

# Templates in Domain Modeling – A Survey

Christian Kop and Heinrich C. Mayr

Alpen-Adria-Universität Klagenfurt,  
Department of Applied Informatics / Application Engineering  
Austria  
{chris,mayr}@ifit.uni-klu.ac.at

**Abstract.** Conceptual modeling is often strongly related to a graphical language. Since the 80s, template-based approaches have also been proposed. However, they seldom reached the same popularity as graphical approaches. Nevertheless, template-based approaches are also important for collecting and representing information. This chapter will give a survey of approaches that used and use templates. It will be shown, how these templates are used and why their role is important.

**Keywords:** template-based approaches, conceptual modeling, predesign.

## 1 Introduction

Conceptual modeling is often strongly related to using a graphical language. The Unified Modeling Language (e.g., UML) and other languages like ER, ORM, Data Flows, etc. are good examples of this relationship. Many parts of UML are diagrams for specific purposes during software engineering (e.g., class diagrams, activity diagrams, use cases, state charts, object interaction diagrams). Class diagrams help to model the data structure of information systems. Use cases model the functionality the information system must provide to its users. Finally activity diagrams, state charts and object interaction diagrams help to model behavior (i.e., behavior of the system, components or single objects). Class Diagrams, use case and activity diagrams are used in the late stage of requirements engineering (domain modeling). Class diagrams, state charts, object interaction diagrams are typically used during the design phase in software engineering.

Although graphical languages like UML are very popular, they only show parts of information needed to develop a system. For more information, template-based approaches have been also proposed. However, they were mostly seen as an “add on” to graphical approaches.

This chapter will give a survey of approaches that used templates which are not based on a graphical (diagrammatic) representation. It will show how these templates are used and why their role is important.

In order to do this, the chapter will be structured as follows. Section 2 gives an introduction into the several categories of templates. Notions are be defined and a categorization is made. Section 3 firstly provides some thoughts why graphical approaches are so popular, but afterwards it also discusses why template-based

approaches are important and thus should not be ignored. Section 4 gives a survey on existing template-based approaches. The next section provides the description and historical summary of a special approach which was developed by the authors (Section 5). Section 6 concludes with a visionary idea and scenario of how template-based approaches can be integrated into the whole software development cycle in the future. The chapter is summarized in Section 7.

## 2 Types of Templates

Before starting with the description and history of template-based approaches, it is necessary to define what is meant here by the term template.

A template here is any kind of textual information which follows a structure. This means, it is possible to know the meaning of some information according to its position within the structure. Templates can be further divided by their purpose and usage into **forms** and **controlled language sentences pattern**.

A form is a template that has predefined information and empty slots for each of the predefined information. These empty slots have to be filled out. The process of filling the empty slots is supported by the predefined information. In other words according to the predefined information to which an empty slot belongs, the reader of the form can conclude about the “semantic” of the slot’s content and knows what to do with it (i.e., what kind of information must be entered and/or checked).

A form can be further refined to **attribute-value-pair lists, tables, matrices, cubes, n-dimensional tables** and **glossaries**. If the form is provided in a tabular format with two columns (attribute, value) and in each row the attribute column is filled with predefined information and the value column can be filled out, then it is an **attribute-value-pair list**.

If the form information is repeated several times and the predefined slots can be extracted and used as column header information because they remain invariant in this repetition then the term **table** will be used.

If a second dimension is added to the table, then it is called a **matrix**. Further dimensions can be added. Then this becomes a **cube** (3 dimensions) or a multidimensional table, in general (n-dimensions). In this case the header information is only invariant with respect to a certain dimension.

Cubes and Multidimensional Tables are only mentioned for the sake of completeness and they will not be mentioned further. They are not commonly used as a mean for communication in Requirements and Software Engineering since two dimensions can be easily described and visualized on a sheet of paper or in standard software products (e.g., Word, PowerPoint, Excel etc.).

The term glossary is commonly defined as follows. A **glossary** is a list of terms together with their definition. From the point of view of a representation, a glossary consists at least of two columns, the column with the term and the column with the definition of terms. Hence a glossary is a table. Beyond this, a glossary has a primary column which contains the terms which must be described. The rows in the other columns support the description of these elements.

A **controlled natural language sentence pattern** is a template in which all information is given, but the semantic of a certain kind of information can be derived from its position within the structure (i.e., the grammar). Each sentence follows this very restrictive grammar. Controlled languages are:

*“Subsets of natural languages whose grammars and dictionaries have been restricted in order to reduce or eliminate both ambiguity and complexity”<sup>1</sup>*

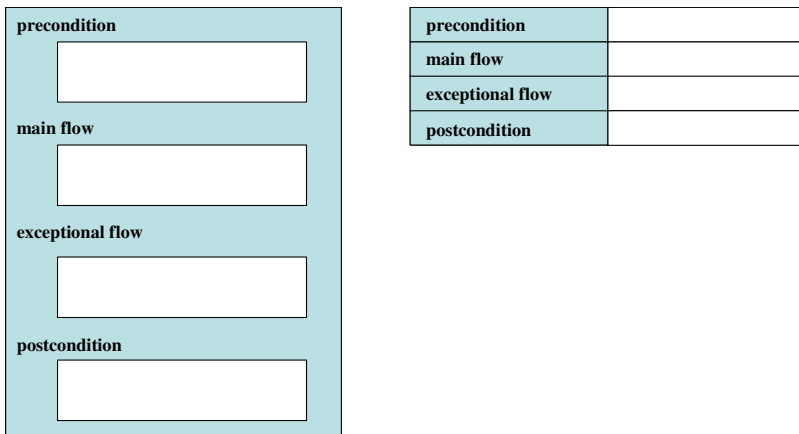
Although the grammar in a controlled language is restrictive, they have a main benefit. The information expressed as a sentence is unambiguous. Furthermore, according to the syntactical structure of a sentence a writer of a controlled language sentence knows which correct combination of word categories he must use in order to get a well formed sentence. For instance, if the pattern is followed that a well formed sentence should have the form *subject, predicate object* he will certainly not forget the subject or object. Furthermore if he knows that a verb needs an agent (someone/something that initiates execution of a process) he will not forget and the model becomes much more complete.

Furthermore, templates can be divided according to their level of abstraction into

- templates representing the schema level and
- templates representing the instance level.

### 2.1 Templates for the Schema (Model) Level

In a form that describes the schema, the predefined information belongs to the meta-model. Each of the empty slots belongs to a certain schema (model) element. Figure 1 shows some examples for a general form and an attribute value pair list.



**Fig. 1.** Example for a form and an attribute-value pair list

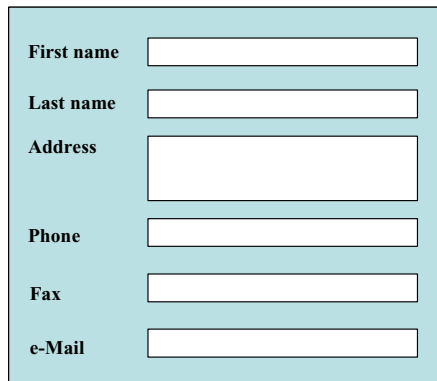
<sup>1</sup> See: <http://sites.google.com/site/controllednaturallanguage/> (last access: Sept., 27 2010)

A controlled natural language sentence might be “*The customer orders many products*”. Based on the grammar conclusions can be made about certain words in the context of modeling (e.g., nouns become classes or attributes, verbs become relationships, verbs with an agent become candidates for actions in scenario steps etc.).

## 2.2 Templates for the Instance Level

Templates can also be used on an instance level. Instead of specifying the meta-model in the predefined information and the schema in the empty slots, the predefined information is on the schema level and the empty slots must contain the instances of this schema. The previously mentioned notions form, table, matrix and glossary can be found here as well. Typical examples for forms, tables and matrixes are user interfaces, Excel-Sheets, forms for invoices, tabular reports etc.

They are also useful for design since the predefined information is useful for the construction of the schema. The information which has to be entered into the empty slots must be managed by the schema and therefore is useful for concrete test cases.



|                   |                      |
|-------------------|----------------------|
| <b>First name</b> | <input type="text"/> |
| <b>Last name</b>  | <input type="text"/> |
| <b>Address</b>    | <input type="text"/> |
| <b>Phone</b>      | <input type="text"/> |
| <b>Fax</b>        | <input type="text"/> |
| <b>e-Mail</b>     | <input type="text"/> |

**Fig. 2.** Example of a form at the instance level

Controlled language pattern can also provide information on the instance level. In this case, instead of common nouns, word categories like proper nouns and numbers are used in combination with common individual nouns. These proper nouns and numbers represent the instance level.

## 3 Graphical Modeling Languages vs. Template-Based Approaches

Before talking about template-based approaches, graphical approaches will be discussed.

### 3.1 An Advocacy for Graphical Modeling Languages?

There are a number of reasons why graphical approaches are so popular:

- Humans are visual beings. They perceive and explore the world through their eyes. Other senses - like the sense to hear, taste, feel, and smell something - are often used secondarily.
- Pictures, graphical symbols (e.g., cave paintings), statues and totem-poles were the first models of mankind.
- Pictures and graphical representations can provide us a spatial overview and an overall description of a physical thing, a phenomenon or a problem. This is why it is often said that a “picture is worth a thousand words”. Due to the good visual comprehension of human, they can better “read” a fact from an overall picture than from a sequence of letters (e.g., the Latin alphabet).
- Pictures and graphical representations (if it does not belong to graphical art) is often made exactly just for the purpose to offer a summary (e.g., bar charts in business domains).
- A conceptual model in informatics is often compared with a blue print for the construction of a material object (e.g., house, car, bridge etc.). To be trained to think in that way during the education also has an influence on the skills to read and interpret certain model representations.
- Finally, it is often argued that according to the above mentioned advantages, these graphical representations are a good basis for later steps. Even model to model transformation is possible. The class diagrams strongly used in model driven architecture (MDA) are representatives for that argument.

### 3.2 Limitations of Graphical Modeling Languages

According to the above subsection, a graphical representation for a modeling system offers advantages. However, a more detailed look to graphical representation techniques also shows that they are not optimal in every situation and for every reader.

In order to read or interpret the graphical description, it is necessary that the “reader” understands the semantics of notions. If a person does not understand an underlying notion (e.g., the general idea of a class, an attribute or association) then a graphical representation (e.g., a UML class diagram) is useless for a reader who does not have the necessary skills. If the “reader” does not know that \* means “many” in a detailed description of an association, he will not understand in which way the concepts are related to each other. The same happens if we see today a cave painting which was created ten thousand years ago. Although we see different kinds of painted animals we do not understand anymore what the meaning of the complete picture was. We can only make vague assumptions or guesses.

Even if the modeling concepts of the graphical representation are semantically understandable, there is still a question concerning the number of concepts visible in the graphic or picture. If there are only a dozen, then the human being of course can

realize and interpret all of them at once. However, if there are many (e.g., hundreds) of them, the reader will get lost and the graphical representation loses its ability to be an overview which can be interpreted immediately. Imagine, for example, a class diagram with 100 classes or a Use Case diagram with 100 use cases and actors related to these use cases. In such cases, human beings must introduce abstraction mechanisms (clustering of classes, use cases) to structure the picture.

The different kinds of notions represented in a graphical design are the second problem. If there are too many, the reader may get confused. Imagine for example a class diagram that shows everything: classes, attributes, associations, visibilities, data types of attributes, multiplicities, roles, stereotypes of classes and associations, comments on all modeling concepts and much more. Such a class diagram will also likely become unreadable since the reader will not know on which concepts he has to focus. He will be overwhelmed with too much information which he cannot interpret at once. Conceptual modeling tools therefore often offer features to hide information (e.g., to hide attributes, roles, visibilities etc.).

Beside the skills of the reader, often the intended purpose of usage has a great influence on the type of model that is used. If the overview character that is provided very well by a picture or graphic is of less importance, then the graphical representation may have less importance. Imagine for example a situation where a questionnaire or pattern is needed. Typically the structure of a questionnaire (i.e., pre defined questions) gives hints to the person who uses the questionnaire regarding which questions are important. Although many software engineers like to see themselves as architects of new systems, if they work in the field of requirements engineering, this “self description” is only conditionally valid. During the early phase of software engineering the relationship between a software (requirements) engineer and another stakeholder (e.g., end user) is like the relationship between a medical doctor and his patient. It is the responsibility of the doctor to extract the causes and later on find solutions for the causes. From the end user he will only get symptoms. Without questionnaires and patterns (at least in mind) he will not be able to find the causes. If requirements engineering experts do not use these questionnaires and patterns explicitly then this does not mean that they are never used since these experts already have internalized the questionnaires and can use them very flexibly, based on the state of communication with the end user.

The final argument for graphical approaches (i.e., that they provide a good basis for subsequent steps) is not always true. Consider for instance a use case diagram. The core of the diagram consists of use cases, actors, relationships between use cases, and relationships between use cases and actors. With these information alone transformation to a subsequent step is not possible. It is a template-based approach – the uses case description – that allows a transformation (at least manually). Only with information about preconditions, exceptional flows, main flow and other parts of the use case description, the designer gets an imagination of the details and hence an understanding about the behavior of the future software system. In general a graphical representation is not responsible for the success of a model transformation. The underlying modeling notions of the source and target model are responsible for the success of a transformation. If a match between the notions of the source and target model is possible, then also a transformation is possible.

### 3.3 Why Template-Based Approaches?

According to Section 3.1 it seems that template-based approaches cannot compete with graphical approaches and graphical representation are better for human beings. According to Section 3.2 graphical approaches have limitations. Hence, template-based approaches and graphical approaches can complement each other and there is no reason why graphical approaches alone should dominate. The next figure shows that a template in the form of a glossary can be used as a questionnaire. The columns represent hints for questions which must be asked if a cell of a column and row is empty.

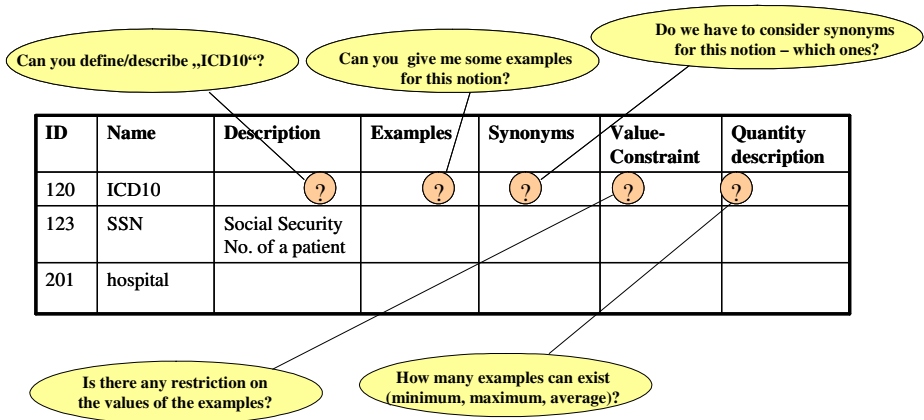
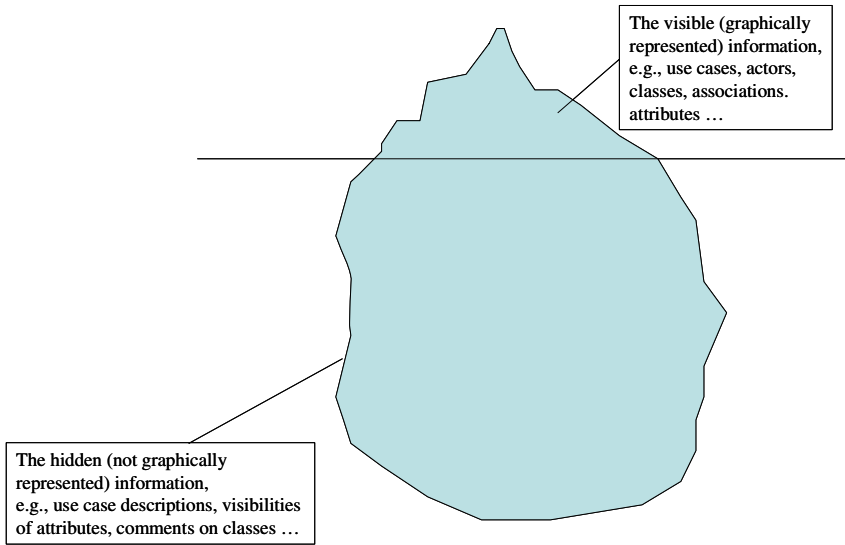


Fig. 3. A KCPM Glossary as a questionnaire

If these and other information would appear in a class diagram, then it would be confusing. Therefore, a glossary can complement a diagram in such a way that it provides the involved stakeholders with information that might be necessary for software development. Use cases are another example for such a symbiosis. The use case diagram is a nice picture to describe a certain aspect of the future software, however the most essential information are hidden in the use case descriptions. Thus the relationship between some graphical approaches and template approaches can be seen as an “iceberg”. Only a small percentage of the information appears graphically. The rest, which is represented by templates, is hidden under the “water”.

### 3.4 Template-Based versus Diagrams – Comparison Studies

In the previous subsections pros and cons for diagram-based and template-based modeling languages were mentioned. In this subsection some studies found in literature are presented. These results are only exemplary for specific kinds of templates.



**Fig. 4.** Iceberg of schema information

A study [44] was presented which compared textual use cases (TUC) and sequence diagrams (SSD) in the domain of safety system analysis. Textual use cases were tables with the following columns: user actions, system response, threats and mitigation. In each row of the table a piece of text described the user action, a possible system response as well as the threats and mitigations. For the system's sequence diagram (SSD) the UML notation was used. The textual use case (TUC) was used for interaction between the system and the user. The SSD was used for the interaction between the system and the user and also for internal system communication. The main feature of safety systems is to guarantee safety and avoid failures. During analysis of such systems it is necessary to detect risky situations (failures which might occur). For the comparison of the two presentation techniques, two groups of people (one in Norway, the other in France) were asked to find failures. Statistical methods (t-test, proportional test) were used for answering the following research questions: Is a Textual Use Case (TUC), in general, better than SSD for identifying failure modes? If it is better, are there problem areas where SSD is better? If TUC is better, are there specific failure modes where SSD is better?

According to the tests, the following were found: TUC is better to identify failure modes related to functionality of the system and the operator. However, for failure modes which occur within the system, SSD is better.

Another study [9] focused on the comparison of tabular representation and bar charts in order to represent production statistics or income statistics. In this experiment people with two different educations (business and engineering) were asked to answer questions concerning income or production based on tabular and graphical representations (bar charts). The time which was needed to get the answer and the number of correct and incorrect answers were measured. The following results were found. There is no difference between two educations concerning the



time which is needed to get the answer. If the questions become more complex, the time grows to answer it. Independent of their education however, all persons working with tables could answer the question much faster. Concerning the right answers however, there was an educational difference. People with an engineering background were much better using the graphical representation. On the other hand, people with a business background were better at using the tabular representation.

Another study [4] came to the conclusion that graphical approaches are better for displaying instructions. A flow of instructions was presented in five formats: flowchart, logical tree, yes/no tree, decision table and list. Furthermore a control panel in a current state was presented. In total, the control panel was able to have eight states. The subjects who did the experiment had to find out the instruction path which leads to a certain state. This was done by pressing one of the eight button (each button represented a state) in each of the formats. According to the representation, the buttons were located at the end of a path (flow chart) beneath a leaf (decision tree) or in a row (in the decision tables). The number of correctly pressed buttons was measured. The results showed that subjects who worked with decision trees make much more errors. Furthermore, it took longer to find the solutions if decision tables and lists are used instead of the three other graphical representations (flow chart, logical tree, yes/no tree). However it also turned out that subjects who worked with a certain kind of format also preferred this format after switching to another format. The only exceptions were subjects who used lists.

To summarize, the pros and cons of these different representations are still debated. The same study [9] which focused on graphs and tables for statistical information, listed in their related work many other results which reported the superiority of graphic representation and studies which reported the reverse.

## 4 A Historical Survey of Important Template-Based Approaches

The main stream of techniques belongs to graphical methodologies. In the 70s and 80s methods like SADT, dataflow diagrams, entity relationship diagrams were used to model different aspects of a software system. The latest representative, UML covers static (class diagrams) as well as dynamic aspects (sequence diagrams, state charts, activity diagrams and use cases) of a system using graphical models.

### 4.1 Forms, Tables, Matrices, Glossaries

Apart from graphical representations, template-based approaches were also used since the 70. Parnas [39] used tabular representations for the description of software functions. The rows and columns represented value ranges of two possible input parameters  $x$  and  $y$ . The cells described the result depending on the value range combinations of the two input parameters.

In the 80s, the DATA ID approach [7] used glossaries as a central concept in their methodology. The generation of glossaries was embedded in an engineering process. Firstly the universe of discourse was divided into organizational units. A characteristic of an organizational unit is the homogenous usage of terms (i.e., no homonyms or synonyms exist in an organizational unit). Secondly users and their tasks which must be provided by the information system were identified. A matrix

was generated which showed which user can tell information about which tasks. This matrix was the basis for interviews. The designers asked each user details about his tasks. The results of the interviews were written down as requirements statements. These requirements statements were categorized into data, operation and event statements. Finally the information in each statement was more refined and collected into entries of data, event and operation glossaries. The glossaries were the basis for traditional conceptual schemas (ER diagrams and Petri nets).

KCPM adopted these glossaries as a modeling methodology. A detailed survey of KCPM is given in Section 5.

In the 80s and in the 90s, object-oriented analysis and design were the buzz words of software engineering. Graphical languages like the Object Modeling Technique (OMT), Booch's method, Jacobson's Object Oriented Software Engineering (OOSE) became very popular. In the mid nineties, the previously mentioned languages were merged to form UML (Unified modeling language). UML once again was a graph-based modeling language (with some exceptions e.g., OCL). Though object oriented graphical languages gave good spatial overview, one deficiency was detected. What is the best granularity for a class? How large should a class be? Does an end user have the same understanding of the size and features of a class as the designer has? Do all designers share the same understanding of a certain class? It is interesting that these questions were not answered with the graphical opportunities given in OMT, the Booch method or UML. Instead Class Responsibility Cards were introduced in [3] and [51] as a mean to promote better communication between several stakeholders. For each class, an easily understandable template was generated. This template consists of the class name and a list of responsibilities. A responsibility is a short natural language description (i.e., one short sentence) of what the class offers to its environment. A responsibility is not a class method. It is only a general description of something that will be handled by one or more class methods. Together with these templates, guidelines were given (e.g., how many responsibilities a class should have in the best case). According to the number of responsibilities a certain class has, the stakeholders made decision if a class should be divided into more fine-grained classes or not.

Although the graphical representation of use cases introduced in OOSE were adopted in UML and became very popular, there was a need to complement them with a template [10]. Cockburn added additional, very important information to use cases and also explained how use case descriptions should be applied during requirements engineering. Hence, someone interested in the detailed model could learn much more from use case descriptions than they could learn from the graphical representation of a use case. A use case description is a form that has the following items which must be filled out:

*Use case name, preconditions for the use case, primary actors, secondary actors, main (normal flow), alternative flows, exceptional flows, post conditions.*

Today use case descriptions are widely accepted as templates which store detailed information for a certain use case.

The latest approach using form templates is NDT [15], [16]. NDT classifies requirements in *storage*, *actor*, *functional*, *interaction* and *non functional requirements*. For each of these requirement types, NDT provides a special template

that has to be filled out during requirements elicitation. Particularly if there is a certain storage requirement (e.g., *customer*) or functional requirement (e.g., *DVD rental*) then a template is made for it. The two templates are typical examples of storage requirements and functional requirements, respectively. A template for functional interaction is similar to a use case description.

**Table 1.** Template for a NDT storage requirement (SR) according to [15]

|               |  |        |
|---------------|--|--------|
| SR-01         | Customer                                       |        |
| Description   | Natural person who has the right to rent DVDs. |        |
| Specific Data | Name and description                           | Nature |
|               | Customer id: identification for the customer   | String |
|               | Name: the field stores the customer's name     | String |
|               | Address: the field stores the postal address   | String |

**Table 2.** Template for a NDT functional requirement (FR) according to [15]

|                 |  |   |
|-----------------|--|---|
| FR-01           | rent DVD   |   |
| Description     | Renting process of a DVD which is offered and executed by a clerk in the DVD rental store. |   |
| Actors          | Clerk  | AC-01: Clerk  |
| Normal sequence | Step   | Action  |
|                 | 1  | The system provides possibilities to search for DVDs                                  |
|                 | 2  | The clerk searches for a certain DVD  |
|                 | 3  | The clerk enters the Customer ID  |
|                 | 4  | The clerk selects DVD and mark it as rented for customer with given Customer ID       |
| Exception       | Step   | Action  |
|                 | 2  | DVD not available or all the requested DVDs are rented – restart with 1 or stop.      |
|                 | 3  | Customer ID not valid – stop the process  |
|                 | 3  | Customer is not yet a registered customer, continue with FR02 (customer registration) |

**4.2 Forms on the Instance Level**

On the instance level, forms were used as input for schema design [2]. They took forms used in offices and analyzed their structure. They identified several types of structures which can appear on a form (e.g., parametric text, structured frame, descriptive parts). In combination with their previously mentioned glossary approach (see DATA ID in Section 4.1) they proposed a design consisting of three steps.

- Form analysis and area design,
- Form design,
- Interschema integration.

In the first step, the concepts which appear in certain areas in the form and their relationships to other concepts are examined. This knowledge is stored in glossaries. During area design, knowledge about the form areas is taken to map each area to a corresponding conceptual schema. During the second step, the derived schemata are integrated for each form. Instead of several schemata which belong to the several areas of one form only one integrated schema for the whole form remains. The last step is another integration step. The schemata for the several forms are once more integrated to an overall first draft conceptual schema.

Whereas the analysis of forms was done manually [2], other work [11] presented a computer supported tool which derives a conceptual schema from a user interface.

Further works on user interfaces (forms) and their influence on database schemas were made [14] [47]. Forms and user interfaces were used as a user centered mean to query the database.

### 4.3 Controlled Language Sentences

Though, controlled natural language now is a buzzword, the idea behind it was introduced in the 80s. Synonymous terms for controlled natural language are sentence patterns and structured language.

For a better understanding of his Entity Relationship (ER) approach, Chen [8] proposed 11 heuristics to map sentences to an equivalent ER schema. He used 11 structured English sentences and showed on the basis of their structure how the elements of such a sentence can be mapped to elements of the ER schema (i.e., entity types, relationship types or attributes).

Other research results (e.g., [5], [6], [30], [36], [37], [46], [48]) complemented and refined these initial results. Some of them provided automatic transformation from controlled language sentences by using parsers.

In [12] the functional grammar defined in [13] was used as the linguistic basis. Dik's verb patterns describe semantic roles which nouns can play in combination with a certain verb. The result of these studies was the Color-X model [49]. It is a graphical approach, but it is based on linguistic templates.

In [19] the term controlled language is explicitly used. There a controlled language is applied for paraphrasing of ontologies.

In [32] an artificial controlled language ("*Orthosprache*" / "*Normsprache*") was developed. Natural language was the basis as well as logic. The artificial language only contains words which are needed to describe the domain (e.g., nouns and verbs). Words like prepositions and articles do not appear in this language. All notions of such a language must be well defined. For instance if the term customer is used then all stakeholders know what is meant. The same holds if another term e.g. "to order" (customer order products), is used. During requirements analysis, the main aim of the language is to support the deeper understanding of semantics of concepts ("*Fachbegriffe*") which are used in a certain domain.

Since controlled language avoids ambiguities, it is not only used as a basis for conceptual modeling but also during requirements engineering. It is the task of the requirements engineer to break down complex and ambiguous sentences to controlled language requirements statements [38].

#### 4.4 Controlled Language Sentences on the Instance Level

A typical representative for a controlled language on the instance level is NIAM [31]. Structured language generation and analysis were the proposed procedure to get a graphical schema. Once again templates were important and were seen as a support. The idea was that the stakeholders should describe the domain with simple example facts (e.g., *The student with student id XYZ visits the course with course number ABC; The course ABC can be visited by the student with student id XYZ*). From the set of facts, a graphical model consisting of object types (e.g., student, course) with connected roles (student visits; course can be visited) can be generated. This model evolved and is now called ORM [21].

### 5 The Story of KCPM

KCPM (Klagenfurt Conceptual Predesign Model) is a template-based modeling language that was initiated and developed in the Application Engineering research group at Klagenfurt University.

#### 5.1 The Beginnings

The idea to use glossaries was proposed by one of the authors according to his practical experiences [28]. The DATA ID approach, which was published a few years earlier, was introduced as an effective approach to communicate with the end users.

#### 5.2 Research

Three master theses examined different aspects of the DATA ID approach. One focused [17] on the transformation rules between glossaries and natural language sentences. Another thesis [40] extended the model of the DATA ID by improving the concept of event type (later called co operation type). The last one [23] focused on the structural and functional aspects (thing types, operation types). The DATA ID data glossaries were refined to thing glossaries (later called thing type glossaries). Also the DATA ID operation glossary was refined (later called operation type glossary). Since the representation was intended to be applied to any modeling concept of the new approach, also a connection (type) glossary was introduced. This master thesis also combined the glossary representation approach of DATAID with the fact-oriented approach used in NIAM. Therefore an algorithm was developed which maps thing types and connection types into entity types, value types, attributes and relationships of an entity relationship model. During that thesis also the name and acronym (KCPM) was born: *KCPM = Klagenfurt Conceptual Predesign Model*. With this name it is pointed out that the approach must be applied in between requirements analysis and conceptual modeling. It is a model that helps the software engineer to ask the right questions during requirements engineering but also has the advantage that the conceptual model can be generated easily from these working results. It is “conceptual” since there must be an agreement upon the language notions. Since it supports the development of the final conceptual **design model** it was called a **predesign model**.

A few years later a PhD thesis [24] started with the aim to integrate the results of the first research studies. The outcome was a first, core, lean model based on a small set of notions namely:

- **Thing types (concepts):** Notions/Object types which are important for the domain.
- **Connection types:** Relationships between thing types.
- **Operation types:** Services provided by the future system or components of the system.
- **Cooperation types with pre- and post conditions:** Behavior that triggers operation types.

Another aim of the PhD thesis was to build bridges between KCPM and natural language sentences. Therefore, KCPM was embedded into the NIBA<sup>2</sup> project. NIBA was a German acronym for natural language based requirements analysis. Computer linguists developed a grammar model for natural languages sentences called NTMS (Naturalness Theoretical Morphosyntax) [18], [27]. The NTMS was taken as the basis to examine the relationships between KCPM glossaries and natural language text.

There is still ongoing research on the KCPM topic. Meanwhile there is a PhD project that examines the possibilities of schema integration using the modeling notions of KCPM [50]. Another PhD project focuses on the questions of user centered visualization of web services. Template-based visualization is seen as one good possibility [20]. There are also research connections to other research groups. In 2006 a study started with the aim to map between KCPM thing types and connection types to CBEADS smart business objects [26]. Two master thesis [29],[52] focused on different mapping aspects from KCPM to the OLIVANOVA conceptual model [33], [34] and [35]. Another master thesis [41] analyzed the usage of thing types and connection types for teaching modeling in informatics education at a high school. There is ongoing research with the University of Kharkiv on aspectual and qualitative predesign [22],[42]. Furthermore KCPM is adopted as an ontology representation in software engineering [1] in a joint project with the University of Marburg/Lahn.

### 5.3 Practical Studies

The research was also complemented by practical studies. After the first master theses on this topic were finished, a student working in a local publishing company used the approach to collect their requirements. It was demonstrated, that a domain expert could validate the requirements very well using the collected information in the glossaries.

In another master thesis [45], a further practice study with KCPM was done. The primary goal of this master thesis was to investigate possible usage of KCPM in the domain of business process modeling. Since the student already worked in a software development enterprise which needed this information, the student was told to ask the involved stakeholders in which situations glossaries can be preferred over graphical representations. This study pointed out that glossaries are very well understood and preferred in situations where a listing is necessary, or information can be presented as check lists. Hence thing type glossaries, connection type glossaries as well as

---

<sup>2</sup> NIBA was funded by the Klaus Tschira Stiftung Heidelberg.

operation type glossaries were seen as the best candidates for a glossary representation (i.e., list of terms = thing type glossary, list of relationships = connection type glossaries, list of services of the system = operation type glossaries). The interviewed persons only had problems with cooperation type glossaries because of their complexity. A cooperation type glossary contains the set of operation types together with their preconditions and postconditions. The persons argued that a graphical representation is more suitable. This study was a very interesting hint for further research.

Other practical experiences were gathered in a medical domain. The content of this requirements engineering project were data driven. New software was necessary to manage the data, to get statistics from the database and for decision support of the users. We found that thing type and operation type glossaries could be applied very successfully for the collection of all the requirements.

To summarize, parts of the KCPM approach could be applied in several practical projects. The results were twofold. On the one hand it could be shown that there is a need for template approaches. On the other hand, the feedback and experiences from the practical studies were used to further improve the approach.

#### 5.4 Beyond Glossaries – Current State of KCPM

The arguments for glossaries and graphical approaches in Section 2 and the practical studies of Section 5.3 showed that glossaries are important but graphical approaches must always be considered. It would be a mistake to ignore graphical approaches. The reasons are simple:

- The different skills of different users must be always considered. Some of them like templates and others like graphical approaches.
- The situation (purpose) is always changing in a typical requirements engineering project. At one point in time, the requirements engineer must act like a medical doctor extracting the causes from the symptoms of the patients (end user). In the very next moment, he must give an overview to the end user or he needs the overview for his own better understanding.

Because of these reasons, in one study [25] the main research goal was not to discuss why and in which situation glossaries are better rather to think of how combining different kinds of representations. Particularly:

*How could a graphical representation be combined with template representations?*

The conclusion was: Graphical and template-based representations must be seen as equally valuable views within a toolset.

## 6 Conclusion and Future Vision

### 6.1 Conclusion

A recent paper [43] describes how to switch among different aspects of a model (e.g., the dynamic aspects, static aspects etc.) within a meta-model. However, this was presented once again for a graphical representation. Particularly, based on the meta-model a tool provides a view with a class diagram and a view with the use case

diagram, the meta-model guarantees consistency between the views. In other words, a use case is not longer seen as an independent model within UML where relationships to classes or other UML concepts are not checked.

The idea to switch between different aspects of a model consistently must be combined with different representation techniques. It must not only be possible to switch between classes and use cases but also to switch between a graphic representation and a template-based representation.

## 6.2 Templates in SW Development – Future Vision

In order to describe our view of the future, firstly we comment on the past - in particular something on the evolution of programming languages. At the beginning of software development, assembler languages were used to program the computer. In these assembler languages important and very commonly used combinations of low level operations were clustered into macros. However these languages had the disadvantage that they were close to the machine. The developers had to keep in mind to which memory address and registers they have to store values. Third generation languages like Cobol and Fortran were the first step towards a more human readable programming language. These programming languages were used in specific domains. Cobol was developed and used for implementing business solutions. Fortran was mainly used in mathematical domains. During those days scientists also tried to understand what is essential for good programming. The solutions were programming languages for structured programming (e.g., ALGOL, Pascal etc.). Further style guides (e.g., how to decompose large software into units that can communicate with each other without side effects) lead to further evolutions of 3rd generation programming languages to module and object based languages and then to object oriented languages (e.g., Modula-2, Modula-3, Smalltalk, C++, Java etc.). The idea to make programming languages more user-understandable was also realized in the 4th and 5th generation languages. These languages were developed for certain kind of applications (e.g., SQL for database queries, LISP and Prolog mainly for problemsolving situations). These languages were more human readable since they focused on the “WHAT” and not on the “HOW”. Using these languages the user had to specify WHAT he needed and not how it should be executed. This was once again achieved by hiding some internals (e.g., in SQL the user need not know the access paths to records in a database; in Prolog he need not know the technical details about backtracking but can rely on the Prolog interpreter to fire rules based on the available facts).

If we summarize this, then it can be learned that during the evolution of programming languages, complexity was hidden and style guides as well as patterns were introduced. During the evolution of 3rd generation programming languages the goal always was to “transform” 3rd generation language code to efficient machine readable code in a 2nd or even 1st generation programming language. With model driven architecture (MDA), 3rd generation programming languages became the final target of transformation processes. Nowadays, MDA is based on the idea that every platform independent model (PIM) will be extended with platform specific features. Doing this, the PIM is transformed to a platform specific model (PSM) which itself can be the PIM for the next transformation step. What is done now in MDA is an



evolution like the evolution of program languages in the past. Scientists came to an overall understanding about good programming. Therefore programming languages are now the target of transformation. Model driven architecture can be also seen as a next step to make modeling more human understandable. During the evolution of the programming languages, the main focus was to exempt the languages from machine-specific pieces of code. During model driven architecture one of the ideas is that pictures often represent more than 1000 lines of code. Once again certain kinds of macros are introduced. Classes can be specified with its attributes but there is no need to specify their constructors and their get- and set-methods. These methods are automatically derived from the class specifications.

Combining this information with the knowledge about templates described in this paper, one vision of the future might be the following. Scientists will come to a better understanding about model driven architectures, based on style guides, design and architectural patterns. They will improve the quality of model driven architecture. At the end we will get a common understanding what a PIM needs in order to be of good quality. Having this, the question will not be any longer to get executable software or a completed source code according to a certain specification. Instead the focus will be on how to get a specification (the first PIM) from requirements.

Templates will support this new construction idea. Instead of trying to get a graphical model, requirements engineers will behave like medical doctors to collect all the necessary information and to generate the target graphical model. Such a working step will focus on extracting structured requirements specifications from unstructured requirements specifications. Templates will play an important role within this step. As an intermediate result, for quality checking the stakeholders will mainly work with

- template (e.g., glossary) entries and
- only if really necessary with a graphical model
- that represents either the final PIM or an intermediate version.

On any of these intermediate results the stakeholders will be able to make corrections. End user and designer together will be able to check the templates.

This will also be a further step towards human readability of models. The human reader is now of another kind. It is no longer a technically experienced user but also an application domain expert with little technical knowledge. Thus, in the future modeling a software system will be like going through a specific checklist for this software.

## 7 Summary

This chapter presented a survey on template-based conceptual modeling approaches. Historical approaches were presented. Although some of them are mainly known as graphical modeling techniques, they are based on templates (e.g., forms, glossaries and in most cases linguistic templates). The aim of this chapter was to create awareness, that the usage of graphical representation is not always the best solution. It

strongly depends on the stakeholder's skills and situation. In certain situations, a template is better suited than a graphical representation. However the best usage of graphical and template-based techniques is always a situation depended combination of these techniques.

**Acknowledgments.** The authors would like to thank Dr. h.c. Klaus Tschira for his support of the NIBA project. Without this support the extensive research on this interesting topic would not have been possible. Furthermore, we thank all the colleagues and students which have worked in the area of conceptual predesign. Finally we would thank the reviewers for their helpful hints and comments with which it was possible to improve this chapter.

## References

1. Bachmann, A., Russ, A., Vöhringer, J., Hesse, W., Mayr, H.C., Kop, C.: OBSE - an Approach to Ontology-based Software Engineering in the Practice. In: Reichert, M., Strecker, S., Turowski, K. (eds.) Proceedings of the 2nd International Workshop on Enterprise Modeling and Information Systems Architectures. GI Lecture Notes in Informatics (LNI), vol. 119, pp. 129–142. Köllen Verlag (2007)
2. Batini, C., Demo, B., Di Leva, A.: A Methodology for conceptual design of offices data bases. *Information Systems* 9(2-3), 251–264 (1984)
3. Beck, K., Cunningham, W.: A Laboratory For Teaching Object-Oriented Thinking. In: Conference Proceedings on Object-oriented programming systems, languages and applications, pp. 1–6. ACM Press, New York (1989)
4. Boekelder, A., Steehouder, M.: Selecting and Switching: Some Advantages of Diagrams Over Tables and Lists for Presenting Instructions. *IEEE Transaction on Professional Communication* 41(4), 229–241 (1998)
5. Buchholz, E., Cyriaks, H., Düsterhöft, A., Mehlan, H., Thalheim, B.: Applying a Natural Language Dialog Tool for Designing Databases. In: Proc. International Workshop on Applications of Natural Language to Databases (NLDB 1995), pp. 119–133 (1995)
6. Buchholz, E., Düsterhöft, A., Thalheim, B.: Capturing Information on Behaviour with the RADD-NLI: A Linguistic and Knowledge Based Approach. In: Riet, v.d., Burg, R.P., Vos, A.J. (eds.) Proceedings of the 2nd Int. Workshop on Applications of Natural Language to Information Systems (NLDB 1996), pp. 185–192. IOS Press, Amsterdam (1996)
7. Ceri, S. (ed.): *Methodology and Tools for Database Design*. North Holland Publ. Comp., Amsterdam (1983)
8. Chen, P.P.: English Sentence Structure and Entity Relationship Diagrams. *Int. Journal of Information Sciences* 29, 127–149 (1983)
9. Coll, R.A., Coll, J.H., Thakur, G.: Graphs and Tables a Four-Factor Experiment. *Communications of the ACM* 37(4), 77–86 (1994)
10. Cockburn, A.: *Writing Effective Use Cases*. Addison Wesley Publ. Comp., Reading (2000)
11. Choobineh, J., Mannino, M.V., Tseng, V.P.: A form-based approach for Database Analysis and Design. *Communication of the ACM* 35(2), 108–120 (1992)
12. Dignum, F., Kemme, F., Kreuzen, W., Weigand, H., van de Riet, R.P.: Constraint modelling using a conceptual prototyping language. *Data & Knowledge Engineering* 2, 213–254 (1987)

13. Dik, S.: *Functional Grammar*. North Holland Publ. Company, Amsterdam (1978)
14. Embley, D.W.: NFQL: The Natural Forms Query Language. *ACM Transactions on Database Systems* 14(2), 168–211 (1989)
15. Escalona, M.J., Reina, A.M., Torres, J., Mejías, M.: NDT a methodology to deal with the navigation aspect at the requirements phase. In: *OOPSLA Workshop: Aspect-Oriented Requirements Engineering and Architecture Design* (2004)
16. Escalona, M.J., Koch, N.: *Metamodeling the Requirements of Web Systems*. In: *Proceedings of the 2nd Int. Conf. Web Information Systems and Technologies (WebIST 2006)*. *Lecture Notes in Business Information Processing (LNBIP)*, vol. 1, pp. 267–280. Springer, Heidelberg (2006)
17. Felderer, A.: *Zur Tabellarisierung natürlichsprachlicher Anforderungsbeschreibungen*. Diplomathesis, Universität Klagenfurt (1992)
18. Fliedl, G.: *Natürlichkeitstheoretische Morphosyntax – Aspekte der Theorie und Implementierung*. Gunter Narr Verlag, Tübingen (1999)
19. Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., Schneider, G.: *Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines*. In: Eisinger, N., Maluszynski, J. (eds.) *Reasoning Web*. LNCS, vol. 3564, pp. 213–250. Springer, Heidelberg (2005)
20. Gälle, D., Kop, C., Mayr, H.C.: *A Uniform Web Service Description Representation for Different Readers*. In: Berntzen, L., Smedberg, A. (eds.) *Proceedings of the second International Conference on the Digital Society (ICDS 2008)*, pp. 123–128 (2008)
21. Halpin, T., Bloesch, A.: *Data modelling in UML and ORM: a comparison*. *Journal of Database Management* 10(4), 4–13 (1999)
22. Kaschek, R., Kop, C., Shekhovtsov, V.A., Mayr, H.C.: *Towards simulation-based quality requirements elicitation: A position paper*. In: Rolland, C. (ed.) *REFSQ 2008*. LNCS, vol. 5025, pp. 135–140. Springer, Heidelberg (2008)
23. Kop, C.: *Herleitung von EERM+ Schemata aus Zusammenhangsverzeichnissen, erweiterten Ding- und Operationsverzeichnissen*. Diplomathesis, Universität Klagenfurt (1993)
24. Kop, C.: *Rechnergestützte Katalogisierung von Anforderungsspezifikationen und deren Transformation in ein konzeptuelles Modell*. Doctoral thesis, Universität Klagenfurt (2002)
25. Kop, C.: *Visualizing Conceptual Schemas with their Sources and Progress*. *International Journal on Advances in Software* 2(2,3), 245–258 (2009), <http://www.iariajournals.org/software/> (last access September 27, 2010)
26. Liang, X., Ginige, A.: *Smart Business Object - A New Approach to Model Business Objects for Web Applications*. In: *ICSOFTE 2006*, pp. 30–39 (2006)
27. Mayerthaler, W., Fiedl, G., Winkler, C.: *Lexikon der Natürlichkeitstheoretischen Morphosyntax*. Stauffenburg Verlag, Tübingen (1998)
28. Mayr, H.C., Dittrich, K.R., Lockemann, P.C.: *Datenbankentwurf*. In: Lockemann, P.C., Schmidt, J.W. (eds.) *Datenbank-Handbuch*, pp. 486–552. Springer, Heidelberg (1987)
29. Michael, J.: *Connecting the dynamic part of KCPM with the OlivaNova Modeler*. Masterthesis, Universität Klagenfurt (2010)
30. Moreno, A., van de Riet, R.P.: *Justification of the equivalence between Linguistic and Conceptual Patterns for the Object Model*. In: *Proc. 3rd Int. Workshop on Application of Natural Language to Information Systems*, pp. 57–74 (1997)
31. Nijssen, G.M., Halpin, T.: *Conceptual Schema and Relational Database Design – A fact oriented approach*. Prentice Hall Publ. Comp., Englewood Cliffs (1989)

32. Ortner, E.: *Methodenneutraler Fachentwurf*. B.G. Teubner Verlagsgesellschaft, Stuttgart, Leipzig (1997)
33. Pastor, O., Molina, J.C., Iborra, E.: Automated Production of Fully Functional Applications with OlivaNova Model Execution. *ERCIM News* (57), 62–64 (2004)
34. Pastor, O., Gomez, J., Insfran, E., Pelechano, V.: The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Information Systems* 26(7), 507–534 (2001)
35. Pelechano, V., Pastor, O., Insfran, E.: Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques. *Data and Knowledge Engineering* 40(3), 315–353 (2002)
36. Rolland, C., Ben Achour, C.: Guiding the Construction of textual use case specifications. *Data & Knowledge Engineering Journal* 25(1-2), 125–160 (1998)
37. Rolland, C.: An Information System Methodology Supported by an Expert Design Tool. In: Pirow, P.C., Duffy, N.M., Ford, J.C. (eds.) *Proceedings of the IFIP TC8 International Symposium on Information Systems*, pp. 189–201. North Holland Publ. Company, Amsterdam (1987)
38. Rupp, C.: *Requirements Engineering und Management*. Hanser Verlag (2004)
39. Ryszard, J., Parnas, D.L., Zucker, J.: Tabular Representations in Relational Documents. In: Hoffman, D., Weiss, D.M. (eds.) *Software Fundamentals – Collected Papers by David Parnas*, pp. 71–85. Addison Wesley Publishing Comp., Reading (2001)
40. Schnattler, M.: *Herleitung von Ereignisschemata aus erweiterten Operations- und Ereignisverzeichnissen*. Diploma thesis, Universität Klagenfurt (1992)
41. Schein, M.: *Moderne Modellierungskonzepte der Informatik in der Schulpraxis*, Diplomathesis, Universität Klagenfurt (2010)
42. Shekhovtsov, V., Kostanyan, A., Gritskov, E., Litvinenko, Y.: Tool Supported Aspectual Predesign. In: Karagianis, D., Mayr, H.C. (eds.) *5th International Conference on Information Systems Technology and its Applications (ISTA 2006)*. LNI, vol. P-84, pp. 153–165. Köllen Verlag (2006)
43. Sinz, E.: Tool-Unterstützung für die SOM-Methodik: Anforderungen und Lösungsstrategien. In: *Presentation slides: 1st International Open Models Workshop, Klagenfurt (March 2010)*
44. Stålhane, T., Sindre, G., du Bousquet, L.: Comparing Safety Analysis Based on Sequence Diagrams and Textual Use Cases. In: Pernici, B. (ed.) *CAiSE 2010*. LNCS, vol. 6051, pp. 165–179. Springer, Heidelberg (2010)
45. Stark, M.: *Geschäftsprozessmodellierung im konzeptuellen Vorentwurf*, Diplomathesis, Universität Klagenfurt (2000)
46. Tjoa, A.M., Berger, L.: Transformation of Requirement Specification Expressed in Natural Language into an EER Model. In: Elmasri, R.A., Kouramajian, B., Thalheim, B. (eds.) *Proc. 12th International Conference on Entity Relationship Approach*, pp. 127–149. Springer, New York (1991)
47. Terwillinger, J.F., Delcambre, L.M., Logan, J.: Querying through a user interface. *Data & Knowledge Engineering* 63, 774–794 (2007)
48. Vadera, S., Meziane, V.: From English to Formal Specifications. *The Computer Journal* 37(9), 753–763 (1994)
49. Van de Riet, R.: Mokum for Correctness by Design in Relation to MDA, In: Kaschek R., Kop, Ch., Steinberger, C., Fliedl, G. (eds.) *Information Systems and e-Business Technologies. Lecture Notes in Business Information Processing (LNBIP)*, vol. 5, pp. 352–364 (2008)

50. Vöhringer, J., Mayr, H.C.: Integration of schemas on the pre-design level using the KCPM-approach. In: Nilsson, A.G., Gustas, R., Wojtkowski, W.G., Wojtkowski, W., Wrycza, S., Zupancic, J. (eds.) *Advances in Information Systems Bridging the Gap between Academia & Industry*, pp. 623–634. Springer, Heidelberg (2006)
51. Wirfs-Brock, R., Wilkerson, B.: Object-oriented design: a responsibility-driven approach. In: *Conference proceedings on Object Oriented Programming Systems Languages and Applications*, pp. 71–75. ACM Press, New York (1989)
52. Yevdoshenko, N.: *A Proposal for Model Driven Development Life Cycle: From KCPM Requirements Specifications to OLIVANOVA Conceptual Model*, Master Thesis, Universität Klagenfurt (2006)