Jörg Ackermann, Klaus Turowski

Domain Level Specification of Parameterisable Business Components

Combining software components from different vendors to customer-individual business application systems requires sophisticated techniques to specify the components. If a component can be parameterised, its parameterisation properties must be included in the specification. That is the topic of an ongoing research project. This paper discusses how parameterisation issues can be specified on domain level – that is how to describe parameterisable business terms and business tasks as well as parameterisation effects for a functional expert.

1 Introduction

Many information systems currently in use are monolithic, integrated applications that are hard to maintain and hard to adapt to changing requirements. In contrast to that, component-based or service-oriented business applications have the advantage that some of the required flexibility can be reached by changing system parts (replacing components or services) (Overhage and Turowski 2007). Exchanging components alone, however, can not solve all variability issues because it is not efficient for frequent smaller changes. Using parameterisable business components is an alternative to replacing components. To use parameterisation in component-based business applications is particularly suitable in situations, where a locally restricted functionality shall be adapted, where frequent changes occur or where several adaptation variants are required simultaneously (Ackermann and Turowski 2006). This is the reason why we study parameterisable business components.

A crucial prerequisite for a component-based approach is an appropriate and standardised specification of software components (see Sect. 2). How to include parameterisation options in a component specification was earlier not addressed but is now topic of a research project. This paper discusses how parameterisation related properties can be specified on domain level. Specifying business terms and business tasks on domain level is a prerequisite that functional experts can participate in selecting and evaluating business components (see Sect. 4.1). After introducing an exemplary component (Sect. 3) we discuss the current specification on domain level (Sect. 4) and make some preliminary considerations (Sect. 5). After that we develop detailed proposals how to include parameterisation into component specifications for business terms (Sect. 6) and business tasks (Sect. 7). The paper concludes with an assessment of results (Sect. 8), a discussion of related work (Sect. 9) and a summary (Sect. 10).

This paper makes the following contributions to the area of specification and conceptual modelling of business components: We propose how to specify parameters and parameterisation tasks on domain level. Additionally we show how the effects parameterisation has on business terms and business tasks can be specified. Special attention is given to the description of variable term and task relationships. These results by itself form an important building block for the complete specification of parameterisable business components. Moreover, the results are interesting for any software using parameterisation (like standardised business application suites such as

SAP ERP) because our approach is a first step towards general specification of parameterisation effects.

2 Specification of (Parameterisable) Software Components

An appropriate and standardised specification of software components is a prerequisite for a composition methodology (Overhage 2004) and supports reuse of components by third parties (Conrad and Turowski 2001). With specification of a component we denote the complete, unequivocal and precise description of its external view – that is which services a component provides under which conditions (Turowski 2002). The specification is provided by the component producer and serves as contract between producer and user of a component – for that it must include all information which is necessary to reuse the component.

Currently there exists no generally accepted and supported specification standard covering all aspects relevant to component-based software engineering (see Sect. 8). We base our work on the specification framework 'Standardised Specification of Business Components' (Turowski 2002). The content of this work is a recommendation made by the business components working group of the German Informatics Society (GI). For this Turowski (2002) defines different specification levels, identifies the objects to be specified and proposes for each level a specific notation language:

- The interface (or syntactic) level contains the signature of the components interfaces which are specified by OMG IDL (OMG 2004a) (or since recently also with UML 2.0 (OMG 2004b)). Agreements at behavioural level describe how the component acts in general and in borderline cases and are specified using UML OCL (OMG 2006). Agreements at coordination (or synchronisation) level regulate the sequence in which component services may be invoked. As notation language serves a version of OCL that is enhanced by temporal operators (Conrad and Turowski 2001). Quality attributes and non-functional characteristics (as availability or response time of a service) are specified at the quality level. These four contract levels mainly concern the technical (software) expert and are therefore referred to as technical levels.
- In contrast to that we denote by domain levels those contract levels that concern the functional expert: The terminology level serves as central registry and keeps all used terms and their definitions in a dictionary. The task level specifies which business tasks are supported or automatically done through services of the component. Both levels use normative language (Ortner and Schienmann 1996) as notation. Domain level specification is the focus of this paper.
- Finally the marketing level specifies features that are important from a business-organisational point of view as, e.g., (legal) contract terms or vendor contact persons.

Adaptation is of great importance in component-based application systems because components can rarely be reused without being adapted (Bosch 1997; cf. also Sect. 8). One adaptation technique is the so called (data-based) parameterisation (Ackermann 2004). It is a technique for planned adaptation where the component producer defines parameters (which influence structure and behaviour of the component) and the component consumer chooses parameter settings that are suitable for his requirements. Parameter values are assumed to be data-like and non-executable – in contrast to situations where programs or whole components are expected as parameter values (program-based parameterisation). There are several variants to save parameter values persistently: in database tables, in self-defined files (like text or XML) or in configuration files of a component technology (like property files in OMG CORBA Component Model or deployment descriptors in Suns Enterprise Java

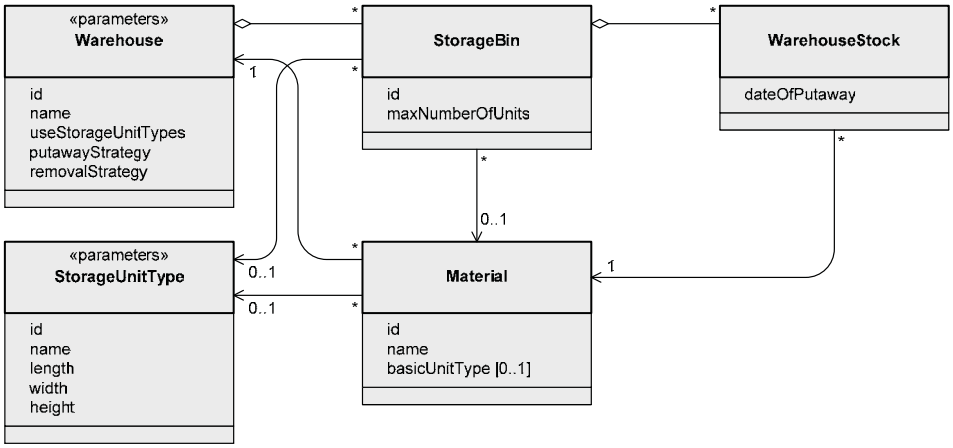

Figure 1: Conceptual data model of component WarehouseManagement.

Beans (EJB)). Main advantages of data-based parameterisation are that adaptation can be easily performed and does not require implementation knowledge because no coding modifications are necessary. One disadvantage is that adaptation is limited to use cases foreseen by the component producer. Another disadvantage stems from the way parameterisation is often used: many software systems like SAP ERP allow complex parameterisation without providing an adequate parameter specification – correlations between parameters and the application functionality become almost impossible to trace (Mertens et al. 1991).

Parameter settings typically change structure and behaviour of a component – that is they influence the component's external view. Consequently parameterisation aspects must be part of a component's specification. This earlier unsolved issue is currently under investigation in our research project.

3 Exemplary Component *WarehouseManagement*

In this section we introduce an exemplary component *WarehouseManagement* that will be used throughout the rest of the paper. To be as realistic as possible design and terminology of our example were influenced by real business application products like SAP ERP (SAP 2005). Note that, however, to be suitable as an example we simplified the component substantially – real applications support, e.g., complex allocation strategies.

The business task of the component is to manage a simple warehouse complex. Figure 1 shows the information objects belonging to the component. The component *WarehouseManagement* allows to define several warehouses which might differ in their warehouse handling (e.g., fixed bin picking area or hall with high rack shelves). Each warehouse consists of different storage bins (storage places) where the goods are physically stored. An entity of *WarehouseStock* represents one unit of a material stored at a specific storage bin. The type *Material* represents the warehouse specific properties of a material. To simplify matters we assume that each material will be stored at exactly one warehouse (real business applications might support complex warehouse determination strategies).

The component allows for parameterisation by providing several parameters (data fields used for parameterisation) as, e.g., *putawayStrategy*. Parameters are typically grouped by parameter groups as, e.g., *Warehouse*. Note that these groups

can have several instances and therefore allow different behaviour in parallel.

The exemplary component has two parameter groups: *StorageUnitType* and *Warehouse*. Storage unit types define the unit size in which materials are stored (e.g., a euro palette or a fixed size box). The organisational unit *Warehouse* offers several control parameters. The parameter *putawayStrategy* (*removalStrategy*) defines how to find a suitable storage bin in which to store (from which to retrieve) a unit of material. To be more precise, the following putaway strategies are supported: each material is assigned to fixed bins (static putaway), the system looks for a suitable bin optimising, e.g., storage space (dynamic putaway) or a storage bin is selected by the operator (manual putaway). The Boolean parameter *useStorageUnitTypes* controls how the size of a storage bin is described. If true, then each storage bin is assigned to the unit type it stores, each material is assigned to the unit type it is delivered in and storage is only allowed if they correspond. If false, then a storage bin can be directly assigned to a fixed material. This means that some of the relationships in Fig. 1 are optional or alternative and their occurrence at runtime is controlled by parameter *useStorageUnitTypes* – these complex dependencies will be used later in the specification of parameterisation effects.

The exemplary component enables to set parameters in *Warehouse* and *StorageUnitTypes* and to manage master data *StorageBin* and *Material*. Furthermore it supports actual stock management activities as storing and retrieving stock and determining the number of available stock units.

4 Current Domain Level Specification for Business Components

This section provides an overview of the current state of specification on domain level – after discussing significance of the domain level (Sect. 4.1) we address specifications for terminology level (Sect. 4.2) and task level (Sect. 4.3). Describing the current state serves two objectives: to foster understanding of the following details and to enable the clear identification of the novel aspects in the results of later sections.

4.1 Relevance and Intention of Domain Level Specification

One important aspect when selecting a business component for an information system is its suitability from a business point of view: Does the component support the required business tasks? Are the restrictions and constraints acceptable? Which business data is supported and what is the relationship between different business terms? The business requirements are usually provided by the business departments – therefore the business adequacy of a component should ideally be verified by functional experts.

Many approaches for component specification either only specify technical aspects (e.g., Beugnard et al. 1999; Han 1998; Hemer and Lindsay 2001) or use technical notations like UML for business aspects (e.g., Cheesman and Daniels 2001). As a consequence such approaches do not adequately support component search and selection (Geisterfer and Ghosh 2006). Without proper domain level specification functional experts can not be involved in component selection or technical experts must 'translate' component properties for functional experts. It is sometimes argued that specifying a component on technical and business level is not efficient due to its partial redundancy. The (one-time) effort for the component producer, however, is small compared to the situation where each prospective component user 'translates' either the component specification for its functional experts or the business requirements into technical terms.

In order to enable functional experts to participate in the component selection one can conclude two requirements: A component specification must contain a description on domain level and the notation used must be suitable for functional experts. Regarding the second aspect it can not

Term: WAREHOUSE STOCK
Short definition: A *Warehouse stock* represents one unit of MATERIAL that is stored at a STORAGE BIN.
Example: One euro palette of material ABC-XYZ stored at storage bin 015-07-02 on October 15th, 2009
Relationships: A WAREHOUSE STOCK has a DATE OF PUTAWAY. A STORAGE BIN consists of 0 to arbitrary many WAREHOUSE STOCK. A WAREHOUSE STOCK is related to a MATERIAL.
Constraints: The MATERIAL related to the WAREHOUSE STOCK must be assigned to the same WAREHOUSE as the WAREHOUSE STOCK.

Figure 2: Specification of term Warehouse stock.

(A | An) *A* consists [in $role_1$] of (a | an | [$number_1$ to] ($number_2$ | arbitrary many)) *B* [in $role_2$] (and (a | an | [$number_3$ to] ($number_4$ | arbitrary many)) *C* [in $role_3$])$^+$.

Figure 3: Complete sentence building pattern for decomposition.

generally be assumed that functional experts understand formal or semiformal (technical) notations like UML.

Therefore the specification framework (Turowski 2002) proposes to specify all relevant business terms and all business tasks supported by the component on domain level (additionally to the technical specification of signatures, protocols etc.). The notation to be used on domain level is normative language, which is an ontology definition language (Ortner and Schienmann 1996). Its idea is to use a standardised form of natural language which is human-understandable, which reduces ambiguities of the natural language and which is also machine-understandable and thus allows for automatic processing.

4.2 Specification on Terminology Level

The task of the terminology level (of the specification framework; Turowski 2002) is to clarify all functional terms on domain level. For this it provides a dictionary of all used terms including their definition and their relationship to other terms. According to Turowski (2002) and Overhage (2004) the specification of a term provides its name, a short definition and an accompanying example in natural language (Fig. 2 shows exemplary how the term *Warehouse stock* is specified). The specification must additionally include the relationship to other terms: An entity of warehouse stock belongs to a storage bin, refers to the material being stored and keeps the date of putaway. Additionally one can supplement constraints the term or its properties need to adhere to.

A central aspect of the term definition is the specification of its relationships to other terms. To specify the relationship between terms four relationship types are predefined:

- Decomposition of a term into other terms (An A consists of a B) – example: a warehouse consists of storage bins.

$Expression_1$ | $Expression_2$	Alternate expressions
[Expression]	Optional expression
(Expression)	Precedence parentheses
(Expression)*	Expression repeated n times ($n \geq 0$)
$(Expression)^+$	Expression repeated n times ($n \geq 1$)
Symbol	Terminal symbol
Symbol	Symbol to be replaced (pseudo terminal symbol)

Figure 4: Notation for syntax of sentence building patterns.

- Association of a term with other terms (An A is related to a B) – example: a material is related to exactly one warehouse (in which it is stored).
- Properties of a term (An A has a B) – example: a storage bin has a maximal capacity.
- Specialisation of a term (An A is a B) – example: a high-rack warehouse is a warehouse.

For each of the relationship types there are corresponding sentence building patterns defined. These patterns must be used to express relationships and thus standardise the relationship specification (Overhage 2004). The sentence building patterns in their simplest form are shown above in parentheses. To express more complicated relationships, there are sentence building patterns for connecting multiple terms and for specifying roles and cardinalities. Figure 3 shows exemplary the complete sentence building pattern for decomposition. Following Overhage (2006) the sentence building pattern in Fig. 3 (and all subsequent patterns) is expressed by symbols of a Backus-Naur form – the used symbols are explained in Fig. 4.

Using sentence building patterns has two advantages compared to natural language: Specifications become less ambiguous und the resulting dictionary structure forms a light-weight ontology allowing an easy way to automatically retrieve relationship information (e.g., find all terms related to a certain term). The advantage of using normative language compared to other ontology notations lies in the fact that it is understandable for functional experts who may not have knowledge of formal modeling or ontology notations (Note that by defining appropriate mappings one can transform normative language expressions automatically into other ontology notations).

The compartment *Constraints* of a term specification shall be used to express restrictions which apply to a term on a business level and which can not be expressed by the predefined sentence building patterns. Constraints are currently specified by natural language – using only predefined sentence building patterns is difficult as the constraints can be rather complex. A more formal representation is desirable in the future and might be achieved by utilising sentence building patterns for first order logic expressions.

4.3 Specification on Task Level

The chore of a business component is to support or execute certain business tasks within a business application system. If a functional expert wants to determine if a component is suitable for his requirements, he must be able to analyse the business tasks supported by the component on a domain (conceptual) level. This shall be done on the task level of the component specification (Turowski 2002).

Task: GET STOCK
Short definition: The task of *Get stock* is to return the number of units of a MATERIAL currently stored within the WAREHOUSE COMPLEX.
Example: 26 euro palettes of material ABC-XYZ currently stored
Relationships: MANAGE STOCK consists of GET STOCK.

Figure 5: Specification of task Get stock.

Term: PUTAWAY STRATEGY
Short definition: A *Putaway strategy* is a strategy which defines how to determine one (or several) STORAGE BINs in which units of MATERIAL will be stored. Supported strategies are: manual, static, dynamic.
Example: Dynamic putaway strategy
Relationships: PUTAWAY STRATEGY is a mandatory parameter. A WAREHOUSE has a PUTAWAY STRATEGY. A PUTAWAY STRATEGY is manual or static or dynamic.

Figure 6: Specification of term Putaway strategy.

According to Overhage (2004) terms and tasks are specified in the same way (both considered as instances of a more general idea concept). Therefore normative language is also used to specify all related business tasks. For a business task its name, a short definition and an accompanying example are given (cf. Fig. 5). If applicable, one can additionally specify relationships between tasks and supplement constraints that need to be adhered to when executing the task.

To specify the relationship between business tasks two relationship types are predefined:

- Decomposition allows describing which subtasks form the task (A consists of B) – example: Manage stock comprises Putaway stock and Remove stock as well as Get stock.
- Specialisation allows distinguishing between different forms (variants) of a task (A is B) – example: Putaway stock manually is a variant of Putaway stock.

For both relationship types corresponding sentence building patterns are predefined and shown above (in their simplest form) in parentheses.

5 Preliminary Considerations

When developing specification proposals for parameterisable business components the following considerations are helpful to structure the task at hand and to enable a structured approach:

- When discussing parameterisation properties one can distinguish between the parameterisation objects (like parameters) themselves and the effects parameterisation has on the components functionality (Ackermann and Turowski 2008) – a specification must consider both aspects.
- For the specification of parameterisation properties one must both identify the specification objects (what to specify) and to make suitable specification proposals (how to specify) (Ackermann 2003).

- Top-level specification objects are parameters and parameter groups on terminology level and parameterisation tasks on task level (Ackermann and Turowski 2008).
- Parameterisation objects have a structure similar to other information objects and in borderline cases it is not always possible to clearly distinguish between them (Ackermann 2003). Therefore (in accordance with the principle of clarity from the Guidelines of Modeling (GoM; Becker et al. 2000) parameterisation objects should be specified similarly to other information objects.
- Parameters need to be clearly recognisable for component users because they often influence component functionality substantially and must be set at configuration time. Moreover, the effects of parameterisation must also be explicitly specified because missing documentation about parameterisation effects complicates the correct use of software (Mertens et al. 1991).

6 Specification on Terminology Level

In this section we extend the specification approach on terminology level (as outlined in Sect. 4.2) such that it also covers parameterisation properties of business components. For that we distinguish (according to Sect. 5) between specification of parameters themselves (Sect. 6.1) and specification of parameterisation effects (Sect. 6.3). Special emphasis will be given to defining sentence building patterns for variable relationship specification (Sect. 6.2).

6.1 Specification of Parameters

In this section we develop a proposal how the terminology associated with parameters (without its runtime effects) can be specified. According to Sect. 5 top-level specification objects are parameters and parameter groups. Parameters and their groups typically have a domain meaning and therefore must be specified on terminology level. Because parameterisation terms have a structure similar to other terms (cf. Sect. 5) we propose to specify parameter and parameter groups similar to other domain terms.

Figure 6 shows how the parameter *Putaway strategy* (of parameter group *Warehouse*) can be specified. Note particularly that parameters are properties of their owning group and therefore a relationship of type 'property' can be used to show the parameter-to-parameter group assignment. As an example Fig. 6 specifies that (parameter group) *Warehouse* has (the parameter) *Putaway strategy*. The values a parameter can take are provided by value declarations – Fig. 6 specifies, e.g., that the parameter *Putaway strategy* can only take predefined fixed values (manual, static, dynamic). Other relationships and constraints are discussed below.

Parameters need to be clearly recognisable for component users (cf. again Sect 5) – therefore we define additional sentence building patterns to identify parameters and parameter groups (cf. Fig. 7). The corresponding pattern for parameters additionally allows specifying if the parameter is mandatory, optional or conditionally optional. The statements built from these patterns are assigned to the *relationships* compartment and in this way extend the collection of statements – as a consequence they are automatically retrievable. The first relationship in Fig. 6 shows examplary how such a pattern is applied and specifies that *Putaway strategy* is a mandatory parameter.

Parameter groups are specified analogously to parameters. Figure 8 shows exemplary the specification of the parameter group Warehouse: All five parameters of Warehouse are listed using a relationship of type property. Relationships with other information objects are specified like any other relationship (here: a warehouse consists of storage bins). Dependencies between different parameters can be expressed as constraints in natural language. As an example the first constraint in Fig. 8 demands that dynamic putaway requires the use of storage unit types.

X is a (mandatory | optional | conditionally optional) parameter.
Y is a parameter group.

Figure 7: Sentence building patterns for parameterisation properties.

6.2 Expressing Variability in Relationships between Terms

We know from domain analysis (Czarnecki and Eisenecker 2000), reference data modeling (Schütte 1998) and an analysis about parameterisation effects on component specifications (Ackermann 2004) that parameters often influence the relationships between terms. Therefore we extend in this section the sentence building patterns (introduced in Sect. 4.2) such that they allow expressing variable participation in relationships and variable cardinalities and roles.

Substantial work on variability was done in domain analysis and we will use their results as starting point. Domain analysis uses feature models to represent 'the common and variable features of concept instances and the dependencies between variable features' (Czarnecki and Eisenecker 2000, p. 86). To express the different variants how a feature can be related to its sub feature(s) six different feature types are defined: mandatory, optional, alternative, optional alternative, (inclusive) or, optional or. We will refer to these variants as variability types. Experience has shown that these variability types are sufficient to model structural variability in domain analysis. Note that the variability type optional or can be normalised to several optional features for composite relationships (like decomposition or association) and is therefore not further considered in domain analysis. For abstractive relationships (specialisation), however, this normalisation is not possible and therefore all six variability types need to be considered.

The current approach for term specification (cf. Sect. 4.2) can only describe must-relationships (mandatory variability type). Therefore it is necessary to extend the sentence building patterns such that they cover the other variability types as well. This will be shown exemplary for decompositions. Solutions for associations and properties are analogous – specialisation needs (because of its abstractive nature) to be treated differently and will be discussed in Sect. 7.2.

The necessary extensions for the sentence building patterns are shown (in its simplest form) in Fig. 9. An optional decomposition is expressed by the additional word optionally in the sentence structure. Decompositions of type alternative, optional alternative and or concern several subterms. To express them we introduce a different sentence structure. Alternative decompositions are described so: 'Exactly one of the following statements is true: an A consists of a B, an A consists of a C.' For optional alternative (or) decompositions it is only necessary to replace exactly by the phrase at most one (at least one). Defining the sentence building pattern in the given way was motivated by two reasons: 1. It is hard to find a combined sentence plan which is understandable and grammatically correct and can be translated into other languages as well – a sentence like 'An A consists of exactly one of a B or a C.' seems rather unnatural. 2. The solution above easily allows combining statements of several relationship types. One could state, e.g., 'Exactly one of the following statements is true: a MATERIAL has a BASIC UNIT TYPE, a MATERIAL is related to a STORAGE UNIT TYPE.' to express, that the unit type of a material is either given by a predefined basic unit type or by an explicitly defined storage unit type.

For easier understanding Fig. 9 shows the sentence building patterns in its simplest form – the complete patterns are depicted in Fig. 10. These patterns additionally allow describing variable

Term: WAREHOUSE
Short definition: A *Warehouse* is a physical or logical subdivision of a WAREHOUSE COMPLEX that is characterised by its warehouse technique, the space used, its organisational form or its function.
Example: Warehouse 001 (High-rack storage Munich)
Relationship: A WAREHOUSE is a parameter group. A WAREHOUSE has a NAME and an ID and a FLAG STORAGE UNIT TYPES and a PUTAWAY STRATEGY and a REMOVAL STRATEGY. A WAREHOUSE consists of 0 to arbitrary many STORAGE BIN. A MATERIAL is related to a WAREHOUSE.
Constraints: For dynamic PUTAWAY STRATEGY the FLAG STORAGE UNIT TYPES must be set to true. For static PUTAWAY STRATEGY the FLAG STORAGE UNIT TYPES must be set to false.

Figure 8: Specification of term Warehouse.

cardinalities or roles in form of an enumeration. To avoid too complex statements it shall not be possible to include variable roles or cardinalities in the expressions for alternative decomposition – instead one can use several expressions. By extending the earlier sentence building patterns to the ones in Fig. 10 we are able to specify decompositions of all variability types and additionally with variable roles and cardinalities. These results will be used in Sect. 6.3 to specify parameterisation effects. Note, however, that the presented extensions are independent from parameterisation and can therefore also be used for variability which is not related to parameters.

6.3 Specification of Parameterisation Effects

In this section we discuss how parameterisation can have an effect on term specification and how these effects shall be specified. Specification objects on terminology level are term definitions, relationships and constraints. If the meaning of a term is variable (depending on a parameter), all variants need to be described in the terms short definition. (For reasons of clarity, however, one should avoid making the meaning of a term parameter dependent. Instead one could use a specialisation and define separate terms for all variants.) Dependencies of a constraint on parameter settings need to be explicitly covered in the constraint. As we use natural language this does not need special techniques – one must only be careful to keep the constraint description understandable.

The relationship specification between different terms can be parameter dependent – variation points are the participation in the relationship itself as well as the cardinalities and roles of the considered terms. The extended sentence building patterns from Sect. 6.2 provide a solution how to express such variability in normative language. These sentence building patterns are so far independent of parameterisation and can generally be used to specify variability. If the variability depends on parameters it is necessary to specify this dependency in detail because missing documentation about parameterisation effects complicate the correct use of software (Mertens et al. 1991).

Mandatory decomposition: (An | A) *A* consists of (a | an) *B*.

Optional decomposition: (An | A) *A* consist optionally of (a | an) *B*.

Alternative decomposition: Exactly one of the following statements holds: (an | a) *A* consist of (a | an) *B*, (an | a) *A* consist of (a | an) *C*.

Optional alternative decomposition: At most one of the following statements holds: (an | a) *A* consist of (a | an) *B*, (an | a) *A* consist of (a | an) *C*.

Or-decomposition: At least one of the following statements holds: (an | a) *A* consist of (a | an) *B*, (an | a) *A* consist of (a | an) *C*.

Figure 9: Variable relationship types and associated sentence building patterns.

To specify how relationship variability is dependent on parameters we follow two approaches in parallel:

- We denote on which parameters the variability depends using predefined sentences. For this we define the additional sentence building pattern in Fig. 11. Corresponding sentence fragments are added to variable relationships (an example is given in the association of Fig. 12).
- We specify in detail which parameter settings result in the occurrence of which relationship variant. Such specifications are included in the constraint section and are done in natural language (an example is also given in Fig. 12).

The detailed specification is necessary because only so the exact effects of parameterisations can be described. As these dependencies can be arbitrarily complex it is quite elaborate to define a complete set of sentence building patterns for them. Therefore we utilise – as for all other constraints – natural language specification. Declaring additionally the parameter dependency using sentence building patterns provides crucial advantages: 1. Parameter dependencies can be automatically retrieved – it becomes possible to search for all effects one particular parameter has. This is a big advantage compared to current business applications where such information can not easily be captured (Ackermann 2004). 2. The specification makes clear if a variable relationship is parameter dependent (this corresponds to so-called build-time operators in reference data modeling; Schütte 1998).

Figure 12 shows how parameter dependent variability is described in the specification of the term Storage Bin. The last relationship specifies that a Storage Bin is either related to a Storage Unit Type or to a Material or to none of them. Which association is allowed is defined by the parameter Flag storage unit types. This dependency is detailed in the constraints where it is specified how the parameter Flag storage unit types (of the Warehouse the Storage bin belongs to) restricts the allowed associations.

To summarise, our approach enables us to express variable relationships between different terms (which frequently occur by parameterisation). Furthermore we explicitly denote on which parameters a variability depends using normative statements – this enables us to automatically retrieve places where parameter dependencies occur. The complete specification how a variability depends on a parameter is done as constraints in natural language – a more formal approach to specify constraints is desirable and a point for future research.

To verify our solution proposal we confirmed that our solution is powerful enough to specify variability as identified by feature types in domain analysis (Czarnecki and Eisenecker 2000) and by so-called build-time operators in reference data modeling (Schütte 1998). Moreover we

Mandatory and optional decomposition with variable cardinalities and roles:

(A | An) *A* consists [optionally] [in $role_1$ (or in $role_2$)*] of (a | an | [$number_1$ to] ($number_2$ | arbitrary many)) (or [$number_3$ to] ($number_4$ | arbitrary many))* *B* [in $role_3$ (or in $role_4$)*].

Alternative, optional alternative, or-decompositions:

(Exactly | At most | Al least) one of the following statements holds: $expression_1$(, $expression_2$)$^+$.

expression = (a | an) *A* consists [in $role_1$] of (a | an | [$number_1$ to] ($number_2$ | arbitrary many)) *B* [in $role_2$]

Figure 10: Complete sentence building patterns for variable decomposition.

Declaration of parameter dependency:

– variability depends on parameter X_1 [of parameter group Y_1] (and on parameter X_2 [of parameter group Y_2])*.

Figure 11: Sentence building pattern for specifying parameter dependencies.

checked that the approach can express all parameterisation effects identified in Ackermann (2004) that are relevant on terminology level and that all terms of the exemplary component Warehouse-Managment could be specified satisfactorily.

7 Specification on Task Level

In this section we propose how parameterisation properties can be specified on task level. For that we distinguish between specification of parameterisation tasks (Sect. 7.1) and specification of parameterisation effects on business tasks (Sect. 7.2). As business tasks are specified quite similarly to business terms (cf. Sect. 4.3) our specification proposals for the task level resemble the ones for the terminology level. Therefore Sect. 7 does not so much introduce new specification proposals but rather shows how to transfer the ones from Sect. 6 to the task level and serves in this way as another proof of concept.

7.1 Specification of Parameterisation Tasks

In this section we develop a proposal how parameterisation tasks (without its runtime effects) can be specified. Top-level specification objects are the parameterisation tasks themselves. Parameterisation tasks are similar to business tasks and therefore we propose to specify them analogously.

Additionally to normal task specification we need to consider two parameterisation-specific aspects (cf. Sect. 5): 1. Parameterisation tasks must be clearly recognisable as such and thus distinguishable from normal business tasks. 2. Specification on task level should include information which parameters are affected by a parameterisation task. It is desirable that both types of information are automatically retrievable. Therefore we propose to specify these aspects in the relationship category and introduce for that the new sentence building patterns shown in Fig. 13.

Figure 14 displays an exemplary task specification for the parameterisation task *Define putaway strategy*. This task sets the related parameters *Putaway strategy* and *Flag storage unit types* of a Warehouse. The specification shows that *Define putaway strategy* is a subtask of the task *Manage warehouses*. Moreover, the constraint category

Term: STORAGE BIN
Short definition: A *Storage bin* (also storage slot) is the smallest available unit of space within a WAREHOUSE that can be separately addressed.
Example: Storage bin 015-07-02 located at lane 015, shelf 07, area 02
Relationship: A STORAGE BIN has an ID and a MAXIMAL CAPACITY. A WAREHOUSE consists of 0 to arbitrary many STORAGE BIN. A STORAGE BIN consists of 0 to arbitrary many WAREHOUSE STOCK. At most one of the following statements holds: a STORAGE BIN is related to a STORAGE UNIT TYPE, a STORAGE BIN is related to a MATERIAL – variability depends on parameter FLAG STORAGE UNIT TYPES of parameter group WAREHOUSE.
Constraints: The number of WAREHOUSE STOCK for the STORAGE BIN must not exceed its MAXIMAL CAPACITY. If FLAG STORAGE UNIT TYPES is true (for the WAREHOUSE the STORAGE BIN belongs to), then the STORAGE BIN is related to a STORAGE UNIT TYPE and is not related to a MATERIAL. If FLAG STORAGE UNIT TYPES is false (for the WAREHOUSE the STORAGE BIN belongs to), then the STORAGE BIN is not related to a STORAGE UNIT TYPE and can be related to a MATERIAL.

Figure 12: Specification of term Storage bin.

demands that the task *Define warehouse* must be performed before *Define putaway strategy*.

7.2 Specification of Parameterisation Effects

Now we discuss how parameterisation can effect task specifications and how to specify such effects. Parameters can influence business task definitions, relationships and constraints. Specification of such parameterisation effects will be analogous to the specification on terminology level. Variations in the definition (again not recommended) or in constraints have to be included into the natural language specification. For variable relationships between different tasks we use again special sentence building patterns. For decomposition of tasks it is possible to use the results from Fig. 10 analogously. Variants in a specialisation are discussed below. If the variability depends on a parameter, this is again denoted using the sentence building patterns from Fig. 11.

Figure 15 depicts the necessary sentence building patterns when applying the six variability types the specialisation relationship. The first pattern describes the special case of a specialisation with only one subtype and corresponds to the mandatory and optional variability types. The second sentence building pattern supports to describe specialisation for the other four variability types. Note that these four variants (deduced from feature types) exactly correspond to the following types of specialisation (in this order): disjoint total, disjoint partial, non-disjoint total, non-disjoint partial. The sentence building patterns in Fig. 15 replace the pattern introduced in Sect. 4.3 because the latter does not clearly distinguish between the different types of specialisation.

As an example Fig. 16 shows how variability in the business task *Putaway Stock* is specified. The component supports three variants (Dynamic

Z is a (mandatory | optional | conditionally optional) parameterisation task.

Z affects (parameter X_1 | parameter X_2 (and parameter X_3)* of parameter group Y_1 | parameter group Y_2) (and (parameter X_4 | parameter X_5 (and parameter X_6)* of parameter group Y_3 | parameter group Y_4))*.

Figure 13: Sentence building patterns for parameterisation properties.

putaway, Static putaway, Manual putaway) how the task can be performed – these variants are specified as subtasks (it is a modeling decision to decide if the differences in task variants are big enough to justify defining separate subtasks). In the relationship in Fig. 16 it is specified that exactly one of the three subtasks is performed when executing *Putaway stock*. Which one will be picked depends on the parameter *Putaway strategy*. By using sentence building patterns it becomes again easy to retrieve the parameterisation dependencies (e.g., which tasks are influenced by a given parameter). The constraint of Fig. 16 describes the strategy finding in detail: the material decides in which warehouse a charge will be stored and the parameter *Putaway strategy* of that warehouse decides which strategy is used.

8 Assessment of Results

The last two sections introduced specification proposals for the parameterisation properties of business components. The correctness, completeness and suitability of these specification proposals were evaluated in two ways:

- An extended analytical validation of the proposals was conducted. As validation criteria served the principles of correctness, relevance, economic efficiency, clarity, comparability and systematic design (taken from the Guidelines of Modeling; Becker et al. 2000). This validation was performed in form of an analysis. Main challenges of this analysis were to extend the GoM from modeling to specification proposals and to ensure the consistency of the approach across several specification levels. By this comprehensive analysis we achieved a validation in a broad sense.
- Two case studies were performed in which parameterisable business components were specified according to the specification proposals. The business component *Warehouse-Management* (an extended version of the one introduced in Sect. 3) was constructed in such a way that it featured a wide variety of potential parameterisation options and effects (as identified in an analysis of parameterisation properties; Ackermann and Turowski 2008). In contrast to that, the second component *Flight-Ticketing* already existed and was in practical use. These two components were completely specified on both domain and technical levels. Based on these component specifications we informally interviewed specification experts (among them some from the industry) regarding the specification approach and results.

The evaluation of the specification proposals (by extended analysis, case studies and expert interviews) yielded the following central findings:

- The specification proposals permit the correct and consistent specification of parameterisable business components. The completeness of the proposals is presumed based on the structured approach while identifying specification relevant parameterisation properties and effects. The aspects of clarity, comparability and systematic design were also considered in the development of the specification proposals.
- A limitation must be noted for the economic efficiency due to the rather large effort necessary for authoring component specifications.

Task: DEFINE PUTAWAY STRATEGY
Short definition: The task of *Define putaway strategy* is to define for a WAREHOUSE which PUTAWAY STRATEGY will be employed and if STORAGE UNIT TYPEs shall be used.
Example: Choose for warehouse 001 (High-rack storage Munich) dynamic putaway strategy and the use of storage unit types.
Relationships: DEFINE PUTAWAY STRATEGY is a mandatory parameterisation task. MANAGE WAREHOUSES consists of DEFINE PUTAWAY STRATEGY. DEFINE PUTAWAY STRATEGY affects parameters PUTAWAY STRATEGY and FLAG STORAGE UNIT TYPES of parameter group WAREHOUSE.
Constraints: The task DEFINE PUTAWAY STRATEGY requires that the task DEFINE WAREHOUSE has been performed for the considered WAREHOUSE.

Figure 14: Specification of task Define putaway strategy.

This can be illustrated by the length of the textual specification of component *Warehouse-Management*: 10 pages for the domain levels and 36 pages for all specification levels together. It shall be noted, however, that this is not specific for parameterisable components but a general issue for the complete specification of black-box components.

- Both case studies supported these findings: the specification approach permitted the complete and satisfactory specification of the components but required high effort. One finding of the case studies is particularly noteworthy: the high effort is not only caused by inherent complexity of parameterisation but also (in our examples even more) by a high number of rather trivial conditions (like specifying value range and optionality of parameters or other data fields). Note that omitting these conditions would contradict the completeness requirement.
- The interviews yielded similar results: the specification approach was judged as reasonable and the specification results as satisfactory and understandable. Concerns, however, were raised by the high effort to produce (and to consume) the specifications. Improvements in the efficiency were seen as a prerequisite for a wide-spread adoption in practice. Possible directions for improvement include better support by specification tools and the utilisation of specification patterns.

It should be noted that the introduced specification proposals are intended for (and appropriate for) business components with normal parameterisation complexity. This focus is justified by the results of an analysis that determined in which adaptation scenarios the use of parameterisation is suitable (Ackermann and Turowski 2006). Technically it might also be possible to apply our approach to big, highly variable components or even to whole application systems like SAP ERP (although this is not the focus of the paper). For such cases with highly complex parameterisation, however, additional techniques for efficient specification will be necessary because our approach by itself will not be economic in such large scale.

9 Related Work

Component-based software engineering requires the precise specification of software components. Currently there exists no generally accep-

Mandatory and optional specialisation:
A is [optionally] C.

Specialisation with variable subtypes:
(Exactly one | At most one | At least one | Any number) of the following variants hold: *C* is *A* (or *B*)$^{+}$.

Figure 15: Sentence building patterns for variable types of specialisation.

ted and supported specification standard covering all relevant aspects. Various authors addressed specifications for specific tasks of the development process as, e.g., design and implementation (Cheesman and Daniels 2001; D'Souza and Wills 1998), component adaptation (Yellin and Strom 1997) or component selection (Hemer and Lindsay 2001). Approaches towards a comprehensive specification of software components are few and include Han (1998), Beugnard et al. (1999), Turowski (2002) and Overhage (2004). Its consideration of technical and domain aspects in one unified proposal is the main advantage of Turowski (2002) and Overhage (2004). Parameterisation aspects are not discussed in the literature about component specification.

Adaptation is an important aspect of component-based software engineering because in practice components can rarely be reused without being adapted (Bosch 1997). Consequently adaptation in component-based systems is discussed by many authors – for an overview and a comparison of different techniques see, e.g., Bosch (1997) or Reussner (2001). Parameterisation is identified as an adaptation technique by many authors, but not discussed in detail. Specification aspects are not covered in the literature about component adaptation.

Integrated standard application suites like SAP ERP allow complex parameterisation (also called *customising*) which is often discussed in the literature. Many authors focus on process models and high-level configuration of business processes (for SAP cf., e.g., Keller and Teufel 1998; Appelrath and Ritter 2000). The detailed description of parameter settings, however, received less attention. The quality of parameter documentation is often not sufficient – correlations between parameters and the components functionality become almost impossible to trace (Mertens et al. 1991). Several works contain detailed recommendations for parameter settings that are specific for a software suite and a functional area (e.g., Dittrich et al. 2006 for production planning with SAP).

A general approach towards specification of parameters and parameterisation effects, however, does not exist.

Parameter settings typically influence the components external view and must therefore be considered in a component specification. This earlier unsolved issue was investigated in our research project. This paper builds on earlier project results on parameterisable business components in general and on technical level specification (Ackermann 2003, 2004; Ackermann and Turowski 2008) and extends earlier results (Ackermann and Turowski 2007).

Variability is an important issue in software engineering and, e.g., relevant for configurable reference models (Becker et al. 2004; Schütte 1998), software reuse (Jacobson et al. 1997), generic programming (Czarnecki and Eisenecker 2000) and the related product-line approach (Bosch et al. 2001). Their results form a methodical foundation of our work.

The communication problems between information system users and system developers are

Task: PUTAWAY STOCK
Short definition: The task of *Putaway stock* is to find suitable STORAGE BINs for a number of units of MATERIAL and store them there physically.
Example: Store one euro palette of material ABC-XYZ – system assigns storage bin 015-07-02
Relationships: Exactly one of the following variants holds: PUTAWAY STOCK is MANUAL PUTAWAY or STATIC PUTAWAY or DYNAMIC PUTAWAY – variability depends on parameter PUTAWAY STRATEGY of parameter group WAREHOUSE.
Constraints: The parameter PUTAWAY STRATEGY (of the WAREHOUSE the MATERIAL is stored in) decides which task to perform: If this parameter is set to DYNAMIC (STATIC / MANUAL) PUTAWAY STRATEGY, then DYNAMIC (STATIC / MANUAL) PUTAWAY is performed.

Figure 16: Specification of business task Putaway stock.

discussed in Ortner and Schienmann (1996). As solution they propose to communicate via normative language, which is an ontology definition language that is both machine and human-understandable and reduces the ambiguity of the natural language. The component specification approach Turowski (2002) proposes to use normative language for the component specification on domain level.

10 Summary and Outlook

The paper discussed the specification of parameterisable business components on domain level. We proposed how to specify parameters and parameterisation tasks and we showed how parameterisation effects on business terms and business tasks can be described. For that we extended the existing normative language for component specification in such a way that it can be used to describe parameterisation properties (in a way that allows for automatic retrieval of this information) and that enables to specify variable relationships between terms and tasks.

The results of this paper form an important building block for the complete specification of parameterisable business components. They additionally foster variability specification on domain level independent from parameterisation. Moreover, the results of this paper are also interesting for any software using parameterisation because our approach is a first step towards general specification of parameterisation effects – a so far unsolved issue.

Topics for future research include an investigation if the constraints on domain level can be completely specified using only predefined sentence building patterns and the development of the necessary patterns. Another interesting issue is the question which role parameterisation plays in the adoption of software services and (if relevant) how parameterisable software services can be specified.

References

Ackermann J. (2003) Specification Proposals for Customizable Business Components. In: Overhage S., Turowski K. (eds.) Proceedings 1st International Workshop Component Engineering Methodology. Erfurt, pp. 51–62

Ackermann J. (2004) Zur Beschreibung datenbasierter Parametrisierung von Softwarekomponenten (in German). In: Turowski K. (ed.) Architekturen, Komponenten, Anwendungen – Proceedings zur 1. Verbundtagung Architekturen, Komponenten, Anwendungen (AKA 2004). Augsburg, pp. 131–149

Ackermann J., Turowski K. (2008) Zentrale Gegenstände der Parametrisierung bei betrieblichen Softwarekomponenten (in German). In: Dinter B., Winter R., Chamoni P., Gronau N., Turowski K. (eds.) Synergien durch Integration und Informationslogistik. Proceedings zur DW 2008. St. Gallen, pp. 509–528

Ackermann J., Turowski K. (2006) Zur Rolle von Parametrisierung bei der fachlichen Anpassung betrieblicher Softwarekomponenten. In: Schelp J., Winter R., Frank U., Rieger B., Turowski K. (eds.) Integration, Informationslogistik und Architektur – Proceedings zur DW2006. Friedrichshafen, pp. 341–359

Ackermann J., Turowski K. (2007) On the Specification of Parameterizable Business Components. In: Draheim D., Weber G. (eds.) Trends in Enterprise Application Architecture (TEAA 2006). Springer, Berlin, pp. 25–39

Appelrath H.-J., Ritter J. (2000) SAP R/3 Implementation: Methods and Tools. Springer, Berlin

Becker J., Rosemann M., von Uthmann C. (2000) Guidelines of Business Process Modelling. In: van der Aalst W., Desel J., Oberweis A. (eds.) Business Process Management: Models, Techniques and Empirical Studies. Springer, Berlin, pp. 30–49

Becker J., Delfmann P., Knackstedt R. (2004) Konstruktion von Referenzmodellierungssprachen – Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle. In: WIRTSCHAFTSINFORMATIK 46(4), pp. 251–264

Beugnard A., Jézéquel J.-M., Plouzeau N., Watkins D. (1999) Making Components Contract Aware. In: IEEE Computer 32(7), pp. 38–44

Bosch J. (1997) Adapting Object-Oriented Components. In: Proceedings of the 2nd International Workshop on Component-Oriented Programming (WCOP '97). Turku, Finland

Bosch J., Florijn G., Greefhorst D., Kuusela J., Obbink H., Pohl K. (2001) Variability Issues in Software Product Lines. In: Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4). Springer, Bilbao, pp. 13–21

Cheesman J., Daniels J. (2001) UML Components. Addison-Wesley, Boston

Conrad S., Turowski K. (2001) Temporal OCL: Meeting Specification Demands for Business Components. In: Siau K., Halpin T. (eds.) Unified Modeling Language: Systems Analysis, Design and Development Issues. Idea Group, Hershey, pp. 151–165

Czarnecki K., Eisenecker U. (2000) Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston

D'Souza D., Wills A. (1998) Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison-Wesley, Reading

Dittrich J., Mertens P., Hau M., Hufgard A. (2006) Dispositionsparameter in der Produktionsplanung mit SAP: Einstellhinweise, Wirkungen, Nebenwirkungen (in German), 4th ed. Vieweg, Wiesbaden

Geisterfer C., Ghosh S. (2006) Software Component Specification: A Study in Perspective of Component Selection and Reuse. In: Proceedings of the 5th International Conference on COTS-Based Software Systems (ICCBSS'06). IEEE Computer Society Press, Orlando, pp. 100–108

Han J. (1998) A Comprehensive Interface Definition Framework for Software Components. In: Proceedings of 1998 Asia-Pacific Software Engineering Conference. Taipei

Hemer D., Lindsay P. (2001) Specification-based retrieval strategies for module reuse. In: Grant D., Sterling L. (eds.) Proceedings 2001 Australian Software Engineering Conference. IEEE Computer Society, Canberra, pp. 235–

243

Jacobson I., Griss M., Jonsson P. (1997) Software Reuse. ACM Press / Addison Wesley Longman, New York

Keller G., Teufel T. (1998) SAP R/3 Process-oriented Implementation. Iterative Process Prototyping. Addison Wesley Longman, Harlow

Mertens P., Wedel T., Hartinger M. (1991) Management by Parameters? (in German). In: Zeitschrift für Betriebswirtschaft 61(5/6), pp. 569–588

OMG (2004a) Common Object Request Broker Architecture: Core Specification. Version 3.0.3, 2004-03-12. http://www.omg.org/

OMG (2004b) Unified Modeling Language: Superstructure. Version 2.0, formal/05-07-04. http://www.omg.org/technology/documents

OMG (2006) Object Constraint Language. Version 2.0, formal/06-05-01. http://www.omg.org/technology/documents

Ortner E., Schienmann B. (1996) Normative Language Approach: A Framework for Understanding. In: Thalheim B. (ed.) Conceptual Modeling. ER '96, 15th International Conference on Conceptual Modeling. Springer, Berlin, pp. 261–276

Overhage S. (2004) UnSCom: A Standardized Framework for the Specification of Software Components. In: Weske M., Liggesmeyer P. (eds.) Object-Oriented and Internet-Based Technologies, Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (NOD 2004). Erfurt

Overhage S. (2006) Vereinheitlichte Spezifikation von Komponenten – Grundlagen, UnSCom Spezifikationsrahmen und Anwendung. Ph.D. Thesis, Universität Augsburg, Augsburg

Overhage S., Turowski K. (2007) Service-orientierte Architekturen – Konzept und methodische Herausforderungen. In: Nissen V., Petsch M., Schorcht H. (eds.) Service-orientierte Architekturen. Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen. Deutscher Universitätsverlag, Wiesbaden, pp. 3–17

Reussner R. (2001) The Use of Parameterised Contracts for Architecting Systems with Software Components. In: Proceedings of the 6th International Workshop on Component-Oriented Programming (WCOP 2001). Budapest

SAP (2005) SAP Implementation Guide (IMG). In: Online Documentation for SAP ERP, Release 6.0. Walldorf

Schütte R. (1998) Grundsätze ordnungsmäßiger Referenzmodellierung. Gabler, Wiesbaden

Turowski K. (ed.) Standardized Specification of Business Components: Memorandum of the working group 5.10.3 Component Oriented Business Application System. University of Augsburg, Augsburg. http://www.fachkomponenten.de

Yellin D., Strom R. (1997) Protocol Specifications and Component Adaptors. In: ACM Transactions on Programming Languages and Systems 19, pp. 292–333

Jörg Ackermann, Klaus Turowski

Chair of Business Informatics and Systems Engineering
University of Augsburg
Universitätsstr. 16
86159 Augsburg
Germany
{joerg.ackermann | klaus.turowski}@wiwi.uni-augsburg.de