Ontology Support for Configurative Reference Modeling

Joerg Becker
ERICIS, becker@ercis.uni-muenster.de

Stefan Fleischer
University of Muenster, stefan.fleischer@ercis.uni-muenster.de

Ralf Knackstedt
University Muenster, ralf.knackstedt@ercis.uni-muenster.de

Armin Stein
University of Muenster, armin.stein@ercis.uni-muenster.de

Follow this and additional works at: http://aisel.aisnet.org/ecis2008

Recommended Citation
http://aisel.aisnet.org/ecis2008/135

This material is brought to you by the European Conference on Information Systems (ECIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ECIS 2008 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Abstract

The manual customization of reference information models to suite special purposes is an exhaustive task that has to be accomplished thoroughly to preserve, expatiate and extend the inherent intention. This can be facilitated by the usage of automatisms like those being provided by the configurative reference modeling approach. Thus, the reference information model has to be enriched by data describing for which application scenario a certain reference model element is relevant. By assigning this data to application contexts, this structure builds a taxonomy. This taxonomy can be extended by relationships between the elements, thus forming an ontology. The paper illustrates the advantage of the usage of such ontologies during three phases of the lifecycle of configurative reference models. Furthermore, algorithms for the evaluation of the ontology and the deduction of the resulting effects are presented. Finally, the impact of the usage of the approach to support the phases of creating and using configurative reference models is demonstrated by using a software tool for reference modeling.

Keywords: Ontology Support, Configurative Reference Models, Tools, Model Lifecycle.
1 CONFIGURATIVE REFERENCE MODELING AS A FIELD FOR ONTOLOGY APPLICATION

Reference information models—in the context of this paper solely called reference models—give recommendations for the structuring of information systems as best or common practices and can be used as a starting basis for the development of application specific information system models. The better the reference models are matched with the special features of individual application contexts, the greater the benefit of the reference model’s usage. Reference models are representations of knowledge recorded by domain experts to be used as guidelines for everyday business as well as for further research. They should be of general validity in terms of being applicable for more than one user (see Schütte, Rotthow (1998); vom Brocke (2003); Fettke, Loos (2004)). General applicability is thus a necessary requirement for any model to be characterized as reference model, as it has to grant the possibility to be adopted by e.g. different companies or even different users. Thus, the reference model has to include information about different business models, different functional areas or different purposes for its usage, e.g. information for different target audiences. A reference model for retail companies might cover economic levels like Retail or Wholesale, trading levels like Inland trade or Foreign trade as well as functional areas like Sales, Production Planning and Control or Human Resource Management. Furthermore, best or common practice information in terms of reference data models, reference process models or reference organization plans might be included (cf. Becker, Schütte (2007)). While this constitutes the general applicability for a certain domain, one special company usually needs just one suitable variant—a subset—of this reference model, for example Retail/Inland Trade, leaving the remaining information dispensable. This yields the problem that the perceived demand of information for each individual user will be hardly met by the complete reference model or by one single variant. Either the information delivered—in terms of models of different modeling languages which might consist of different element types and hold different element instances—is not sufficient or it is too extensive. Hence the person addressed by the model will be overburdened on the one hand or insufficiently supplied with information on the other hand. To compensate this in a conventional manner, a complex manual customization of the reference model is necessary to meet the individual users’ demands. Another implication is the maintenance of the reference model: Each time changes are committed to the reference model due to e.g. new findings or a finer detailing, every model variant has to be manually updated as well.

This is the point where configurable reference models come into operation. The basic idea of an approach by Becker, Delfmann, Knackstedt (2007) is to attach parameters to elements of the reference model in advance, defining the contexts to which these elements are relevant. The user eventually selects a set of best suited parameters for his purpose and the respective configured model variant is generated automatically. However, this approach doesn’t take into consideration that certain parameters might include or exclude others. It is thus possible that an inconsistent choice of parameters is taken. This paper fills this gap. By defining relations between the parameters, an ontology supporting the creation and configuration of a configurable reference model can be developed. This ontology can be evaluated such that consistent model variants can be derived from the reference model.

2 RELATED WORK

The usage of ontologies in combination with the creation of conceptual information models in general and reference information models in particular is not new to the field of IS research. However, in the majority of cases ontologies are used to semantically extend existing modeling languages. For the domain of ERP customizing, Soffer, Golany and Dori (2003) specify the context of an application scenario of the reference model by instantiating several attributes, which define the resulting model variant. Rosemann and van der Aalst (2007) or La Rosa et al. (2007) extend the eEPC (extended event-driven process chain) by creating rules that formalize dependencies between model elements. Thus, when
removing certain elements from the model, the user gets instructions on how this affects other dependent model areas. The Semantic EPC of Thomas and Fellmann (see Thomas, Fellmann (2007)) uses an ontology to enhance the semantics inherent to the model elements by formal specification. This should firstly enable and secondly simplify the comprehensibility of the model for human beings as well as for machines. Besides this, the authors motivate the possibility to deduce additional information about the elements by enriching them in the meta model with connections to existing domain specific ontologies. Another example is the semantically enhanced BPMN of Abramowicz et al. (2007). Another area of interest concerning ontologies in the IS domain is their development. A prominent example for the latter case is the BWW (Bunge-Wand-Weber) ontology (Wand, Weber (1995)), an attempt to adjust the original all-embracing ontology of Bunge (cf. Bunge (1979)) to suite the IS requirements.

Apparently, to enhance existing models with information inherent to ontologies, the ontology which is used has to be developed in the first way, thus involving a two-step-procedure – the creation and the application. This paper motivates a scenario suitable for both steps. Here, ontologies are not only used to improve the comprehensibility of reference models but also to support their configuration. The ontology, of which one is exemplary presented in this paper, contains rules that describe how different application specific variants can be derived. Each of these rules consists of a condition and an implication. Each condition describes one application context of the reference model. The respective implication determines the relevant model variant. For describing the application contexts, configuration parameters are used. These can be seen as dependent to each other. The choice of one parameter might lead to the exclusion of another one, while the choice of a third one might require another specific parameter.

The analysis of the usefulness of ontologies for configurative reference modeling is based upon two parts: The creation and usage of reference models that are configurable and their lifecycle on the one hand and the creation of the supporting ontology on the other. Thus, the paper is structured as follows: Section 3 gives an overview over the two pillars the framework is based upon: The phases of the lifecycle of configurative reference models and ontologies. Section 4 analyses the usefulness of ontologies during three phases identified as being relevant for enabling the configuration process. This is done textual and formally by using pseudo code examples. To demonstrate the feasibility and applicability of the approach presented, a prototypical tool implementation is shown in section 5. The paper concludes with an outlook on further research areas (section 6).

3 ANALYSIS FRAMEWORK FOR ONTOLOGY USAGE DURING THE CONFIGURATIVE REFERENCE MODEL LIFECYCLE

3.1 Lifecycle phases of configurable reference models

The lifecycle of configurable reference models can be divided into two parts called Development and Usage (see Schlagheck (2000), see also figure 1). The first part—relevant for the reference model developer—consists of the phases Project Aim Definition, Model Technique Definition and Model Construction and Evaluation, the second one—relevant for the user—includes the phases Project Aim Definition, Search and Selection of existing and suitable reference models and Model Configuration. The resulting configured model can, but doesn’t need to be further adapted to satisfy individual information needs afterwards (see Becker et al. 2004). Several phases can be identified, where the application of ontologies can be of value, especially Project Aim Definition and Model Construction (for the developer) and Model Configuration (for the user). Those phases were chosen because either the creation of the ontology relevant for a certain reference model takes place, or its configuration using the ontology. Although there are application areas during the other phases, they are not yet regarded as being as effective as during the chosen ones. Thus these phases are not yet considered as being relevant to the approach presented; however, future research will concentrate on this matter as well.
Figure 1 gives an overview of the whole lifecycle, where the phases that will be discussed in detail are solid, the ones actually not considered are grayed out. The output of both parts Development and Usage is printed in italics. Furthermore, this figure serves as a guideline for the analysis framework of the application and usability of ontologies in the context of this paper.

![Figure 1: Lifecycle of configurable reference models](image)

3.2 Ontologies and their relationship types in the context of this paper

Gruber (1993) defines ontology as “an explicit specification of a conceptualization”. As such, everything that exists can be related to other things in a certain manner. Hence, a lot of relations can be set between the elements of an ontological system. However, the exemplary ontology presented in this paper uses only three very basic relations between the elements (see Figure 2), relating to Guarino (1998), stating that in the simplest case, an ontology describes a hierarchy of concepts (here: characteristics, parameters) related by subsumption relationships.

While many different element types and relationship types between particular elements exist to describe any ontology, we focus on configuration characteristics and configuration parameters as elements and the relations Is-instance-of, Excludes and Implicates. The relationship type Is-instance-of is used to construct a set of configuration parameters where each is instance of a certain configuration characteristic, e.g. the parameter Catalog is instance of the characteristic SalesContactForm. Furthermore we will use an ontology to describe coherences between configuration parameters that may exclude or implicate each other. Such exclusion and implications may be called the rules of an ontology, because they restrict possible combinations of selected parameters. Figure 2 exemplarily demonstrates these three relationship types, Is-instance-of, Excludes and Implicates, between the objects of interest, namely configuration characteristics and configuration parameters. As you can see in Figure 2, VendingMachine and InvestmentGoodsTrade exclude each other, as investment goods can hardly be bought via a vending machine. If the SalesContactForm is SelfService and/or SalesPerson, also the PurchaseInitiationThroughVisitToStore has to be selected, because there is a physical contact between the customer and the location items purchased—the store.

In case that ContactOrientation is MailOrder, customers should be able to order items via Letter, Fax and Internet. The VerticalCooperation can be two or all of the three peculiarities Retail, Wholesale and IndustrialCompanies. These peculiarities are bundled to certain configuration parameters excluding each other to ensure that a minimum of two peculiarities is selected.
4 APPLYING THE ANALYSIS FRAMEWORK

4.1 Phase 1: Project Aim Definition

During the first phase of the lifecycle, the developers have to agree upon the purpose of the reference model to build. They have to decide for which domain the model should be used, which business models should be supported, which functional areas should be integrated to support the distribution for different perspectives and so on. Initially, all parameters for each possible characteristic have to be listed. By selecting the relevant parameter for the reference model, the developers commit themselves to one common project aim and reduce the given complexity. Thus, the emerging combination of characteristics and parameters constitutes a taxonomy, implying the variants included in the integrated configurative reference model. By generating this taxonomy, the developers get aware of all possible included variants. One special variant of the model will later on be generated by choosing one or a set of the parameters by the user.

The choice of parameters should be controlled and supported by the underlying ontology, thus the developers have to decide whether or not dependencies between parameters exist. In some cases, the choice of one specific parameter within one specific characteristic determines the necessity of another parameter within another characteristic. Exclusions should not be considered yet, as the developer might need to integrate conflicting information, e.g. for a reference model that includes both model variants for the parameters *Vending machine* and *Investment goods trade*. However, exclusions and implications must not contradict each other, what still has to be considered.

To select a certain set of parameters, two approaches can be pursued—at the same time if applicable: Firstly, choosing some parameters (and ontology rules implicitly) from a pre-defined set and/or secondly defining new parameters not given yet. In the first case of choosing existing parameters, the concrete buildup has to satisfy the underlying ontology—if one exists already. The pre-defined set of parameters might have been created earlier during another instance of Project Aim Definition. At this
point only the implication rules have to be considered. Exclusion rules will be ignored since a reference model may contain information about different business models, different functional areas and different purposes. In the second case of defining new parameters there is no ontology to check the parameters against. But a new ontology can be created as well. In case of combining the two approaches the eventually existing ontology may be extended. Further implication and exclusion rules must not contradict the ontology with regard to its existing rules. So, exclusions may only be ignored selecting/defining certain configuration parameters but have to be considered validating the ontology’s consistency.

In the following some pseudo-code snippets are presented implementing the postulated ontology functionality. Several so called SAT solvers exist (van Maaren, Franco (2007)) providing such functionality, but they come along with wide generality which is not needed here. To dispense with the generality of existing SAT solvers does not mean a seriously overhead self-implementing the algorithms needed as the latter proved to be very simple and easy to implement. It is rather an edge to do only the computations really needed considering special case information we have. The following pseudo-code validates a set of rules—if there are no conflicting rules and it is thus valid:

```plaintext
bool ValidateRuleSet( Set<Rule> rules )
foreach Exclusion rule in rules
    Set<Rule> rules' := rules without rule
    if ImplicationPathExists( rule.cp1, rule.cp2, rules' )
        return false
    return true

bool ImplicationPathExists( CP cp1, CP cp2, Set<Rule> rules )
foreach Implication rule in rules
    if not rule.cp1 = cp1
        continue
    if rule.cp2 = cp2
        return true
    Set<Rule> rules' := rules without rule
    if ImplicationPathExists( rule.cp2, cp2, rules' )
        return true
return false
```

The function ValidateRuleSet checks if there are no exclusion rules contradicting any implication rule(s). It would not be sufficient to only check single implication rules against all exclusion rules because there may be transitive paths of implication rules, e.g. \( A \Rightarrow B \Rightarrow C \) (\( A \) implicates \( B \), \( B \) implicates \( C \)) but \( A \leftrightarrow C \) (\( A \) and \( C \) exclude each other). ValidateRuleSet iterates the set of rules and simply calls another function ImplicationPathExists for each exclusion rule on the two particular configuration parameters the exclusion rule relates and a shortened set of remaining rules to check against. ImplicationPathExists returns true if the set of rules passed in holds a path \( r_{(1)}, \ldots, r_{(n)} \) of implication rules with \( r_{(1)} = cp_1 \) and \( r_{(n)} = cp_2 \) as well as \( r_{(k)} = r_{(k-1)}, k = 2 \ldots n \), where \( r_i \) is a rule from the set, \( r_i^1 \) and \( r_i^2 \) are the first and second configuration parameters of rule \( r_i \), respectively, and \( cp_1 \) and \( cp_2 \) are the two configuration parameters passed in. Therefore, ImplicationPathExists searches the set of rules for implication rules and quickly returns true if the conditions \( r_i^1 = cp_1 \) and \( r_i^2 = cp_2 \) are met. If not so, the function recursively calls itself. If the whole set has been iterated and the function has not returned true, the function result is false. Finally ValidateRuleSet returns true if no contradictions were found, false otherwise. So—to ensure a valid set of rules—the function ImplicationPathExists must return false for each exclusion rule. ValidateRuleSet can also be applied on a just extended rule set to check if the set remains valid after adding the new rule(s).

As mentioned above during Project Aim Definition only the consideration of implication rules takes place to validate the set of selected configuration parameters (provided that the set of rules is valid for itself). As a matter of principle all selected configuration parameters must satisfy all implication rules that there are no configuration parameters \( A \) selected and \( B \) not selected with \( A \Rightarrow B \) (\( A \) impli-
cates $B$). Even more formal it can be said that $(A \Rightarrow B) \iff ((A \notin \Omega) \lor (B \in \Omega))$, where $\Omega$ represents the set of selected configuration parameters. The following pseudo-code illustrates how this can be realized:

```plaintext
bool ValidateCPSetAgainstImplications( Set<CP> cps, Set<Rule> rules )
    foreach Implication rule in rules
        if not ImplicationSatisfied( rule, cps )
            return false
    return true

bool ImplicationSatisfied( Implication rule, Set<CP> cps )
    bool first := false
    bool second := false
    foreach CP cp in cps
        if cp = rule.cp1
            first := true
        if cp = rule.cp2
            second := true
    return (not first) or (second)
```

$(A \Rightarrow B) \iff ((A \notin \Omega) \lor (B \in \Omega)) \iff (\neg(A \in \Omega) \lor (B \in \Omega))$, in fact, is reflected in the last line of function `ImplicationSatisfied` for the implication rule passed in, $A \Rightarrow B$.

### 4.2 Phase 3: Model Construction

During the Model Construction Phase, the configurable reference model has to be developed in regards to the decisions made during the preceding phase Project Aim Definition. To store the context relevant meta information in the model, the respective parameters are attached to the model elements in form of terms and can later be evaluated to true or false. Only if the equation is evaluated to true or if there is no term attached to an element, the respective element remains in the configured model. To specify these terms, which can get complex if many characteristics are used, a term editor application has been developed, which enables the user to attach them to the relevant elements. Here, again, the ontology can support the developer by automatically testing for correctness and reasonableness of dependent parameters. Opposite to dependencies, exclusions take into account that under certain circumstances parameters may not be chosen together. This minimizes the risk of defective modeling and raises the consistency level of the configurable reference model. In the example given above, if the developer selects that SalesContactForm is VendingMachine, the parameter Beneficiary may not be InvestmentGoodsTrade, as investment goods can hardly be bought via a vending machine (see above). Thus, the occurrence of both statements concatenated with a logical AND is not allowed. The same fact has to be regarded when evaluating dependencies: If, like stated above, ContactOrientation = MailOrder determines the choice of PurchaseInitiationThrough = AND(Internet;Letter/Fax), the same statement may not occur with a preceded NOT. Again, the previously generated taxonomy can support the developer by structuring the included variants. The following pseudo-code describes the consistency check of a given configuration term:

```plaintext
bool ValidateTermNode( Node root, Set<Rule> rules )
    if root is AndNode
        if not ValidateTermAndNode( root, rules )
            return false
    if root is XorNode
        if not ValidateTermXorNode( root, rules )
            return false
    foreach Node node in root.ChildNodes
        if not ValidateTermNode( node, rules )
            return false
    return true

bool ValidateTermAndNode( AndNode root, Set<Rule> rules )
    Set<CP> cps := { }
    foreach CPNode node in root.ChildNodes
        cps := cps with node.cp
    if not ValidateCPSetAgainstExclusions( cps, rules )
        return false
    foreach Implication rule in rules
        if not ImplicationSatisfied( rule, cps )
            return false
    return true
```

$(A \Rightarrow B) \iff ((A \notin \Omega) \lor (B \in \Omega)) \iff (\neg(A \in \Omega) \lor (B \in \Omega))$, in fact, is reflected in the last line of function `ImplicationSatisfied` for the implication rule passed in, $A \Rightarrow B$.
The check for a term’s consistency is limited to a very simple functionality. This functionality, for example, disregards tests if a term or sub-term always solves to true or false, respectively. But it guarantees that no AND-operator combines configuration parameters excluding each other and no XOR-operator combines configuration parameters with one of them implicating another one. Furthermore, it ensures that no AND-operator combines configuration parameters where one of them implicates another parameter with a preceded NOT on its part. As the term’s structure is free from nested operators of same type, e.g. \( A \land (B \land C) = A \land B \land C \) or \(-(-A) = A\), these checks remain simple but very adequate and effective.

The terms are interpreted as trees with inner nodes representing logical operators (AND, OR, XOR, NOT) and leaf nodes holding the configuration parameters to combine in some way. The function ValidateTermNode expects a term’s root node and a set of rules—the ontology. It recursively iterates through all tree nodes calling functions ValidateTermAndNode and ValidateTermXorNode on inner nodes representing AND-operators and XOR-operators, respectively, to perform the consistency checks explained above. Function ValidateTermAndNode on its part uses function ValidateCPSetAgainstExclusions and furthermore function ExclusionSatisfied which are very similar to functions ValidateCPSetAgainstImplications and ImplicationSatisfied, respectively, stated earlier:

\[
(A \leftrightarrow B) \Leftrightarrow ((A \not\in \Omega) \lor (B \not\in \Omega)) \Leftrightarrow (\neg(A \in \Omega) \lor \neg(B \in \Omega)),
\]
in fact, is reflected in the last line of function ExclusionSatisfied for the exclusion rule passed in \( A \leftrightarrow B \).

4.3 Phase 7: Model Configuration

The Usage phase of a configurable reference model starts independently from its development. During the user’s Project Aim Definition Phase (phase 5) the potential user defines the parameters relevant to him to determine which reference model best meets his needs. He has got to search for it during the Search and Selection Phase (phase 6). Here, the advantage of the approach presented comes into play. Instead of having to customize the reference model manually, the user solely picks a suitable configurable reference model and uses the included ontology to pick the parameters relevant for his purpose.
By automatically including dependent parameters, the ontology can be of assistance in the same way as before, assuring that the mistakes made by the user by choosing the wrong parameters are reduced to a minimum. For each parameter—or set of parameters—a certain model variant is created. These variants have to be differentiated by the aim of the configuration. On the one hand, the user might want to configure a model that cannot be further adapted. This happens if a maximum of one parameter per characteristic is chosen. In this case, the ontology has to consider dependencies as well as exclusions. On the other hand, if the user decides to configure towards a model variant that should be further configured, exclusions may not be considered and have to be integrated. Both possibilities have to be covered by the ontology. Furthermore, a validation should cross-check against the ontology that no terms exist that always equate to false. If an element is removed in every configuration scenario, it should not have been integrated into the reference model in the first place. Thus, the taxonomy can assist the user during the Configuration Phase by offering a set of parameters to choose from. Combined with an underlying ontology, the possibility of making mistakes by using the taxonomy during the model adaptation is reduced to a minimum. This can be done by passing the selected configuration parameters and the ontology’s rules in ValidateCPSetAgainstExclusions and ValidateCPSetAgainstImplication from above, looking forward to results being true.

4.4 Summary

The meaningfulness of the application of ontologies during the phases of the lifecycle of configurative reference models has been shown during the preceding section. However, the way how the terms of an ontology are being evaluated changes during the phases. Moreover, not all rules have to be taken into consideration, as the relationship type Implication is not relevant for configuration but only for the structuring of the given characteristics. The following table provides an outline:

<table>
<thead>
<tr>
<th>Phases</th>
<th>Relationship types</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Implication</td>
<td>During the first phase, exclusions don’t have to be taken into account, as a configurable reference model has to contain information for all inherent variants that possibly can be derived by configuration.</td>
</tr>
<tr>
<td>Project Aim Definition</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exclusion</td>
<td></td>
</tr>
<tr>
<td>Model Construction</td>
<td>X</td>
<td>During the third phase, every rule has to be checked whether it is valid or not and delivers a valid model. As such, exclusions have to be considered as well.</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Model Configuration</td>
<td>X</td>
<td>During this phase, the model gets configured. Which relationship types have to be regarded depends on the user’s purpose for configuring the model. If he wants to create a variant that no longer should serve as a configurable reference model, exclusions may not remain in the variant derived. Otherwise they may stay in the variant for further configuration.</td>
</tr>
<tr>
<td></td>
<td>(X)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Relevance of the relationship types during the respective phases

5 CONTRIBUTION TO SUPPORT REFERENCE MODELING TOOLS

The H2-Toolset is a meta modeling tool for the construction of hierarchical models and the underlying modeling methods enabling, disabling and adapting certain modeling techniques. Moreover it provides a configurative reference modeling feature and therefore realizes the basic ideas of ontology functionality analyzed in this paper.
Figure 3: Preselection of relevant Configuration Parameters

During the phase Project Aim Definition usually certain configuration parameters specifying the underlying domain are chosen with respect to satisfy a given ontology. This can be implemented as shown in Figure 3: The user checks the desired configuration parameters to mark them as selected for the purpose present.

Figure 4: Just-in-time-validation of terms during their construction

For a selected configuration parameter the dependencies are instantly displayed. In a following validation step the set of selected configuration parameters will be checked against the underlying ontology. In the Model Construction Phase both exclusion and implication rules come into effect. Certain configuration parameters are put together and combined in logical terms. The terms can be built either by using a comfortable interface to construct a syntax tree and selecting the parameters from a list or by typing in the term’s formal expression (Figure 4, background). The combinations of referenced configuration parameters are then validated against the ontology, and warning messages are generated ac-
cording to rule violations (Figure 4, foreground). During the Usage of a reference model, its adapta-

tion resulting in a new created (reference) model, the tool support may look quite similar to the selec-
tion of configuration parameters during Project Aim Definition, particularly with regard to validating
and responding to the user’s input: In a first step the user selects some parameters representing the
modeling purpose from the set of pre-selected configuration parameters.

In a second step the configuration parameters describing the precise circumstance are then validated
against the ontology’s implication rules. Figure 5 shows a case were some rule violations were de-
tected validating the configuration parameters: SelfService and SalesPerson have been selected, Visit-
ToStore not.

Figure 5: Incorrect choice of parameters

6 OUTLOOK

The approach presented in this paper simplifies the consistent structuring, attachment and usage of
parameters during the lifecycle of configurative reference models. In comparison to existing ap-
proaches of configurative reference modeling, not only the configuration itself can be automated, but
also the verification of the consistency of the set of parameters chosen by the user. However, only
three of the eight relevant phases of the configurative reference model lifecycle have been taken into
consideration. If and how the usage of the ontology effects and hopefully improves the execution of
the other phases is going to be the object of future research. Considering the ontology, only a very li-
mited set of relationship types has been used. It has to be analyzed whether extending the ontology
with additional relationship types and the supply of attributes like cardinalities is of any benefit. By
doing so, the necessity to—for instance—selecting at least two but no more than four parameters can
be modeled in a clearer and more intuitive way. Also that concerns the implication of at least one of
the three parameters Letter, Fax and Internet by Mail order—for logical reasons Letter and Fax may
still be bundled. Considering some business may operate both as a retailer and as a wholesaler, but
never with respect to the same customer order, the ontology needs to support a hierarchy of concepts
and relationships to allow rules being attached to entities of different conceptual levels. Furthermore,
the approach presented in this paper has been tested with eEPC- and Entity Relationship-Models by
extending the meta model of the respective modeling languages with the possibility to annotate addi-
tional information. Nevertheless, the adaptability to other modeling techniques, too, shall be proven.
Finally, the approach has not been tested yet in a day-to-day business environment. It would be ben-
eficial to prove its applicability with support of practitioners and by integrating it into other modeling
tools.
7 ACKNOWLEDGEMENTS

This paper was written within the context of the research projects FlexNet and ServPay, funded by the Federal Ministry of Education and Research (BMBF), promotion signs 01FD0629 and 02PG1010.

References