A User Centered Approach to Requirements Modeling

Heinrich C. Mayr, Christian Kop

Institute of Business Informatics and Information Systems
University of Klagenfurt[1]

{mayr|chris}@ifit.uni-klu.ac.at

Abstract: The paper argues that the conventional methods for object oriented analysis and conceptual modeling suffer from lacks in requirements elicitation and validation by the end-user. Therefore, an intermediate level called 'conceptual predesign' is introduced between natural language requirements specification and conceptual design. The paper introduces the basic notions of a conceptual predesign model and discusses heuristic rules for their automatic mapping to the conceptual level, e.g. to UML.

1 Introduction

The use of object oriented models is widely seen as a means to collect, to analyze and to document the user requirements w.r.t. an information system which is intended to be developed. A couple of different object oriented modeling methods has been introduced since the late eighties under the heading of 'Object Oriented Analysis' (OOA) e.g. [Bo94], [CY91], [Ja93], [Ki94], [Ru91], which today are mostly replaced by the use of the 'Unified Modeling Language' UML [BRJ99]. On the other hand, many designers in practice still work with classical extended entity-relationship diagrams. Apart from many differences, both modeling languages share a common ontology w.r.t. static/structural modeling aspects: the real world and especially those parts (Universes of Discourse, UoD) that are relevant for concrete information system development projects are assumed to consist of „objects" which have particular characteristics and which may be somehow related to each other. In addition, both languages provide concepts for managing complexity, namely semantic abstraction relationships like generalization and aggregation.

Despite of all this, information system projects often suffer from incomplete or inadequate models which are, using the before mentioned (object oriented) modeling methods, conceptual schemes. In our opinion, the reason for these problems results from the fact, that conceptual models are too complex and abstract as to be easily understood and validated by average users. Moreover, they demand early design decisions, e.g. between

[1] The work presented within this paper is part of the NIBA project ("Natürlichsprachige Informationsbedarfsanalyse", "Natural Language Based Requirements Analysis") which has been funded by the Klaus Tschira Stiftung, Heidelberg.

classes and value sets, or between associations and attributes, which are not application domain concepts and thus unfamiliar to the end-user.
On the other hand, end-users and their consultants are the stake-holders of the relevant concepts, notions and requirements of their Universe of Discourse (UoD). Requirements engineering, therefore, should start with collecting this knowledge and representing it in a way the end-user understands and is able to validate. We propose for that purpose the use of glossary-like schemes that are composed by means of a rather lean semantic model which we call *KCPM (Klagenfurt Conceptual Predesign Model)*. The notion 'conceptual predesign' is used since it is based on a semantic (meta)model and since it precedes the usual conceptual design. KCPM may be seen as an object oriented extension of an approach called DATA-ID which was presented first in the early eighties [Ce83].

KCPM offers a set of semantic concepts for modeling static and dynamic UoD aspects. It is based on an NIAM-like [NH89] ontological approach which treats UoD´s as systems consisting of interrelated elements (things) that are able to perform services (operations) and that are activated by events (messages sent by other things). Thus, there is a strong relationship to object-oriented approaches which allows for a rather natural way to map KCPM concepts to OOA concepts.

By this approach we try to reach the following three objectives:
- to provide for an user centered form of requirements documentation,
- to automate as far as possible the process of producing the predesign scheme by extracting the important UoD notions from natural language requirements specifications and collecting them[2],
- to automate as far as possible the mapping from the predesign scheme entries to a first cut conceptual scheme, e.g. formulated with the means of UML. Thus, a system analyst must not 'think' in UML-terms when analyzing requirements but can concentrate on collecting and relating glossary (i.e. predesign schema) entries representing relevant UoD aspects.

Within this paper we will concentrate on the first and on the third objective. For that purpose, section 2 outlines KCPM and it's notions. Section 3 explains the usage of our model. Section 4 then gives an overview of the mapping process. In particular we will show how our modeling notions thing-type and connection-type can be transformed into a scheme consisting of classes, associations, attributes and value-types by applying special rules which interpret the entries in our glossaries for a target UML scheme. In section 5 we will illustrate the mapping with a short example. Section 6 sketches first practical experiences.

2 KCPM

An important aim for the development of KCPM was to harmonize the developer's and

[2] NIBA results concerning this issue may be found in [Fl97], [Fl98]

the end-user's view of a given UoD, i.e. to provide an interface for their mutual understanding. As mentioned in the previous section, the approach was influenced by the DATA-ID model which we extended and specialized to provide at least the same semantic expressiveness as modern approaches to object oriented analysis do. The most important modeling notions of the ***static part*** of our approach are: ***thing-type***, ***connection-type***, ***perspective*** and ***constraint***. We shortly explain these notions and show by examples which meta-attributes are used for their representation in glossaries.

Thing-type is a generalization of the conceptual notions *class* (entity type or entity set, respectively) and *attribute* or a *legal domain (value-type)*. Thus, typical things (instances of thing-types) are

- natural or juridical persons,
- material or immaterial objects,
- abstract notions.

as well as

- descriptive characteristics of the above mentioned examples (e.g. a *customer name, a product number, a product description*) which can be seen as attributes or as specific legal domain determiner.

UoD-area: publishing **UoD:** publishing companies **Project:** P5

id#	name	classification	quantity-description	examples	value domain	synonym	description	requirement source
D001	author	thing-type		e.g. „. T. Mann“				S1, S2,
D002	book	thing-type	500					S1, S2, S3
D004	ISBN number	thing-type						S3
	...							

Figure 1: Part of a thing-type glossary[3]

Things are related within the real world. To capture this, we introduce the notion of ***connection-type***. Two or more thing-types can be involved in a connection-type. To define a connection-type completely, it must be described from the point of view (***perspective***) of all of the involved thing-types. This corresponds to the NIAM object/role model. Sentences leading to connections (and perspectives) are e.g. the following:

(S1) *Authors write books*. (perspective of *author*)
(S2) *Books are written by authors*. (perspective of *book*)

[3] There are two forms of KCPM schema representations: A graphical one and (tabular) one using glossaries. Within this paper we restrict ourselves to the latter. Especially for business people, glossaries seem to be more easy read and checked, in particular in the case of large schemes where tool supported groupings and aggregations are not interfered with layout rearrangements as may happen in graphical representations. The column headers of the glossaries represent KCPM meta-attributes of the resp. modeling notions. Traceability between all the glossary entries and their source (the natural language text) is given by means of so-called requirement source references.

The abstractions *generalization* („is-a“, with set inclusion on the instance level) and *component/object* („is part of“) are treated as specific connection-types. Furthermore we provide an *identification* connection-type with the perspectives *identifies* and *identified_by,* e.g.

(S3) *A ISBN number identifies a book.*

UoD-area: publishing **UoD:** publishing companies **Project:** P5

c-id#	name	connection type determiner	perspective					req. source
			pers-pective#	involved thing-type	name	min/ max	persp-det.	
C001	write/ is_written		p001a	D001, author	write			S1, S2
			p001b	D002, book	is_written			
C002	identification		p002a	D004, ISBN number	identifies			S3
			p002b	D002, book	is_identified_by			
...	...							

Figure 2: Part of a connection-type glossary

The model is open for further semantic connection-types (e.g. possession). Static UoD aspects that cannot be modeled using these notions are captured by (textual) constraints. This is not very sophisticated, but allows the designer to specify functional requirements as well as non functional requirements [So96].

The known conceptual models have different conceptions concerning the relations between attributes and classes, between two or more classes, and between attributes they permit. Hence design decisions often depend on the particular modeling concept of the model in use. The generalized KCPM approach allows to separate these model specific aspects from the requirements elicitation, because there is no a priori restriction. Moreover, the distinction between class and value type, and consequently between association and attribute, as is requested by conceptual models, is not quite natural to end-users and often leads to early design decisions that have to be revised later on. The generalization chosen by KCM induces a simplification for the end-user whose world just continues to consist of related (connected) things, and defers the conceptual distinction to later design steps.

3 Usage of KCPM

The glossaries form the basis of an information resource dictionary for the application in question and emphasize the scratch pad character. As such they are a link between natu-

ral language requirements specifications and the corresponding conceptual schema. The use of dictionaries that contain all relevant business notions of a given UoD, their synonyms etc. is quite common in application software development. However, such dictionaries mostly do not contain structural information. In contrast to that, KCPM supports such structural information and, therefore, comes with the following benefits:

- extended semantic power (e.g. connections, quantity descriptions of thing types, examples etc.) which leads to an
- enhanced transformation of dictionary entries to a succeeding conceptual model;
- more explicit information gathered in columns triggers more questions in the requirements elicitation phase. In a glossary like representation each empty field triggers at least the following question for all stakeholders: "*Why is this column empty?*", "*Is it necessary to fill the column with a value*?", "*If so, what is/are the right value(s)?*". Clearly, this supports completeness of the subsequent models/design.

Thus, the effort of filling structured glossaries instead of just generating a list of notions will be paid back. The stakeholder will have both, a dictionary of all relevant UoD aspects, which is easily to validate, and a conceptual predesign model[4], which may be systematically transformed into a first cut conceptual schema.

Taken into account our current practical experiences (see also sect. 6) and the structure of the glossaries, KCPM seems to be most useful in classical business information system projects, where data and databases play a major role (e.g. banking, insurance etc.).

Clearly, filling KCPM glossaries is a cumbersome task, which must be supported by appropriate tools. Any kind of requirements elicitation technique may be applied within that context. A more sophisticated way is to derive glossary entries by a linguistic approach out of natural language requirements specifications (texts, interview transcripts etc.). This is the main issue of the before mentioned NIBA project.

4 The mapping process

For the derivation of a conceptual scheme from glossary entries we propose an iterative process consisting of several phases and steps. It starts with a check of the glossaries w.r.t. completeness and inconsistencies (validation). If the entries are consistent and if both parties (designer and stakeholders) agree that the listed entries reflect all relevant UoD aspects and their interrelationships, then a stepwise mapping to the conceptual scheme is initiated. In this second step, mapping rules are applied which help to associate concepts of the target model to the glossary entries. In the case of UML, e.g., thing-types are mapped to classes or (attribute) value-types[5], connections to associations or

[4] Note that because of the nature of KCPM its usage is again a kind of conceptual design.

[5] Note that UoD aspects like *customer name* often are modeled as attributes in conceptual models, having, e.g., *String* as associated type (expression). In our approach, if a thing-type is not seen as a class on the conceptual level, it is mapped to a value-type. In this case, there must be a connection-type which relates the given thing-type to another one (e.g. *customer*) by a perspective, e.g. *has_customer_name*. This connection-type then will be mapped to an attribute of class *customer* and named by the perspective name. We then define, e.g. *String* as the value domain of *customer name*.

attributes or generalizations or aggregations. The next steps after the mapping aim at a refinement of the target model by restructuring and completion.

The mapping itself is guided by heuristic rules which we distinguish into *direct* and *indirect* ones according on how we can derive a decision. Direct rules determine a target notion immediately from glossary entries. Indirect rules use in addition results from previous mapping steps. Furthermore we distinguish *laws* and *proposals*: A *law* forces a specific mapping (e.g. a given thing-type to a class). If a designer does not follow the law, the conceptual scheme will become inconsistent. *Proposal* means, that the proposed mapping is more likely than another one. Thus, the designer may accept the proposal or take another decision. The set of 18 rules presented here concentrate on using information from the thing-type and connection-type glossary. Additional rules might be formulated, if the other KCPM glossaries (operation-type, co-operation-type etc.) are considered.

Rule 1 (Law)
A thing-type T is mapped to a class C_T, if T is specified as such by the designer in the classification column (previous designer decision).

Rule 2 (Law)
A thing-type T is mapped to a class C_T, if T is the only involved thing-type of a connection-type (reflexive connection-type; note that reflexive attributes make no sense).

Rule 3 (Law)
A thing-type T is mapped to a class C_T (an association class in particular), if T is a connection-type determiner of connection-type C_1 and is involved thing-type of another connection-type C_2 with $C_1 \neq C_2$ (objectified association).

Rule 4 (Law)
A thing-type T is mapped to a class C_T, if T is involved in a connection-type A by a perspective which has a perspective determiner.

To understand this rule consider our example in section 2: perspective determiners strongly correspond to what is called a role in UML and other conceptual models. Consequently, a thing-type the instances of which play a role within a connection should be mapped to a class.

Rule 5 (Law)
A thing-type T is mapped to a class C_T, if T is involved in a connection-type A by a perspective 'is_ identified_by'.

Rule 6 (Proposal)
A thing-type T may be mapped to a class C_T, if T is involved in a connection type A by a perspective having the minimal cardinality 0.

This proposal considers the minimal cardinality of a perspective as an existence dependence indicator: if the minimal cardinality is 0 then the resp. thing-type may exist without a connection to another thing-type. Such a situation is in our opinion a hint that the thing-type might be mapped to a class.

The following two rules have a linguistic background. We could have employed these rules in an earlier step, e.g. during linguistic analysis. However, if such an analysis is not performed during requirements analysis, we need it now for the mapping process.

Rule 7 (Proposal)
A thing-type T may be mapped to a class C_T, if T is involved in a connection-type A by a perspective whose perspective name matches with a predefined verb in a lexicon.

This proposal is based on the fact that perspective names are derived from verbs and on the assumption, that there are verbs from which we can conclude on the nature of the subject within a sentence built around that verb. E.g. *'X buys/sells Y'* allows to conclude that X is a natural or a juridical person. Usually, person is modeled as a class.

Rule 8 (Proposal)
A thing-type T may be mapped to a value-type if its name is the second member of a composition (the first member being mapped to a class).

The idea of this rule is, that value-type names often are members of a linguistic composition like *number* and *name* in the compositions *article number* and *customer name* respectively.

Rule 9 (Law)
A thing-type T is mapped to a value-type, if it is specified as such by the designer in the classification column (previous designer decision).

Rule 10 (Law)
A thing-type T is mapped to a value-type if T identifies another thing-type.

Rule 11 (Law)
A thing-type T is mapped to a value-type if T has an entry in the value-domain column.

This rule starts from he assumption that entries in the value-domain column always refer to atomic value-types.

Rules 1-11 are direct rules. In contrast to that the following rule 12-19 relate to preceding mapping decisions.

Rule 12 (Law)
A thing-type T is mapped to a class (to a value-type), if it is connected by a generalization connection-type to another thing-type which already has been mapped to a class (to a value-type).

This rule is derived from the fact that generalization is only possible between equivalent concept types.

Rule 13 (Law)
A thing-type T is mapped to a class (to a value-type), if it is connected by a aggregation connection-type to another thing-type which already has been mapped to a class (to a value-type).

Rule 14 (Proposal)
A thing-type T may be mapped to a class, if it is connected by connection-types to at least one thing-type, which has been previously mapped to a class and to at least one thing-type which has been previously mapped to a value-type.

Rule 15 (Proposal)
A thing-type T may be mapped to a value-type, if it is involved in only one connection-type (by minimal cardinality 1) and if it is neither a connection-type determiner nor a perspective determiner.

Rule 16 (Law)
A thing-type T is mapped to a class C_T, if T is involved in a binary connection-type and the other involved thing-type is already mapped to a value-type.

Rule 17 (Law)
A connection-type Co is mapped to an Attribute At_{Co}, if it is a binary connection-type and if one of its involved thing-type is mapped to a class whereas the other is mapped to a value-type.

Rule 18 (Law)
A connection-type Co with involved thing-types T_1 ... T_n is mapped to an Association if at least two of the involved thing-types have been mapped to classes previously

All the laws and proposals are now used together to get the mapping result. In particular the "mapping game" is as follows: Tell me what you know about a glossary entry (thing type). The idea behind this proces is: We cannot prevent stake holders to make mistakes and inconsistencies (please note that this is not a lack of our model). In fact with the mapping process we do not only want to classify glossary entries but we also detect some modeling inconsistencies. To achieve this, the rules given above must work acording to meta rules. They process indicate the 'priority' of laws w.r.t proposals, if more than one rule are applicable to a given situation.

	Class	**value-type**	**mapping**
1	law/proposal	---	class
2	---	law/proposal	value-type
3	Law	proposal	class
4	Proposal	law	value-type
5	Proposal	proposal	design decision
6	Law	law	contradiction

Table: meta mapping rules

As can be seen in the table a (at least one) law overrules a proposal (lines 3 and 4). A designer decision and a maintenance activity is necessary in two cases (lines 5 and 6). In the first case for both, classes and value-types only proposals are found. The designer should decide which mapping should be used. In the second case contradicting mapping

results indicate that the input is wrong. The reason could be wrong requirements or simply a wrong glossary entry. According to the contradicting rules, the designer has to check the glossary entries and optionally ask the stakeholder for a better specification. Then he has to change the entries to solve the conflicting situation.

The distinction between direct and indirect rules gives the mapping step an internal sequence. At the end of the mapping, meta-rules are applied to make a final check (E.g. *in a n-ary connection-type at least two classes and mapping solutions for the others - e.g. mapping to value-type must exist)*.

5 A simple example

We are now going to illustrate the previously outlined theory in the light of an example. This example completes the example sentences in chapter 2. For reasons of brevity it is a rather simple example and thus does not cover all aspects of KCPM, however, it should allow for a first impression of how our approach works. Nevertheless, it explains, that we do not only focus on notions but also on the interrelationships between these notions. Let us therefore suppose an UoD were we have do deal with publishing companies. The description of structural aspects is as follows:

(1) Publishers publish books. (2) Authors write books. (3) A book can be written by several authors.(4) An author has a unique name, a birth date and an address. (5) A book has a unique title and a prize. (6) An ISBN number identifies a book (7) Each book is published by exactly one publisher. (8) A publisher has a unique denomination and an address. (9) A publisher has employees. (10) An employee has a unique social insurance number, a name and a birth date.

For a better understanding of the mapping, assume that all entries (connection-types and thing-types) together generate a network of information as given in figure 3.

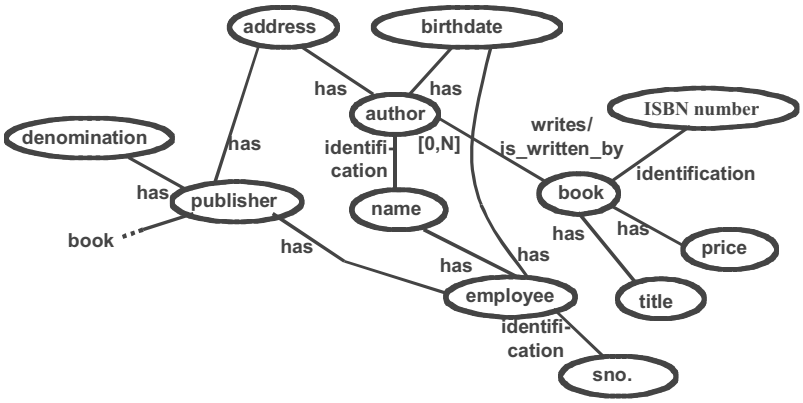

Figure 3: Graphical representation of the thing-type and connection type glossaries

Further assume that the phrases "*An author has a unique name*", "*An ISBN number identifies a book*" and "*An employee has a unique social insurance number.*" express

that name, ISBN number, social insurance no. (*sno*.) identify author , book, employee, respectively. Thus they led to special semantic connection-types between those thing-types. Now, the idea behind the mapping process is as follows: Start with thing-types and apply the mapping rules to them at first; then proceed to the connection types. E.g., we can apply **rule 5** and **rule 10** to the entries name , title, social insurance no., author, book, employee. We can also apply **rule 6** to author. If we have context information and know that every string which ends up with "*...name*", "*...title*", "*...denomination*", "*...no*.", "...number", "...price", "*date*" is a candidate for a value-type then **rule 8** will work. The underlying idea is simple, namely, seek for nouns which do not describe complex structures. Such nouns could be collected in a separate dictionary. According to these rules, we will derive the classes *book*, *employee*, *author* and the value-types *name*, *title*, *sno*, *denomination*, *price, ISBN number*. Now it is time to apply indirect rules. E.g., indirect **rule 14** may be applied to map publisher to a class, since it is connected to at least one value-type (see: *denomination*) and at least one class (see: *book,* resp. *employee*). No rule supports us to classify the thing-type *address*. In this case the designer has to decide. Let's say address will become a value-type. It is also possible, that a thing-type is classified as a value-type and a class by different laws. This is not a failure but an advantage. In that case, the designer gets aware that he has not modeled correctly. Finally we may apply **rule 17** and **rule 18** (mapping connection-types to associations and attributes). The resulting UML object model is depicted in figure 4. As can be seen, value-types, or in terms of UML named type-expressions, are user defined ones and not implementation oriented ones (like Integer, Date, String etc.).

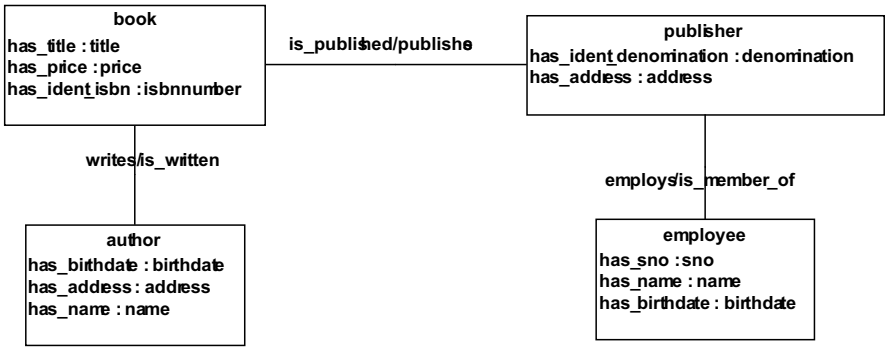

Figure 4: First cut UML-Schema

6 Experiences in practice

Though the whole approach including the automatic linguistic parsing and interpretation has not yet been tested in a comprehensive application project, there is a lot of experiences in evaluating the appropriateness of the KCPM approach in real life projects. The first project concerned the whole logistics of a big newspaper producer. All stake holders (from workers via clerks to managers) were confronted (after the requirements elicitation phase) with the static and dynamic parts of our model in order to validate them from

their point of view. We were very confirmed in our approach by the fact, that nobody had – after a short introduction into the glossaries and their intention – problems to understand and discuss these models. Based on the KCPM schema a contract with a big software producer was established and, in addition, the schema is used as a kind of 'enterprise handbook'.

Another interesting experience was made in the context of a master thesis [St00] which was intended to check possibilities to extend KCPM for business process modeling. Since this student works with a software producer supplying software for county government, the latter was chosen as a real life UoD for modeling and validation. The student checked his modeling results with employees of 5 different county governments. Four of them were immediately able to work with the static part of the schema and to validate its entries. Also, they gave rather positive and encouraging comments. In one case, a graphical notation of the schema (which, obviously is possible, see figure 3) was preferred. Difficulties arose in understanding the dynamic part of the schema. We actually work at a more intelligible form of representation for that part.

From all our practical studies we learned that a tool supporting the usage of KCPM is strictly necessary. A beta-version of such a tool is actually under development in the framework of the above-mentioned NIBA project. It's actual version supports the whole process (from natural language sentences via KCPM glossaries to the UML object model) for static UoD aspects based on a certain normalization of German sentences. Moreover it also performs a first cut function point analysis based on the resulting KCPM schema.

7 Conclusion

Conceptual predesign precedes the conceptual design and allows a more user centered way for requirements elicitation, analysis and validation. Clearly, to fill up KCPM glossaries is a cumbersome and time-consuming task even if it is supported by an appropriate tool. We therefore currently exploit mechanisms to extract glossary entries automatically from natural language requirements specifications. This is done in the joint project NIBA ([Fl00]). Within this paper we have focused on usage of KCPM and the mapping of the static part of an UoD model. The mapping provides a first modeling language transformation (transformation of basic notions). Clearly, KCPM also provides notions for the dynamic aspects of an UoD, and we are actually finishing the definition of rules for their mapping to the dynamic modeling concepts of UML (activity diagrams, state charts). This mapping is done in equivalent steps.

References

[Bo94] Booch, G.: Object-Oriented Analysis and Design with Applications. Benjamin/Cummings Publ. Comp., 1994.

[BRJ99] Booch, G.,; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley Publ. Comp. 1999.

[Ce83] Ceri, S. (ed.): Methodology and Tools for Database Design. North Holland 1983.

[CY91] Coad, P.; Yourdon, E.: Object-oriented Analysis, Prentice Hall, 1991.

[Fl97] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: „NTS based Derivation of KCPM perspective determiners“. In: Proceedings of the 3rd International Workshop on Applications of Natural Language to Information Systems (NLDB'97),26-27, Vancouver, Canada 1997, Vancouver, Canada; pp. 215 - 226

[Fl98] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: „Disambiguation of Part of Relationships within the NIBA Project“. In: DEXA98 Workshop Proceedings, Vienna, August 1998; pp. 171 - 175

[Fl00] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: Linguistically based requirements engineering – The NIBA Project. In: Data & Knowledge Engineering, Vol. 35, 2000, pp. 111 – 120.

[Ja93] Jacobson, I.; Christerson, M.; Jonson, B.; Övergaard, G.: Object-Oriented Software Engineering: A Use Case driven Approach. ACM Press, Addison-Wesley Publ. Comp., Wokingham, 1993.

[Ki94] Kristen, G.: Object Orientation, the KISS Method - From Information Architecture to Information Systems. Addison Wesley, 1994.

[KM98] Kop, C.; Mayr, H.C.: “Conceptual Predesign – Bridging the Gap between Requirements and Conceptual Design”. In: Proceeding of the 3rd International Conference on Requirements Engineering ICRE’98, Colorado Springs, April, 1998.

[NH89] Nijssen, G.; Halpin, T.A.: Conceptual Scheme and Relational Database Design – A fact oriented approach. Prentice Hall Publ. Comp. 1989.

[Ru91] Rumbaugh, J.; Blaha, M.; Premelani, W.; Eddy, F.; Lorensen, W.: Object oriented modeling and design, Englewood Cliffs, NJ, Prentice Hall, 1991.

[So96] Sommerville, I.: Software Engineering. Addison Wesley Publ. Comp., 3rd edition, 1996

[St00] Stark, M.: Business process modeling with konceptual Predesign (Geschäftsprozessmodellierung im konzeptuellen Vorentwurf) Master Thesis, Universität Klagenfurt, 2000.