

# **Towards Tool Support for Configurative Reference Modeling - Experiences from a Meta Modeling Teaching Case**

Patrick Delfmann, Christian Janiesch, Ralf Knackstedt, Tobias Rieke, Stefan Seidel

European Research Center for Information Systems (ERCIS)  
University of Münster  
Leonardo-Campus 3  
48149 Münster, Germany  
{delfmann | janiesch | knackstedt | rieke | seidel}@ercis.de

**Abstract:** The adaptation of conceptual information models to specific requirements has been discussed for several years. Especially, different approaches to information model configuration have reached certain popularity. Nevertheless, up to now, model configuration is not supported sufficiently by recent modeling tools. In this paper, we present the implementation of a meta model based model configuration approach in order to close this gap. The results of the implementation are two modeling tool add-ons that enable model configuration based on the modeling tools ARIS and H2. The implementation was conducted in the course of a teaching seminar at the University of Münster. Besides the configuration approach as well as architectures and functionalities of the developed tools, we discuss teaching experiences.

## **1 Introduction**

Reference models are conceptual models that are constructed with the intent to be reused for the design of specific models [Sc98, Ro03]. Using reference models as templates for deriving application context-specific models leads to time and cost reduction [Sc98]. However, in order to realize these time and cost effects, reference models have to fit to the characteristics and requirements of the company respectively the user group. Otherwise, the reference model would be either far too general to offer a real utilization possibility or it is not applicable to the company characteristics and requirements due to an overwhelming amount of non-relevant details. In order to attain a broader customer clientele while keeping the economic utilization, model configuration has been discussed in different approaches [Be02, SDG03, RA05]. Model configuration allows for deriving specific views from the reference model. Such views can be determined by *characteristics* of the company [Wi03] (e.g. the business type, number of employees, or business transactions) and/or user specific *perspectives* [DS96, RG00] (e.g. business reengineering, risk management, or software engineering).

Both company characteristics and perspectives represent parameters that can determine a certain model configuration. Therefore, we subsume company characteristics and perspectives in so-called *configuration parameters*. Considering configuration parameters that furthermore allow combinations of perspectives and company characteristics, *configuration parameter structures* are necessary. Cf. figure 1 for a conceptual specification as Entity-Relationship Model (ERM [Ch76]).

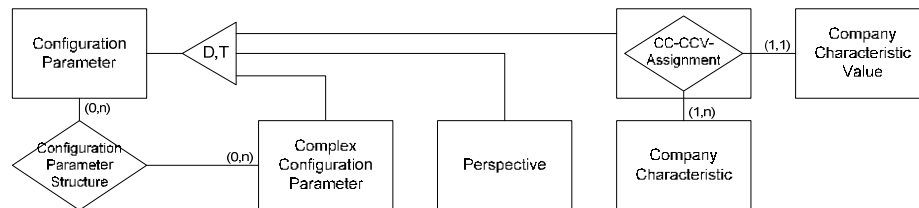


Figure 1: Configuration Parameter Structure

An integrated approach for configurative reference modeling has not been supported by modeling tools, yet.<sup>1</sup> Hence, the research contribution of this paper is on the one hand to present the feasibility of the – so far primarily theoretic – configuration concepts, developed by BECKER ET AL. [Be02, Kn06, De06]. The main challenge is to transform their meta model based specification of configurative reference modeling languages into applicable tool implementations.

On the other hand, the research contribution of this paper is to present experiences from a meta modeling teaching case. The configuration mechanisms that are able to modify information models are defined with meta models, i.e. their specification is integrated into the modeling language specifications. In order to be able to configure meta models as well – this equals a configuration of the modeling languages themselves – some configuration mechanisms had to be specified on a more abstract specification layer. This has been realized with the help of a meta meta model. These theoretic configurative reference modeling concepts based on meta modeling have been taught in lectures for three years and were implemented within a student project seminar.

The paper is structured as follows: In section 2 we present basics of configurative reference modeling with a brief reference to related work. Section 2 also introduces the teaching scenario. In section 3 we present the specification and implementation of the two modeling tools. Moreover, we present the findings of our teaching case concerning issues and decisions that led to the final result. A conclusion and outlook is given in section 4.

<sup>1</sup> In an empirical study that was performed in the course of a university seminar as well, several modeling tools and meta modeling tools have been evaluated concerning their support for defining and maintaining variants, especially by model configuration. No tool could be identified to support an integrated configuration approach. The findings of this study are already submitted for publication and are currently being reviewed.

## 2 Foundations

### 2.1 Conceptual Framework of Configurative Reference Modeling

Our tool implementations are based on the configurative reference modeling approach of BECKER ET AL. [Be02] (alternative approaches have been developed, e.g. by ROSEMAN and VAN DER AALST as well as SOFFER, GOLANY, and DORI, cf. [SDG03, RA05]). We chose this approach first, due to its greatest comprehension and second, due to the fact that it was developed “in-house” which allowed an in-depth pre-understanding of the approach.

Within this approach, a configurative reference model is defined as an integrated total model that contains specific information for all perspectives and company characteristics that shall be supported. In this way, redundancies are avoided that could otherwise arise if each perspective or business characteristic would be provided with an own model. The specific models are provided by creating views on the integrated total model. The step of creating views on the integrated model is called *configuration* and is performed by hiding all information which is not relevant for the specific perspective or company characteristic. Hiding non-relevant information is provided by so-called *model projections* that reduce the total model to relevant model elements.

Conceptually, model projections are performed by modifying the modeling languages and models dynamically. For this purpose, we use a framework that consists of different modeling layers: the *model layer*, the *meta model layer* and the *meta meta model layer*:

- On *model layer*, the reference models that shall be configured are situated. Depending on which configuration parameter instance is valid at the moment, they contain different information.
- On *meta model layer* [St96], the modeling languages, which are used for configurative reference modeling (here: Event-Driven Process Chains (EPCs) [Sc00], ERM, Organigrams, Function Trees, MetaMIS [Ho01] etc.), are specified. Just like the models on model layer, they contain different information depending on which configuration parameter instance is valid at the moment. This means that the modeling languages can have a different expressive power depending on the configuration parameter.
- On *meta meta model layer* the modeling language of the meta models is specified. I.e., it is specified that there exist model element types on meta model layer that are interrelated. Furthermore, the meta meta model layer serves as configuration environment. I.e., the meta meta model layer is used to specify whether *model element types* or *particular model elements* are relevant for different configuration parameter instances. This is why configuration parameters are specified on meta meta model layer as well.

In the following, we show how configuration specifications on meta meta model layer can be used to perform model projections that affect *model element types* on the one hand and *particular model elements* on the other hand.

Model projections can be distinguished in two different types: On the one hand, it can be necessary to hide all model elements that belong to a particular model element type (e.g. *organizational units* that represent execution responsibility of activities in process models). Here, it is appropriate to modify the modeling language by removing the specification of the according model element type, e.g. *organizational unit*. This happens on meta model layer. Hence, this model projection type is called *meta model projection*. Meta model projections are dependent on configuration parameters, i.e. the meta model looks different dependent on the configuration parameter instance that is valid at the moment. Hence, a specification area is needed in which configuration parameters can be assigned to model element types. This is done on meta meta model layer, whereas instances of meta meta model elements appear as meta model elements.

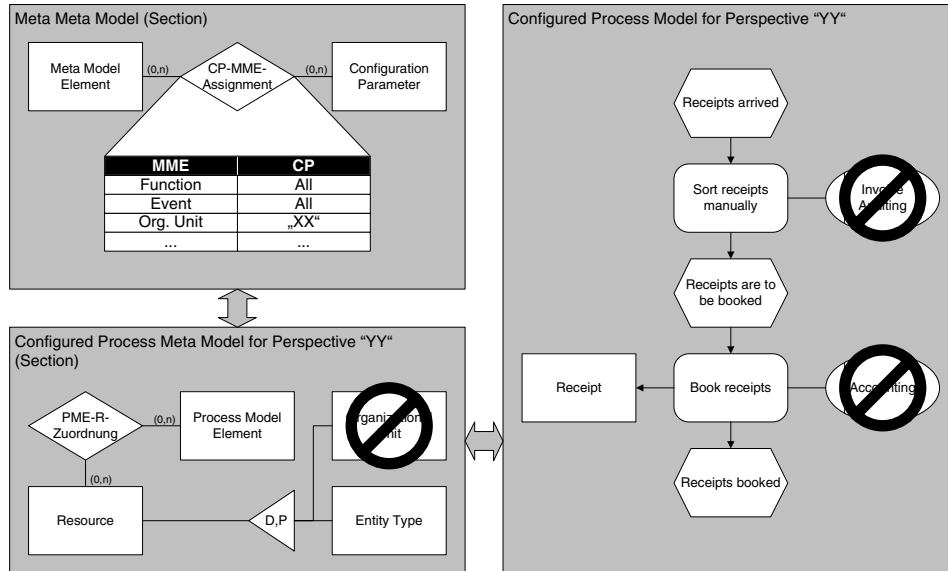


Figure 2: Specification of Meta Model Projection on Meta Meta Model Layer and its Influences on Lower Model Layers [Be02]

As specification language, we use ERMs on meta meta model layer as well as on meta model layer. Hence, the meta meta model layer has to contain the specification on the meta modeling language – the ERM language itself. In our examples, we use a simplified version of the meta meta model, in which all element types of the ERM are subsumed within the Entity Type *Meta Model Element*. Instances of the entity type *Meta Model Element* can be selected by assigning them to the *Configuration Parameter*. A configuration is performed by selecting a configuration parameter. Each element type that is not assigned to this configuration parameter (e.g. *organizational unit*) is hidden on meta model layer. As a consequence, every model element on model layer that belongs to this element type, is hidden as well (cf. figure 2; organizational units that can be anno-

tated to functions in EPCs are hidden. This is achieved by hiding their definition on meta model layer. This, in turn, is provided by choosing a perspective “YY” which is not assigned to the organizational unit on meta-meta model layer).

On the other hand, *model projections* allow for hiding distinct model elements on model layer (e.g. distinct, non-relevant process branches in EPCs; cf. figure 3).

For model projections, a simple assignment of configuration parameters to model element types is not suitable, since this always affects all model elements of an element type. In order to perform a configuration of distinct model elements (e.g., a single activity within a process model), a mechanism is needed that is able to act on instances of model element types. Since a configuration means hiding of model elements, it is appropriate to introduce a mechanism that restricts the set of instances of model element types to those that are regarded as relevant for a specific configuration parameter instance. Consequently, *Constraints* are introduced on meta meta model layer, which are assigned to *Configuration Parameters* and *Meta Model Elements*. Depending on a specific perspective or company characteristic, a constraint can be assigned to an element type on meta model layer that restricts the set of element type instances to be displayed. Each instance that does not belong to the selected set is hidden on model layer (e.g. those elements that belong to a process branch, which is regarded as non-relevant for the current perspective). The constraints operate depending on *attributes* that characterize a model element as belonging to a specific configuration parameter.

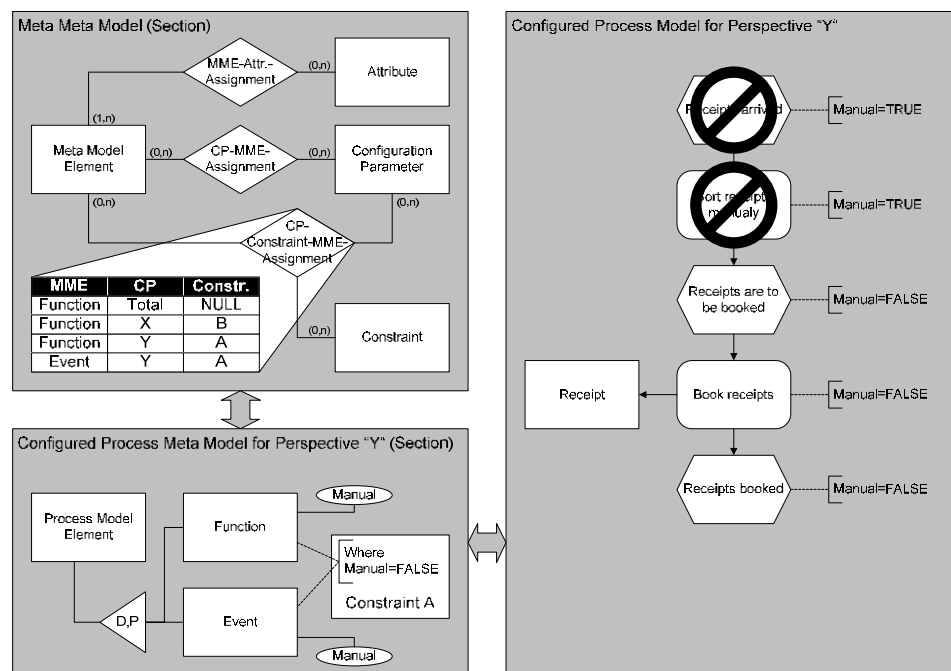


Figure 3: Specification of Model Projection on Meta Meta Model Layer and its Influences on Lower Model Layers [Be02]

Cf. exemplarily figure 3: activities within a process branch, which are performed manually, are hidden. This is achieved by hiding their instances through the application of a constraint on meta model layer on *Functions* and *Events*. In this example, not only functions but also events are marked as manual or non-manual in order to show that they belong to a process branch that is performed manually or not. This restricts the set of elements to be shown to those that belong to a process branch performed not exclusively manually. This, in turn, is provided by choosing a perspective “Y” on meta-meta model layer that results in the annotation of the constraint “A” to the element types *Function* and *Event*.

Note that each model projection is preceded by a meta model projection, i.e. the meta model projection that controls the assignment of constraints to meta model elements.

Within the configurative reference modeling framework of BECKER ET AL., the following configuration mechanisms are distinguished which each are based upon meta model projection or model projection:

- *Model Type Selection* allows for providing only those modeling languages and their according model types to users that are relevant for them. E.g., employees who use process models as guidelines for their everyday work, do not need to be provided with data models describing database structures. Model Type Selection is based upon meta model projection.
- *Element Type Selection* considers the necessity to provide modeling language variants with different expressive power for different user groups. E.g., practitioners prefer process models that are easy-to-read. This can be achieved by e.g. fading out resource types that are annotated to process functions. As well as Model Type Selection, Element Type Selection is based upon meta model projection (cf. figure 2).
- *Element Selection* operates on particular parts of a model. *Term-based Element Selection* and *Attribute-based Element Selection* are distinguished. Term-based Element Selection assigns logical terms to model elements that connect them directly to configuration parameters, whereupon Attribute-based Element Selection assigns characteristics to model elements that mark them as relevant or non-relevant for different configuration parameters. Element Selection is based upon model projection (cf. figure 3).
- In different parts of a company different naming conventions may have been established (synonyms); i.e. different names have the same meaning (e.g., procurement employees call a supplier invoice just “invoice“, whereas distribution employees use the naming “supplier invoice“). In order to consider these conventions in line with information modeling, the configuration mechanism of *Synonym Management* allows for perspective-specific exchanging of model element namings. Synonym Management is based upon meta model projection since a name exchange is always performed for each element of an according type.
- Through *Representation Variation*, the representational aspect of modeling languages can be changed depending on the current perspective. I.e., symbols of model elements can be exchanged. Representation Variation is realized by meta model projection.

In order to build a configurative modeling tool, our goal was to implement these configuration mechanisms within a modeling environment. The implementation activities were performed by students in the course of a seminar at the University of Münster.

## 2.2 Teaching Scenario

For the course of this teaching case's seminar, called "Configurative Information Modeling (COIN)", the assignment was to implement the previously introduced mechanisms for model configuration. To border the seminar's topic certain constraints were imposed on the students.

In order to provide a mature modeling environment and to concentrate only on configuration issues, we decided not to develop an own modeling tool but to use existing modeling tools and to build configurative add-ons. First, two software tools were selected to provide the basis for the implementation. Second, the programming language was decided upon. And third, the range of configuration mechanisms was specified.

The decision was made to implement the configuration mechanisms as introduced above for two tools with distinct characteristics. Since configuration mechanisms already have been introduced for operational [De06] and planning purposes [Kn06], it was decided that both should be covered in the seminar. To cover most of the other characteristics, tools for each purpose were selected that had only few commonalities concerning the morphological box shown in table 1.

purpose	operational		planning	
supported levels of modeling	modeling		meta modeling	
modeling languages available	EPC	ERM	MetaMIS	Organigram
code access	source code available	API	only import and export	
maturity of product	mature software	prototypical software	conceptual specification	

Legend:



 Tool 1: ARIS
  Tool 2: H2

Table 1: Morphological Box for Tool Selection

For the modeling of information systems for operational purposes the *ARIS Business Architect 7.0* of the *IDS Scheer AG* was selected. The former *ARIS Toolset* was ranked as one of the pioneers in process modeling [Ga05] and is mature software. ARIS is not a meta modeling tool and changes to the languages available are not intended. However, the ARIS framework provides languages for all major areas of information systems design, in particular EPCs, ERMs, and Organigrams. Prototypical tests with prior versions concerning the Application Programming Interface (API) were promising. However, for the purpose of comprehensive model configuration, the current version did only allow import and export of models and no configuration via an API. Despite these restrictions

(and since some of this was only found out in the course of the seminar), ARIS was chosen as basic modeling tool in order to be able to provide the implemented configuration concepts to a preferably wide user group. The option to use a meta modeling tool – such as *MetaEdit+*, *Metis*, *Cubetto*, or *ConceptBase* – was either discarded due to not providing an own comprehensive language, due to a lack of usability, and due to the fact that these tools are neither well-known nor wide-spread so that a later broad usage could not be expected. Despite the fact that the meta modeling environments of these tools were promising concerning the specification of configuration mechanisms, the effort that had to be spent on the implementation of user-friendly configuration mechanisms turned out not to be significantly less than using a proprietary modeling tool without any meta modeling environment.<sup>2</sup>

For the modeling of information systems for planning purposes the *H2 Toolset* was selected. H2 is a meta modeling toolset developed by the *European Research Center for Information Systems (ERCIS)* and maintained by the *Prof. Becker GmbH* [Be05]. Although being a meta modeling tool, the original purpose of the toolset was to provide modeling support for management information systems. Hence, most of the modeling languages designed for H2 deal with information systems for planning purposes, such as *MetaMIS*. MetaMIS is a modeling language for specifying management views on information systems [Ho03]. It was used as a reference language to test the implemented configuration mechanisms. H2 is a specialized meta modeling software that focuses on modeling hierarchical structures and utilizes an own meta modeling language. The source code of the software, which at that time was in a stable but somewhat prototypical state, was freely available to the students.

### 3 Tool Prototypes for Configurative Reference Modeling

#### 3.1 adapt(x)

##### *Conceptual Specification*

Since the conceptual specification of configurative reference modeling as proposed by BECKER ET AL. [Be02] was already designed according to the ARIS concept and tool, further conceptual specifications were not necessary (for basic conceptual specifications, cf. Section 2.1).

##### *adapt(x) Architecture*

In ARIS, no meta modeling environment is available. Consequently, each configuration specification had to be programmed by hand. Furthermore, model manipulations operating on the database ARIS uses to store its models was not possible, since the database structure of ARIS is not provided by IDS Scheer in an open document. An own reconstruction of the data model of the data base was impossible due to the cryptic denotation

---

<sup>2</sup> These facts were found within the empirical study already mentioned in footnote 1.



of its tables. Since ARIS 7.0 provides a script language (ARIS Script) that is able to modify models, this script language was selected as configuration environment in a first step. Due to a lack of functionality and speed, this option was discarded after a few tests. Finally, the decision was made to use the XML import and export interface of ARIS in order to perform model configurations in an external configuration add-on called *adapt(x)*. Nevertheless, ARIS Script had to be used to be able to add configuration rules such as configuration attribute instances (cf. Section 2.1) to ARIS models within its modeling environment. This was necessary because model access through an application programming interface (API) is not supported sufficiently by ARIS. Since both the models administrated by ARIS and the configuration mechanisms of *adapt(x)* make use of configuration parameters, it was decided to store them in a data base accessible by ARIS Script and *adapt(x)* concurrently. Furthermore, ARIS Script was used to start *adapt(x)* out of ARIS via a macro. Thus, the basic architecture of *adapt(x)* was designed as presented in figure 4:

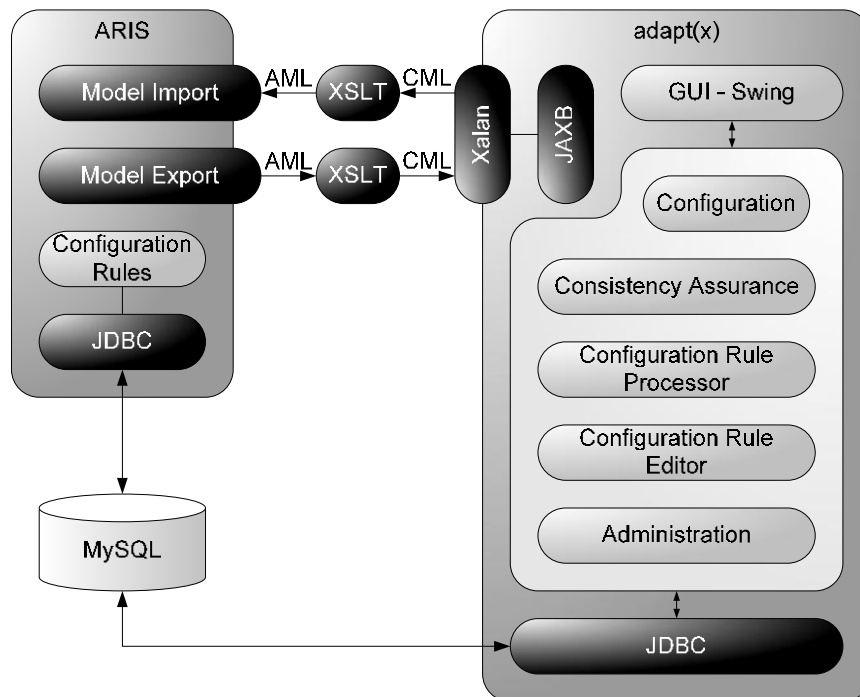


Figure 4: *adapt(x)* Architecture in Detail

Within the modeling environment – in this case ARIS – possibilities have to be established to make information models configurable; i.e. configuration terms as well as configuration attributes have to be specifiable. Here, ARIS Script is used in order to assign configuration terms or configuration attributes to models or model elements in an easy-to-use configuration rule specification environment. In order to guarantee a consistent use of configuration parameters resp. valid configuration attribute instances, these are

balanced with those used by the adapt(x) configuration environment via a MySQL database that serves as mediator. ARIS Script accesses the database via a JDBC interface.

In order to configure a model that is enriched with configuration rules, it has to be exported as XML file. Here, the built-in XML export interface of ARIS is used. The model export is easily started through the configuration rule specification environment based on ARIS Script. The XML files are provided in a proprietary XML format of IDS Scheer AG, the so-called ARIS Markup Language (AML). Due to the aim of making the adapt(x) configuration environment reusable for further different modeling tools, the decision was made to develop a generic XML format that is suitable for storing configurative model information. As a result, AML export files are transformed into so called COIN-ML (CML) files via an XSLT transformer. COIN-ML is generic in the sense that it abstracts from the modeling language the stored models are based on. The relevant information for configurative models is restricted to model elements, model element relations, models, and configuration rules stored in attributes. Syntax information of the used modeling languages have to be provided by the tools that use the CML files (here: adapt(x)). Proprietary information is stored in an extra section of the CML file that can be restored once the CML file is transformed back into a proprietary format such as AML.

The adapt(x) configuration environment consists of a three-layer-architecture that comprises the components *presentation*, *configuration logic*, and *database access*. The configuration logic layer uses Xalan as XSLT transformer that is able to translate different XML formats into CML (in our prototype, only AML transformation is implemented). Since the configuration logic layer uses Java, we make use of JAXB, a Java-XML-object mapper that creates objects out of the CML data that can be manipulated in Java. This way, the configuration environment accesses the model data. In order to perform model configurations, different components are necessary within the configuration logic layer:

- The *administration component* comprises user administration as well as model administration. Within *user administration*, user authorizations can be specified. User logins and passwords of adapt(x) that correspond with those of ARIS make an extra login dispensable when adapt(x) is started out of ARIS, since login and password are adopted automatically. *Model administration* allows for loading model data through import and saving model data via export. Model imports are executed either manually or automatically when adapt(x) is started out of ARIS. In turn, model exports trigger an automatic import in ARIS.
- Within the *configuration rule editor*, the user is able to define configuration adjustments that can comprise sets of configuration mechanism instances. E.g., the configuration rule *No Data* comprises the instances *No Data Model* of *Model Type Selection* and *No Entity Types in Process Models* of *Element Type Selection*. Another example is the configuration rule *No Manual Activities* that comprises the configuration mechanism instance *Hide Functions and Events in Process Models that are Assigned with the Attribute Instance Manual=TRUE* of *Attribute-based Element Selection*.

- The *configuration rule processor* assigns configuration rules to configuration parameters. E.g. the perspective *Organizational Design* is assigned to the configuration rule *No Data*. The result of a configuration according to *Organizational Design* will be the elimination of all data models as well as of all entity types in process models in a model system to be configured.
- With the *configuration component*, configuration parameter sets can be defined on whose basis the configuration shall be performed, e.g. *Perspective: Organizational Design AND Business Type: Warehousing*. Furthermore, the configuration process itself is performed by the configuration component. The result of a configuration is stored in a CML file via JAXB, transformed to AML and re-imported in ARIS.
- In the course of configuration, inconsistencies can occur in information models that consist e.g. of gaps that are results of model element hidings. The *consistency component* provides algorithms that are able to restore the configured models in order to guarantee their syntactical correctness [De06, pp. 129-147].

#### Exemplary Configuration Process

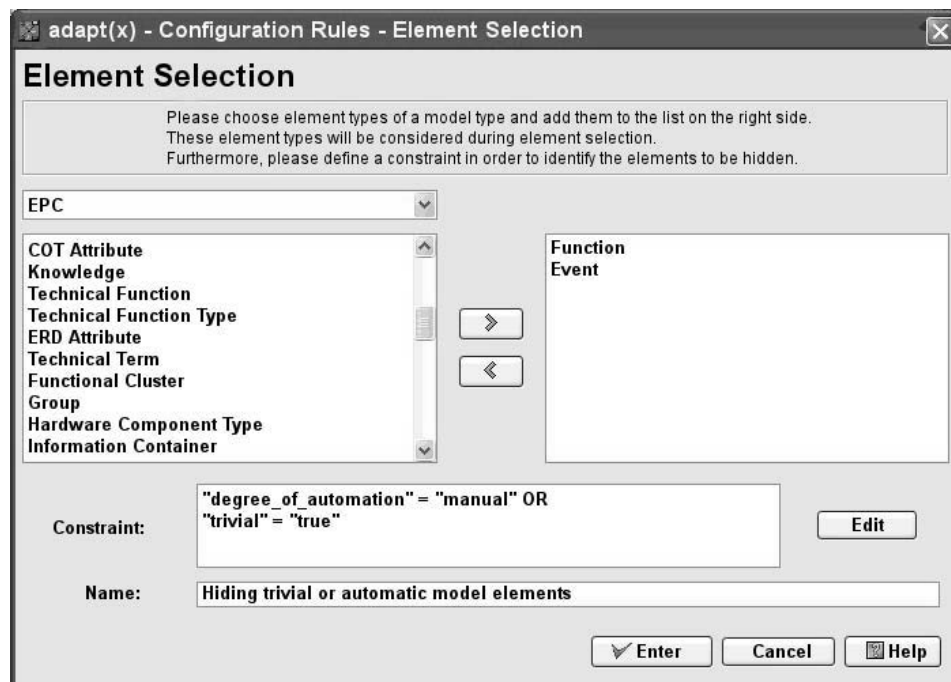


Figure 5: Specifying Cnfiguration Rules in adapt(x)

In order to perform a configuration, *first*, within adapt(x), company characteristics and perspectives to be supported have to be specified with the configuration component. *Second*, within the configuration rule editor, configuration rules are specified that base on configuration mechanisms (cf. figure 5). *Third*, configuration rules are assigned to

perspectives and company characteristics with the configuration rule processor (cf. figure 6).

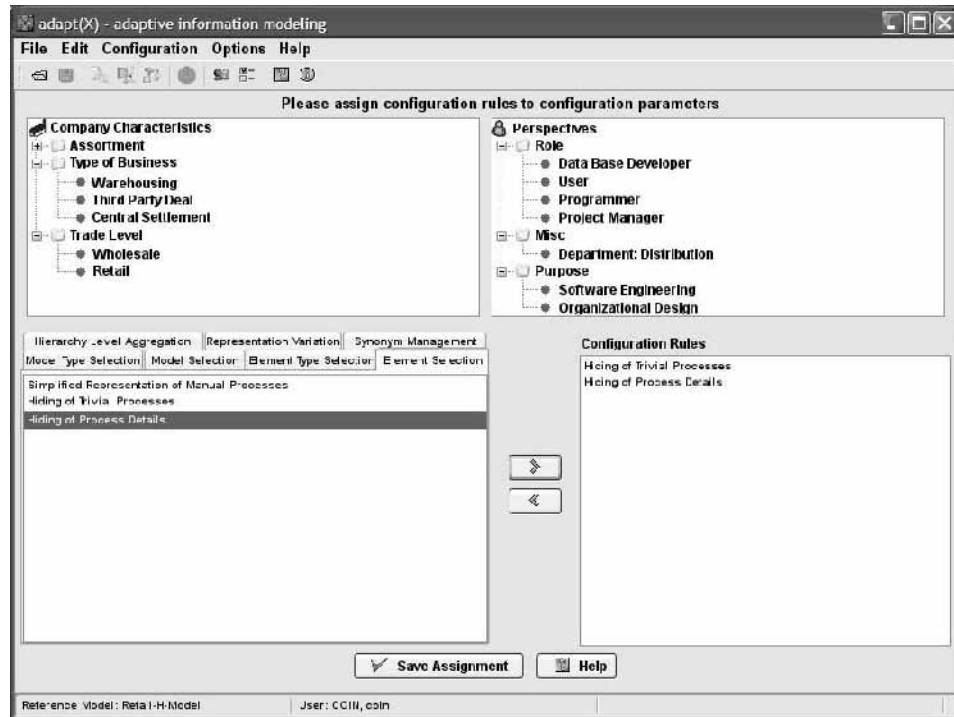


Figure 6: Assigning Configuration Rules to Configuration Parameters in adapt(x)

*Fourth*, in ARIS, the reference model is constructed. Configuration attributes or terms are assigned to model elements that shall be configured. Since configuration parameters and valid attribute instances have been specified in the course of configuration rule specification in adapt(x), these are now available consistently in ARIS via the MySQL database interface (cf. figure 7).

*Fifth*, the models to be configured are exported via XML. *Sixth*, within adapt(x), which starts automatically, the perspectives and company characteristics for which the model shall be configured are chosen in the configuration component. *Seventh*, the configuration itself is performed. *Eighth*, the configured model data are exported and imported in ARIS. The configuration is completed.

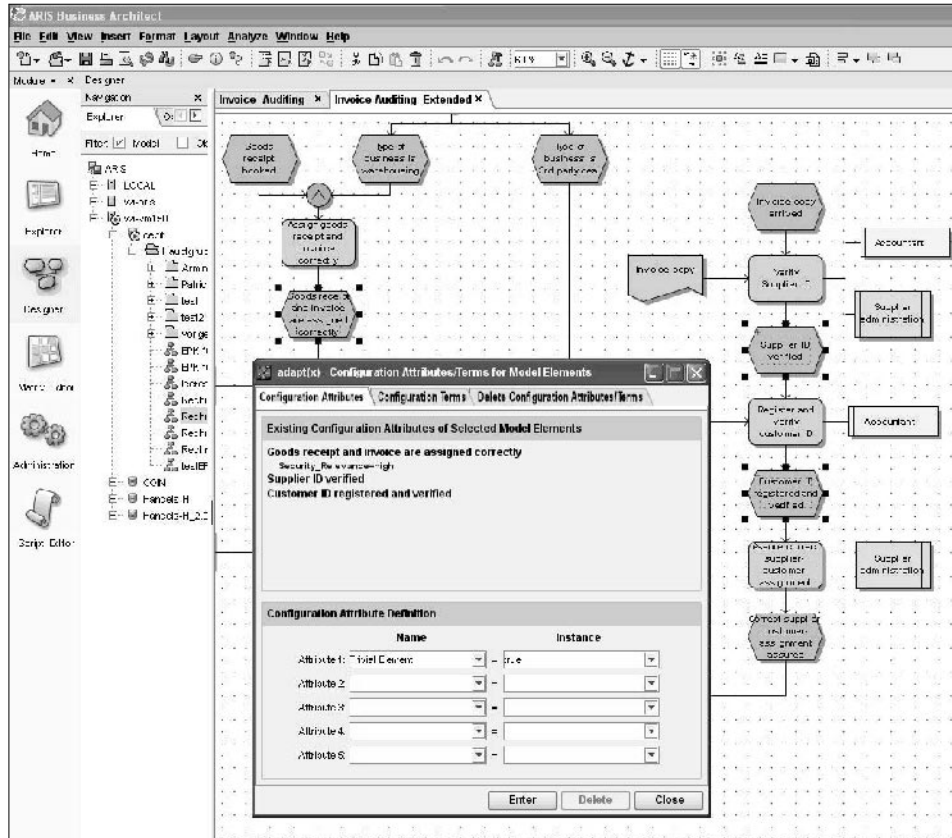


Figure 7: Assigning Configuration Rules to Model Elements in ARIS

### 3.2 H2configurative (H2c)

#### Previous Work on H2

During a research project, an approach has been developed to conceptually specify information needs [Be03]. It comprises a conceptual language that facilitates the communication between management and IT analysts. Its main concept is the use of hierarchies. Hierarchies represent an inherent concept of abstraction and have been found to be intuitively understandable for users. A hierarchy is understood as a transitive, irreflexive, and asymmetric relationship of entities. It is represented as a connected directed acyclic graph with a designated initial (root-)node forming a tree structure. Existing general purpose meta case tools such as *MetaEdit+*, *Metis*, *Cubetto*, or *ConceptBase* were not found to be suitable since the modeling of extensive hierarchies requires built-in model navigation and management features to efficiently access the models.

Therefore, a repository was designed to support the (meta) modeling of hierarchical structures. The repository contains both language specifications and the actual models to

allow a careful adaptation of the meta models even while modeling. Consequently, the repository consists of two parts. The first part represents a container to create the language specifications (i.e. meta models) and, therefore, serves as a meta meta model related to the real world [St96]. The second part represents the actual models that are instances of a certain language which, in turn, are models related to the real world. An excerpt of the repository can be conferred in figure 8.

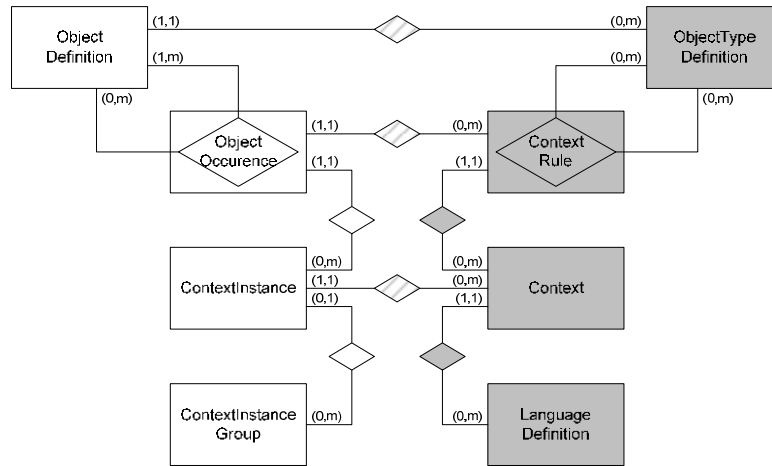


Figure 8: Excerpt from the Repository's Schema

A language consists of several elements i.e. *ObjectTypeDefinitions*. These *ObjectTypeDefinitions* represent real world constructs. Between *ObjectTypeDefinitions* respectively their instances there are relationships. These are represented by *ContextRules*. Several *ContextRules* set up a so called *Context*. A *LanguageDefinition* consists of several *Contexts*. Therefore, this part of the repository's schema can be seen as a language based meta meta model that describes the conceptual part of meta modeling languages used to define models of actual modeling languages. These models of modeling languages are meta models related to the real world.

Based on a *LanguageDefinition* *ObjectTypeDefinitions* are instantiated and *ObjectDefinitions* are created. Relationships between two Objects are represented by *ObjectOccurrences*. Any *ObjectOccurrence* is based on a certain *ContextRule* and is part of a *ContextInstance*. Hence, the same *ObjectTypeDefinition* can have several relationships. Consequently, on instance level *ObjectDefinitions* can be part of several *ContextInstances*. Thus, different views on the same element can be expressed. As a constraint, an *ObjectDefinition* can only occur in a certain *ContextInstance* once. It is possible to define rules concerning a *Context's* structure. These rules can either be related to the *Context* in general or to a *ContextInstanceGroup* that is a set of *ContextInstances*. *ContextInstanceGroups* have to be defined during the modeling of the instances. Properties for the groups can be defined.

H2 is an implementation of this repository. It allows the definition of various modeling languages as well as the actual modeling and management of models. The toolset con-

sists of a meta modeling editor, the so called *language editor* to define modeling languages, an *attribute editor* to define attributes and attribute groups independently of the modeling constructs used, a so called *hierarchy level editor* to define hierarchy levels that can be used to structure the trees created, and a *model editor* to create the actual models in a particular language.

### Software Architecture

The software architecture of H2 and consequently of H2c features several specific components. Figure 9 gives an overview. Some components had to be adapted in order to comply with the requirements imposed by the conceptual specification. These components are marked in bright gray while new components are marked in dark gray. In the figure, layer spanning support components have been arranged vertically while the other components are ordered in their respective layers horizontally. Apart from the H2c specific layers, a couple of external components exist: in particular the .Net Framework 2.0 and any database management system (DBMS).

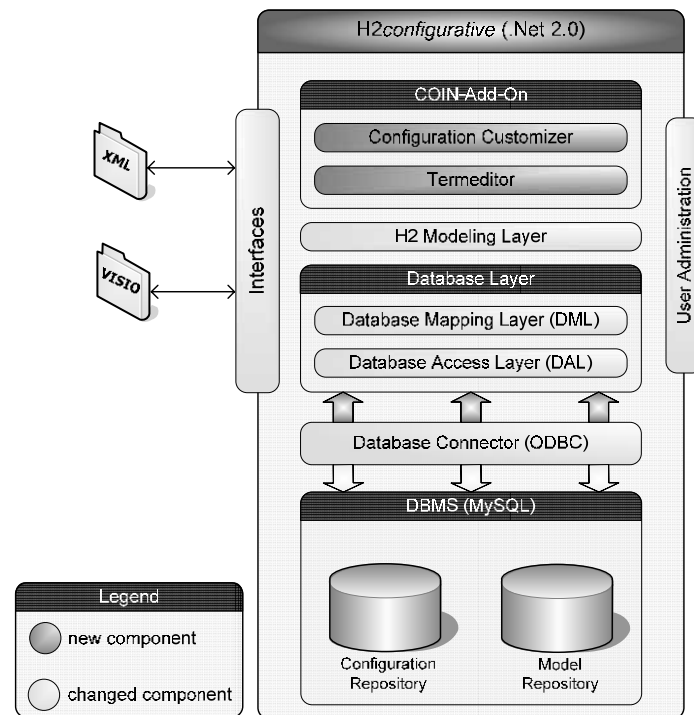


Figure 9: Architecture of H2c

On top of the chosen DBMS, a database connector layer enables the communication of all subsequent layers with the databases via ODBC. The database layer includes methods to access the databases, encapsulate their content in object-oriented structures, and map them to internal objects. The XML interface was revamped and user administration was updated to comply with the additional requirements. The H2 modeling layer comprises

the original business logic to create modeling languages and models and is extended with add-ons that make configurative reference modeling possible: In particular, these are a *term editor* and a *configuration customizer* that enable the assignment of configuration terms to model or meta model objects. Both comprise the core functionality implemented by the project seminar. They are covered in more detail in the following.

### Conceptual Specification

To enable the toolset to comply with the requirements of configurative reference modeling, modeling languages and their models have to be assigned to a higher instance than *LanguageDefinition*. Therefore, the entity type *ReferenceModel* is introduced. It encapsulates *LanguageDefinitions* and serves as the root node in the database explorer of the model editor. All *ConfigurationParameterDefinitions*, *Rules*, and *Terms* are assigned to one and only one *ReferenceModel*. Languages, rules and terms can be exported and reused in any other reference model but they are unique within each of them. Cf. figure 10 for an ERM of the core extensions in relation to the former root node *LanguageDefinition*.

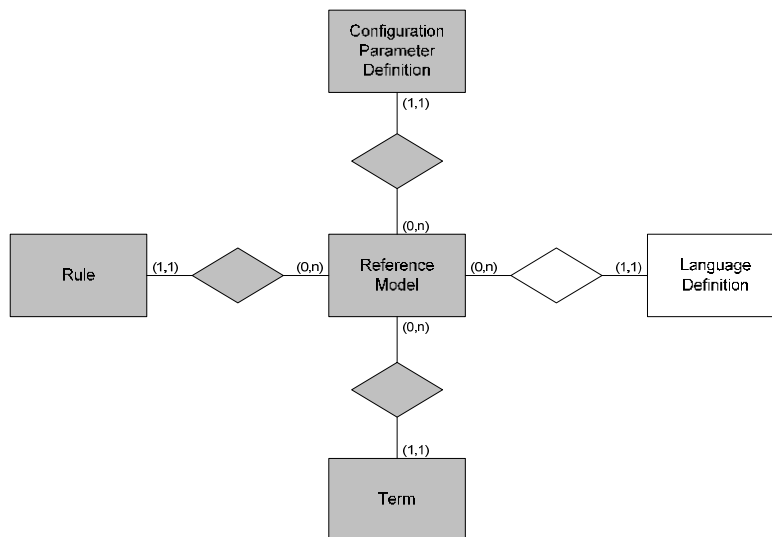


Figure 10: Components of a Reference Model

The configuration process is as follows: In the course of modeling, models have to be annotated with information that allows for an automated configuration of models according to the specific requirements of an enterprise. The annotated information are configuration parameters and belong to a configuration parameter definition. Not all general configuration parameters may be necessary for the adaptation of the reference model, thus a pre-selection of configuration parameters can take place to shorten and simplify the process. The annotation itself is conducted with terms that declare the applicability of model elements according to certain configuration parameters.



Configuration parameters are core to all configuration mechanisms of H2c. They are of three kinds: company characteristic, perspective, and combined configuration parameters (cf. figure 1). Each of these groups is of the following structure: A group (e.g. perspective) contains dimensions (e.g. role), each dimension comprises dimension occurrences (e.g. manager). Combined configuration parameters are different and do not fit into this structure. Even though they are of similar nature as dimensions, they are more similar to their occurrences. They are necessary e.g. to represent intersections of dimension occurrences (sets) with configuration parameters.

Configuration parameters can have relations with each other. The selection of one configuration parameter can entail either that another parameter is selected as well (inclusion) or that another configuration may not be selected any more (exclusion). The software supports the user with ontologies that comprise this information. Ontologies are stored as rules.

An overview of the necessary data structures for the implementation of configuration mechanisms gives figure 11. The entity types and relationship types shaded in gray are additional ones to those displayed in figure 10.

Figure 11: Configuration Parameter

The entity types *ObjectOccurrence* und *ObjectDefinition* are essentially the same as those that are used for models. Since configuration parameters are handled like regular models, this is the logical consequence. Accordingly, *ConfigurationParameterDefinition* is the equivalent to *ObjectDefinition* and *ConfigurationParameter* is the equivalent to *ObjectOccurrence*. This applies similarly for their relationship type. A Rule or respectively an ontology defines the relation between two *ConfigurationParameterDefinitions*. Its relationship to *ConfigurationParameterType* specifies whether it is an inclusion or an exclusion rule. The attribute *isSelected* assigns *ConfigurationParameterDefinitions* to a specific Reference Model. These pre-selected *ConfigurationParameters* can be utilized by Terms and can be assigned to users of the reference model. Any parameter that is not pre-selected cannot be used in the configuration.

As mentioned above the meta model of configuration parameters can be adapted. Its original form should, however, only be changed by modeling experts. The canonical form of the configuration language in H2 notation is the following:

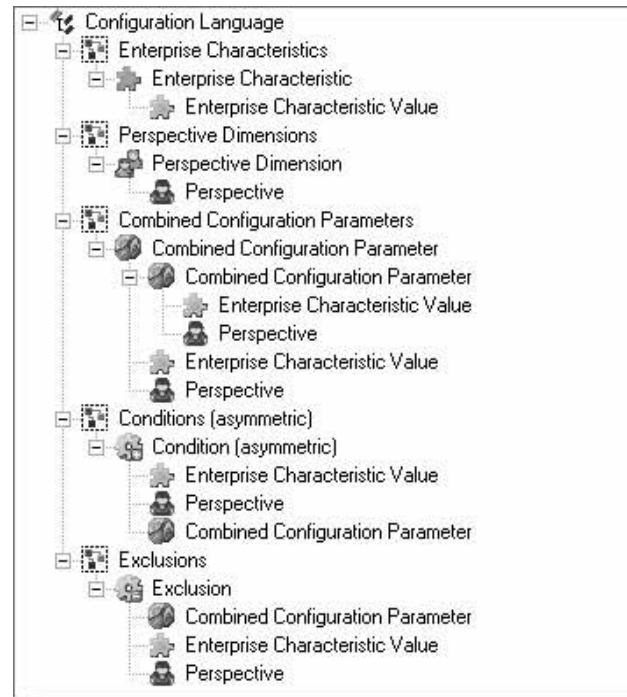


Figure 12: Meta Model of the Canonical Configuration Parameters and Rules

The core configuration process in H2c is roughly the following:

1. *Specify configuration parameters*: According to the above meta model, configuration parameters are specified (cf. figure 13).

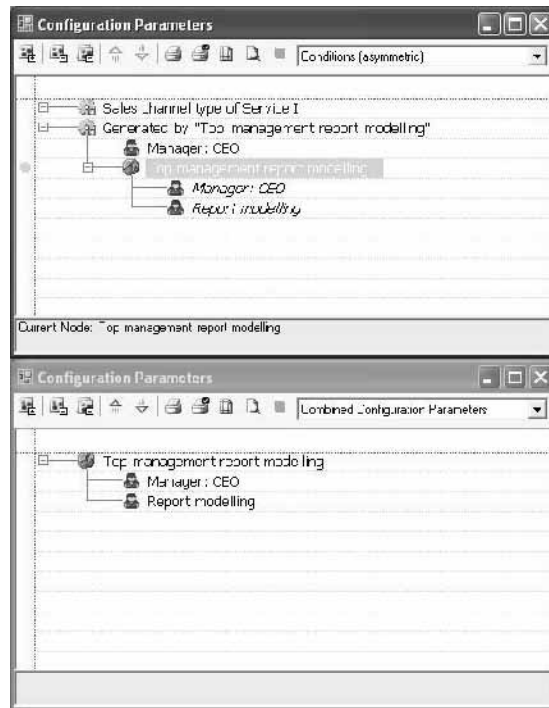


Figure 13: Specification of Configuration Parameters

2. *Pre-select configuration parameters:* From these configuration parameters the ones required for this configuration project are selected beforehand (cf. figure 14).

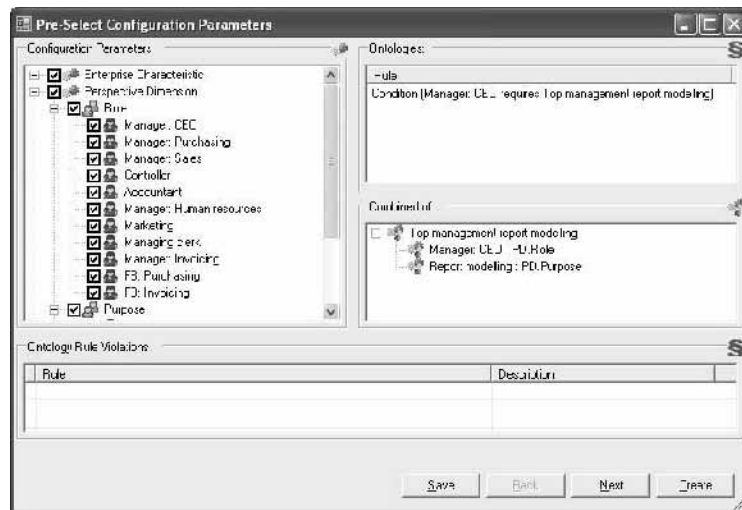


Figure 14: Pre-selection of Configuration Parameters

3. *Define configuration rules:* With configuration customizer and term editor these configuration parameters are assigned to model and meta model elements and their applicability is defined. The configuration itself is checkbox-based (cf. figure 15).

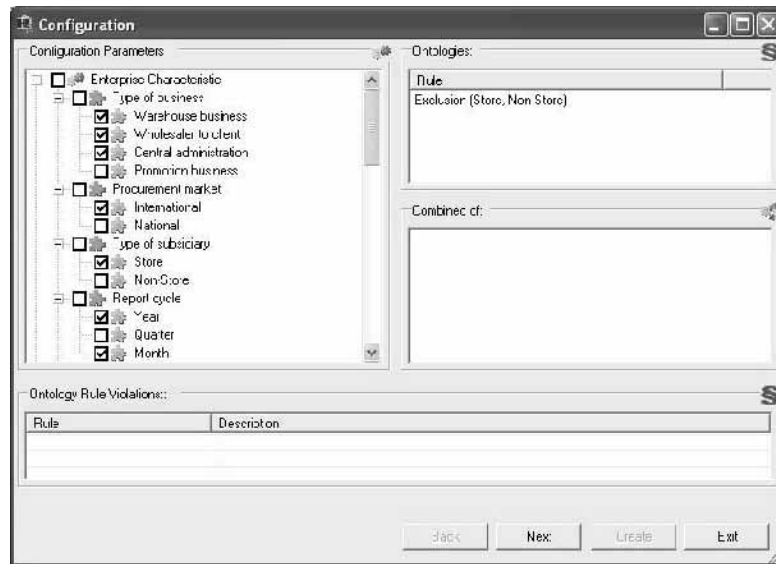


Figure 15: Model Configuration

### 3.3 Lessons learned

Summarizing, the format of the project seminar provides an ideal playground for prototypical implementations such as *adapt(x)* and *H2c*. As presented, even complex topics such as meta modeling or rather meta meta modeling and their implementation can be conducted by students self dependently. This is only possible since the seminar is conducted during a later stage of the studies and is open only to graduate students. Furthermore, students were prepared to perform a meta modeling seminar by lectures in the early graduate studies that deal with meta modeling in high detail. Other seminars on meta modeling have shown that such a detailed preparation is highly eligible due to the high complexity and abstraction of this topic. Exchange students from other universities that were not provided with such a preparation turned out to be overstrained in becoming acquainted with that topic in a short time.

In meta modeling seminars, it is important to give the students free room to operate. Only if the students feel responsible for “their” project, they identify with the topic and the assignment and produce superior results. It is, however, important to provide them with a flying start as it has been done with the workshops since the timeframe is limited and has to be divided between conceptual and implementation work. The preparation of the seminar papers leads to student “experts” for the topics prepared. These specialists are first contact persons for discussing topic-related concepts and problems. Presenta-

tions of the seminar papers within the initial workshop provide first insights for the whole group about the conceptual background of the implementation project. Therefore, careful assignment of important seminar topics to potentially superior students can have significant impact on the overall performance.

It has proven to be beneficial to let the students organize the course of the projects themselves (i.e. the project plan) with the exception of milestones which should be provided and monitored by the supervisors. In this way, the students are motivated to reach the objectives with their implementation plan and, thus, prove their procedure.

Three issues have been found to be of major importance when monitoring the project. First of all a clear statement concerning the milestones is important. At first, the topic and the assignment is very diffuse for the students and it only clarifies if the ultimate goal of the project is clearly specified. Otherwise, “something” is produced as a project result, but it most certainly does not meet the expectation of any party. Second, the time gap between major milestones has to be bridged with minor issues as topics for the weekly briefings. Otherwise an ongoing effort is hard to uphold, since work is only done when it is due. Third, the designation of contact persons (i.e. a single point of contact) is important to evade competence and responsibility issues. This choice should be taken by the students, but has to be approved by the supervisors. Prior experience shows that this choice is a crucial point for the performance of the complete team.

As regards the teaching of meta modeling in specific, it could be observed that oneself is the best teacher. Through unattended engagement in practicing meta modeling the students are forced to solve their problems through intensive discussions with other fellow students. In this way, they can much better appreciate the power of modeling language design and adaptation. As a matter of course, this requires tool support since paper-based or Visio-based assignments do not enable comparable understanding. Since meta modeling appears very complex at first, a step-by-step procedure is essential.

A feasible teaching procedure is to provide an example language with which the students can model to understand the constraints imposed in the meta model. Then, through careful adaptation of the meta model, they can value the impact of change towards the models. Finally the design of own modeling languages tops off the meta modeling experience. It has been proven beneficial to have a WYSIWYG language editor that basically utilizes the same constructs as the modeling editor in the same toolset. Due to the H2 Toolset’s specificity to hierarchical languages, the scope of modeling was limited and ultimately made the learning process easier. It was mentioned as an advantage of hierarchically ordered models that they always appear properly arranged.

Concerning external software, it has shown that prior knowledge of the status of the software to work with is very important. While H2 was freely available, the implementation with ARIS proved to be problematic at best.

This leads to the final lesson learned: Always have a fallback plan in mind in case you work with an external partner. Nothing is more time consuming and frustrating than trying to work around problems or even sheer walls put up by others that you cannot by-

pass. This was not particularly the case during this seminar, but it showed that working with an external partner always involves a certain amount of uncertainty.

## 4 Conclusions and Outlook

While several approaches exist that provide conceptual specifications on how to configure and adapt models, no comprehensive implementation has been accomplished up to now. In an ambitious teaching case, the implementation of configuration mechanisms for two modeling tools has been conducted as a proof of concept. The implementation has been done by a group of self organized students while the academic supervision was conducted by research assistants. The resulting prototypes were the result of an intense effort by the students and consequently reached the high goals set at the beginning of the project seminar. Several obstacles had to be worked around but in the end two comprehensive implementations could be presented.

*H2configurative* is a specialized meta modeling software that allows the modeling of hierarchies and components thereof. *H2c*'s core construct is a reference model, which contains as well the modeling languages as the actual models. Configuration mechanisms allow the adaptation of *H2c*'s modeling languages and models by means of configuration parameters annotated with terms. The structure of the configuration parameters can also be subject to change via meta modeling. Configuration parameter selection is supported by ontologies that prevent illicit use. *H2c* has been successfully evaluated configuring report definitions for the Retail-H, a retail reference model [BS04]. Furthermore, the applicability of *H2c* for the configuration of core components for UBL [BM06], OAGIS [Op06], and similar ontologies for business semantics is work in progress.

As a direct result of the “inaccessibility” of ARIS a general model configuration tool, *adapt(x)* was developed, which provides the basis to modify basically any model from any modeling environment that can be exported to XML. Further research on this general applicability has to be conducted and test cases with other common (meta) modeling tools have to be performed and evaluated. They are scheduled to take place later this year in the course of a seminar on meta modeling languages.

Another logical next step – already mentioned above as a “second step” – is the integrated configuration of models for operational and planning purposes. For this endeavor a meta configuration concept has to be developed that allows the integrated configuration of two or more modeling tools.

## References

- [Be02] Becker, J.; Delfmann, P.; Knackstedt, R.; Kuropka, D.: Konfigurative Referenzmodellierung. In Wissensmanagement mit Referenzmodellen - Konzepte für die Organisations- und Anwendungssystemgestaltung (Becker, J.; Knackstedt, R., Eds.), Physica, Heidelberg, 2002, S. 25-144.

- [Be03] Becker, J.; Dreiling, A.; Holten, R.; Ribbert, M.: Specifying Information Systems for Business Process Integration: A Management Perspective. *Information Systems and e-Business Management*, 1 (3) 2003, S. 231-263.
- [Be05] Becker, J.; Janiesch, C.; Seidel, S.; Brelage, C.: Specifying Modeling Languages with H2. In *Proceedings of the 1st Workshop on Meta-Modelling and Corresponding Tools (WoMM 2005)* (Frank, U.; Jung, J.; Kirchner, L., Eds.), Essen, 2005, S. 9-11.
- [BM06] Bosak, J.; McGrath, T.: *Universal Business Language 2.0 Public Review Draft*, 2006. <http://docs.oasis-open.org/ubl/prd-UBL-2.0.2006-02-22>.
- [BS04] Becker, J.; Schütte, R.: *Handelsinformationssysteme*. 2nd Edition. Redline Wirtschaft, Frankfurt am Main, 2004.
- [Ch76] Chen, P.P.-S.: The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1 (1) 1976, S. 9-36.
- [De06] Delfmann, P.: *Adaptive Referenzmodellierung: Methodische Konzepte zur Konstruktion und Anwendung wieder verwendungsorientierter Informationsmodelle*. Berlin 2006.
- [DS96] Darke, P.; Shanks, G.: Stakeholder Viewpoints in Requirements Definition. *Requirements Engineering*, 1 (1) 1996, S. 88-105.
- [Ga05] Gartner, Inc.: *Vendor Details for the 4Q04 Enterprise Architecture Tool Magic Quadrant*, Berwyn 2005.
- [Ho01] Holten, R.: The MetaMIS Approach for the Specification of Management Views on Business Processes. *Arbeitsberichte des Instituts für Wirtschaftsinformatik No. 55* (Becker, J.; Grob, H.L.; Klein, S.; Kuchen, H.; Müller-Funk, U.; Vossen, G., Eds.), Münster, 2001.
- [Ho03] Holten, R.: Specification of Management Views in Information Warehouse Projects. *Information Systems*, 28 (7) 2003, S. 709-751.
- [Kn06] Knackstedt, R.: *Fachkonzeptionelle Referenzmodellierung einer Management-Unterstützung mit quantitativen und qualitativen Daten: Methodische Konzepte zur Konstruktion und Anwendung*. Berlin 2006.
- [Op06] Open Applications Group, Inc. (OAGi): *Open Applications Group Integration Specification (OAGIS) Release 9.0*, 2006. <http://www.openapplications.org/downloads/oagidownloads.htm>. 2006-02-22.
- [RA05] Rosemann, M.; van der Aalst, W.M.P.: *A Configurable Reference Modelling Language*. *Information Systems*, In Press 2005.
- [RG00] Rosemann, M.; Green, P.: Integrating multi-perspective views into ontological analysis. In *Proceedings of the 21st International Conference on Information Systems*, Brisbane, 2000, S. 618-627.
- [Ro03] Rosemann, M.: Application Reference Models and Building Blocks for Management and Control. In *Handbook of Enterprise Architecture* (Bernus, P.; Nemes, L.; Schmidt, G., Eds.), Springer, Berlin, 2003, S. 595-615.
- [Sc00] Scheer, A.-W.: *ARIS - Business Process Modeling*. 3rd Edition. Springer, Berlin, 2000.
- [Sc98] Schütte, R.: *Grundsätze ordnungsmäßiger Referenzmodellierung. Konstruktion konfigurations- und anpassungsorientierter Modelle*. Gabler, Wiesbaden, 1998.
- [SDG03] Soffer, P.; Golany, B.; Dori, D.: ERP Modeling: A Comprehensive Approach. *Information Systems*, 28 (9) 2003, S. 673-690.
- [St96] Strahringer, S.: *Metamodellierung als Instrument des Methodenvergleichs - Eine Evaluierung am Beispiel objektorientierter Analysemethoden*. Dissertation. Shaker Verlag, Aachen, 1996.
- [Wi03] Wigand, R.T.; Mertens, P.; Bodendorf, F.; König, W.; Picot, A.; Schumann, M.: *Introduction to Business Information Systems*. Springer, Berlin, 2003.