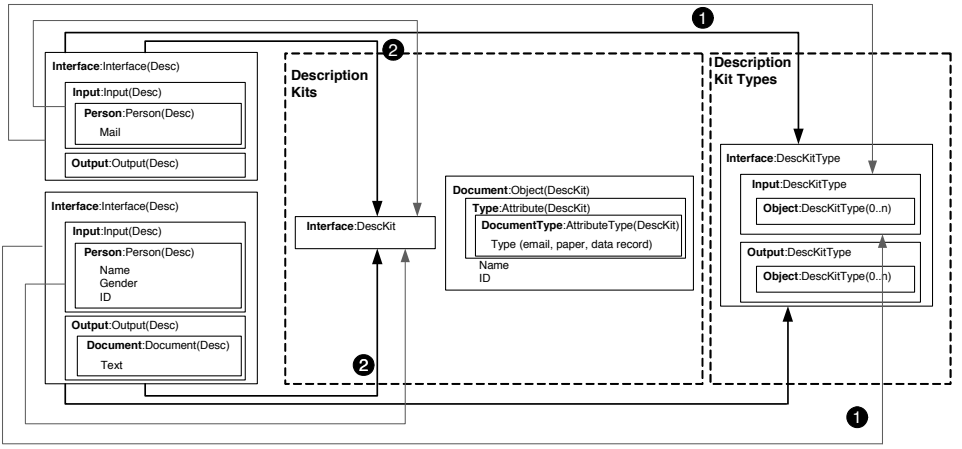

Figure 8: The 1:1-mapping algorithm in action

4. Proceed into the embedding hierarchy: Comparison of the embedded DescKitTypes, DescKits and Descriptions:

   (a) First “Input” and “Output” match; proceed to the next hierarchy level

   (b) “Person” matches with “Person” as DescKitType and as DescKit, while “Document” does not match another Description

   (c) “Person” does not match as a Description, but now the algorithm can compare the set of parameters

   (d) Each parameter again may contain constraints or values, which then can be compared

As can be seen, the algorithm begins with a rough comparison of the outer part of a description. This is of course only giving a slight idea of similarity, but resembles already some part that similar comparison approaches can do. The full strength the DKA unfolds by the hierarchical structure. The mapping algorithm can compare the whole internal structure of a description, which means that e.g. objects can be compared with all their states and dependencies, not only as objects itself.

The present examples show only part of the power of the algorithms, but the general idea should be transparent. The DKA combines a new approach on modeling with powerful comparison algorithms, and therefore can yield much better results than e.g. any attempt of simply standardizing business objects.

5 Evaluation and Summary

The addition (which can be enforced) of DescKits to process models allows on the one hand to use natural language for Descriptions in the sense of DescKits, on the other hand a restriction of the way how to model, and therefore allows to specify parts of the procedure model. The restrictions for using DescKits result in restrictions — in the sense of guidelines — for the ordinary modeling procedure. As a result an approach for the creation of procedure models has been created. For the special case of modeling in the sense of modeling by forming DescKits the creation of procedure models is automized by using the methodology.

5.1 Evaluation

Constraints in conceptual modeling The creation of the DKA has not the aim — in contrast to UML extensions — to include techniques into a modeling method, which allow arbitrary extensions to a modeling language. Quite in contrast, the aim of the DKA is a restriction of the freedom of modeling, without, however, restricting the modeler in his/her freedom to express certain (domain specific) ideas. The creation of DescKits can be seen as creating guidelines in the sense of a restriction of the use of natural languages when describing certain facts without destroying the link to the real world real-world phenomena.

Preventing language conflicts By introducing domain specific DescKits on layer M^{0*} it is possible to prevent homonym conflicts and synonym conflict. The solution of the abstraction conflict is also addressed by the DKA, since guidelines in the form of DescKits partly force the modeler to chose a certain abstraction level when creating Descriptions. In addition, the algorithms, which can compare the structure of Descriptions, may even identify similarities without having existing generalization or specialization relationships between DescKits on layer M^{0*}. By the embedding structure of the DescKits, a presetting is defined that partly can solve separation conflicts and annotation conflicts. Depending on the procedure model, also control flow conflicts may be able to be addressed by the embedding structure of DescKits.

Facilitating consensus finding The consensus finding between domain experts and technical experts is explicitly incorporated into the method. Beforehand, an extension and modification of the descriptive tools was available only by a modification of the meta-model. However, modifications of the meta-model can result in a corruption of existing object models. Therefore, a statical specification of a domain specific language can be problematic. By the introduction of the modeling layer M^{0*} an approach has been found, that makes the extension of a modeling language obsolete. The DKA implements a domain specific language now in an adaptable way. Even during the modeling on object layer the language is still flexible.

Generic algorithms When developing the DKA, a high focus was on the development of

generic algorithms that are controllable on the layer of the DescKitTypes. Therefore they can be used in very different scenarios.

A general problem is still to mark concrete objects within a model. Especially within the service identification scenario it is important for a specification for the propagation of business objects when orchestrating several service functionalities. If for example two processes consecutively process two person objects, which may be described by Descriptions. When composing these two functions, the question is, which person is which. Objects are not instantiations of a certain DescKit, but of a certain Description. However, a Description, even if completely specified, is only a description of an object state. This state can be matched always by a set of concrete objects. A function (or Interface) containing a description for an object only guarantees to work on objects that are in the specified state.

5.2 Conclusion

This article presents new aspects of the DKA that continue and go beyond what has been done in [22]. This includes the introduction of the role and procedure models and an more detailed description of how the different parts of the mapping algorithm apply in practice. It discusses the observed impacts, the difficulty in assessing the modeling language and the role of complementary and contextual factors. Furthermore, special intra-organizational collaboration during the consensus finding process allows to compare this method to classical modeling approaches. The domain expert creates a link from the modeled information to real-world phenomena already during the modeling process by following certain guidelines, represented by Description Kits. Hence, the information contained in these models is prepared for automatic processing. The semantic gap is significantly smaller than in a classical system development process.

Starting from this contribution, we see opportunities for further research in various directions. First, the DKA can only be successful if domain experts and technical experts can be convinced to accept the model-based methodology of the DKA. In the future, the prototypical implementation has to be extended to a full case study to thoroughly prove the applicability and usefulness of the DKA. Especially it has to be shown that the requirements on a model-based method can be met by the DKA better or more cost-efficient compared to ordinary methods. To possibility to combine the DKA with ordinary modeling methods will help to gain a higher acceptance in practice.

References

[1] Aier, S. and Schönherr, M.: Flexibilisierung von Organisations- und IT- Architekturen durch EAI. In: Aier, S. and Schönherr (eds.): Enterprise Application Integration — Flexibilisierung komplexer Unternehmensarchitekturen. Berlin: GITO Verlag, 2004, p. 1–60.

[2] Baker, S. and Dobson, S., Comparing service-oriented and distributed object architectures. In: Proceedings of the International Symposium on Distributed Objects and Applications 3760,

Springer, 2005 LNCS, pp. 631–645.

[3] Balzert, H.: Die Entwicklung von Software-Systemen. Prinzipien, Methoden, Sprachen, Werkzeuge. Mannheim: Bibliographisches Institut, 1982.

[4] Balzert, H.: Lehrbuch der Software-Technik: Software-Entwicklung. Heidelberg: Spektrum Akademischer Verlag, 2001, 2nd ed.

[5] Becker, J., et. al.: Konstruktion von Methodiken: Vorschläge für eine begriffliche Grundlegung und domänenspezifische Anwendungsbeispiele. Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, Arbeitsberichte des Instituts für Wirtschaftsinformatik 77, 2001.

[6] Cervantes, H. and Haller, R. S., Technical Concepts of Service Orientation. In: Stojanovic, Z. and Dahanayake, A. (eds.): Service-Oriented Software System Engineering: Challenges and Practices. Hershey et al.: IDEA Group Publishing, 2005, p. 1–26.

[7] Chmielewicz, K.: Forschungskonzeptionen der Wirtschaftswissenschaft. Schäffer-Poeschel Verlag, 1994.

[8] Erl, T.: Service-oriented architecture: concepts, technology, and design, Prentice Hall PTR (2005)

[9] Ferstl, O. K.; Sinz, E. J.: Grundlagen der Wirtschaftsinformatik. 1, 4, München, Wien: Oldenbourg, 2001.

[10] Frank, U: Zur Verwendung formaler Sprachen in der Wirtschaftsinformatik: Notwendiges Merkmal eines wissenschaftlichen Anspruchs oder Ausdruck eines übertriebenen Szientismus? In: Becker, J., et. al. (eds.): Wirtschaftsinformatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven. Wiesbaden: Gabler Verlag, pp. 127–160 (1999)

[11] Frank, U.: Conceptual Modelling as the Core of the Information Systems Discipline — Perspectives and Epistemological Challenges. In: Haseman, D., et. al. (eds.): Proceedings of the Fifth Americaâs Conference on Information Systems (AMCIS 99). Milwaukee, pp. 695–697 (1999)

[12] Gehlert, A.: Migration fachkonzeptueller Modelle. Berlin: Logos Berlin, 2007.

[13] Greiffenberg, S.: Methodenentwicklung in Wirtschaft und Verwaltung. Hamburg: Dr. Kovac, 2004.

[14] Griffel, F.: Componentware. Konzepte und Techniken eines Softwareparadigmas. dPunkt, Heidelberg (1998).

[15] Gurr, C.; Tourlas, K.: Towards the principled design of software engineering diagrams. In: Proceedings of the 2000 International Conference on Software Engineering, pp. 509–518 (2000).

[16] Gutzwiller, T.: Das CC RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen. Heidelberg: Physica, 1994.

[17] Harel, D.; Rumpe, B.: Modeling languages: Syntax, semantics and all that stuff, part I: The basic stuff. Weizmann Institute Of Science, pp. 1–28 (2000).

[18] Herrmann, H.-J.: Modellgestützte Planung im Unternehmen: Entwicklung eines Rahmenkonzepts. Wiesbaden: Gabler Verlag, 1991.

[19] Heutschi, R.: Serviceorientierte Architektur: Architekturprinzipien und Umsetzung in der Praxis. Berlin, Heidelberg: Springer-Verlag (2007).

[20] Hevner, A. R. et al.: Design Science in Information Systems Research. In: MIS Quarterly **28**(1), pp. 75–105 (2004)

[21] Heym, M.: Methoden-Engineering: Spezifikation und Integration von Entwicklungsmethoden für Informationssysteme, Hochschule St. Gallen für Wirtschafts-, Rechts- und Sozialwissenschaften, Dissertation, 1993.

[22] Juhrisch, M., Dietz, G., Esswein, W.: Perspectives on Semantic Business Process Modeling — A Generic Approach. In: Proceedings of the 13th Pacific Asia Conference on Information Systems (PACIS 2009) (2009).

[23] Juhrisch, M., Dietz, G.: Constraints in Conceptual Modelling — Outlining an Approach to Business Driven Web Service Composition. Int. J. Internet and Enterprise Management **6**(3), 248–265 (2010).

[24] Krafzig, D. et al., Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, 2004.

[25] Kühne, T.: Matters of (Meta-) Modeling. In: Journal on Software and Systems Modeling, 5(4), pp. 369–385 (2006)

[26] Linke, A.; Nussbaumer, M.; Portmann, P. R.: Studienbuch Linguistik. 5th ed., Tübingen: Max Niemeyer Verlag, 2004.

[27] Lorenz, K.: Methode. In: Mittelstrass, J. (ed.): Enzyklopädie Philosophie und Wissenschaftstheorie Band 2. Stuttgart, 1995, pp. 876–879.

[28] Lorenz, K.: Sprache, natürliche. In: Mittelstrass, J. (ed.): Enzyklopädie Philosophie und Wissenschaftstheorie: Sp-Z Band 4. Mannheim, Bibliographisches Institut (1996)

[29] Matthes, F., The impact of SOA on the Enterprise Application Landscape. In: Proceedings of the SOA Days 2005 Business Conference. Bonn: Deutsche Post World Net, 2005.

[30] McGovern, J. et al, Service-Oriented Architecture. In: McGovern, J. et al. (eds.): Java Web Services Architecture. San Francisco: Morgan Kaufmann, 2003, pp. 35–63.

[31] Ortner, E.: Methodenneutraler Fachentwurf: zu den Grundlagen einer anwendungsorientierten Informatik. Stuttgart; Leipzig: Teubner Verlagsgesellschaft, 1997.

[32] Pfeiffer, D.: Constructing comparable conceptual models with domain specific languages. In: Proceedings of the 15th European Conference on Information Systems (ECIS2007), pp. 876–888 (2007)

[33] Pfeiffer, D., Gehlert, A.: A Framework for Comparing Conceptual Models. In: Desel, J., Frank, U (eds.): Enterprise Modelling and Information Systems Architectures: Proceedings of the Workshop in Klagenfurt. Bonn: Köllen Druck + Verlag GmbH, Lecture Notes in Informatics P-75, pp. 108–122 (2005).

[34] Remme, M.: Konstruktion von Geschäftsprozessen: Ein modellgestützter Ansatz durch Montage generischer Prozeßpartikel. Wiesbaden: Gabler, 1997

[35] Schaffner, J.; Meyer, B.: Mixed initiative use cases for semi-automated service composition: a survey. In: Proceedings of the 2006 international workshop on Service-oriented software engineering. Shanghai, China: ACM, pp. 6–12 (2006).

[36] Schütte, R.: Grundsätze ordnungsgemäßer Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle. Neue betriebswirtschaftliche Forschung 233, Wiesbaden: Betriebswirtschaftlicher Verlag Dr. Th. Gabler GmbH, 1998

[37] Searle, J. R.: Speech acts: an essay in the philosophy of language. University Press, Cambrigde, 1969.

[38] Sessions, R., Fuzzy Boundaries. In: ACM Queue, 9 (2004), pp. 40–47.

[39] Sinz, E. J.: Modellierung betrieblicher Informationssysteme: Gegenstand, Anforderungen und Lösungsansätze. In: Pohl, K., et. al. (eds.): Proceedings Modellierung '98, pp. 27–28 (1998).

[40] Stachowiak, H.: Allgemeine Modelltheorie. Wien: Springer Verlag, 1973.

[41] Stahlknecht, P.: Einführung in die Wirtschaftsinformatik. 7. Aufl., Berlin, 1995.

[42] Stojanovic, Z., A Method for Component-Based and Service-Oriented Software Systems Engineering, Delft University of Technology, Dissertation, 2005

[43] Tolvanen, J.-P. (1998) Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence, University of Jyväskylä, Phd thesis.

[44] Verschuren, P., Hartog, R.: Evaluation in Design-Oriented Research. In: Quality and Quantity, 39(6), pp. 733–762 (2005).

[45] Veryard, R., Business Adaptability and Adaptation in SOA. In: CBDi Journal, (2003), November, pp. 15–23.

[46] Vogels, W., Web Services Are Not Distributed Objects. In: IEEE Internet Computing, 7 (2003) 6.

[47] Wand, Y.; Weber, R.: Research Commentary: Information Systems and Conceptual Modeling — A Research Agenda. In: Information Systems Research, 13(4), pp. 363–377 (2002).

[48] Winter, A.: Referenz-Metaschema für visuelle Modellierungssprachen. Wiesbaden, Universität Koblenz-Landau, Dissertation, 2000.

[49] WKWI: Rahmenempfehlung für die Universitätsausbildung in Wirtschaftsinformatik. Wissenschaftliche Kommission Wirtschaftsinformatik, Gesellschaft für Informatik e.V., 2007.

[50] Zelewski, S.: Eignung von Petrinetzen für die Modellierung komplexer Realsysteme: Beurteilungskriterien. In: Wirtschaftsinformatik, 38(4), pp. 369–381 (1996).

[51] Zelewski, S.: Grundlagen. In: Corsten, H., Reiss, M. (eds.): Betriebswirtschaftslehre. 3rd ed, München, pp. 1–125 (1999)

[52] Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design. IBM developerWorks (2004)