

A User Centered Approach to Requirements Modeling

Heinrich C. Mayr, Christian Kop

Institute of Business Informatics and Information Systems
University of Klagenfurt¹

{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 pre-design’ is introduced between natural language requirements specification and conceptual design. The paper introduces the basic notions of a conceptual pre-design 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. [1], [3], [9], [10], [14], which today are mostly replaced by the use of the ‘Unified Modeling Language’ UML [15]. 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 classes and value sets, or between associations and attributes, which are not application domain concepts and thus unfamiliar to the end-user.

¹ The work presented within this paper has been funded partly by the Klaus Tschira Stiftung, Heidelberg.

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 [2].

KCPM offers a set of semantic concepts for modeling static and dynamic UoD aspects. It is based on an NIAM-like [13] 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 notions of the UoD from natural language requirements specifications and collecting them,
- 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. In a subsequent step, the predesign scheme then is transformed into a conceptual one (based on a given conceptual model). This transformation follows a set of heuristic rules and, thus, may be done automatically to a large extent (see figure 1).

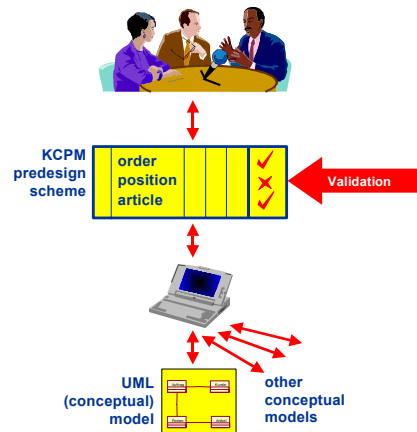


Figure 1: KCPM schemes serve as an interlingua and as communication basis

Within this paper we will concentrate on the first and on the third objective. For that purpose, section 2 outlines KCPM and its notions. Section 3 then gives an overview of the transformation steps followed by detailed discussions of the individual steps. 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 4 we discuss an example starting with structured sentences that are transformed into a conceptual schema. Section 5 sketches first practical experiences.

2. KCPM

An important aim for the development of KCPM was to harmonize the developer's and 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 give examples which show how to represent their usage within glossaries.

Thing-type

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.

Apart from the simplification this generalization induces for the end-user (in whose world everything just continues to be a „thing“), it also provides for more flexibility during linguistic analysis. For, although there are heuristics and rules to decide which parts of a natural language sentence correspond to a class and which do not, these do not work in all situations. In undetermined cases, however, using KCPM we can defer the decision to later transformation steps. Moreover, 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 linguistic analysis because there is no a priori restriction.

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) *a customer rents a car.* (perspective of *customer*)

(S2) *a car is rent by a customer.* (perspective of *car*)

The abstractions *generalization* („is-a“, with set inclusion on the instance level) e.g.

(S3) *a truck is a car*

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.

(S4) *a customer-number identifies a customer.*

The model is open for further semantic connection-types (e.g. *possession*).

Static UoD aspects that cannot be modeled using these notions (and their characteristics, see below) are captured by (textual) constraints. This is not very sophisticated, but allows the designer to specify functional requirements as well as non functional requirements [16]. The glossaries form the basis of an information resource dictionary for the application in question and emphasize the scratch pad character. Traceability between all the glossary entries and the requirements source is given by means of so-called requirement source references. In the default case, these references point to natural language sentences. Meta-attributes are used for characterizing aspects of our basic notions. These are represented as glossary columns. In addition, each glossary entry is equipped with a unique identification number.

Relevant thing-type meta attributes

name: Thing-type designator, directly extracted from the requirements document.

classification: information about the conceptual „equivalent“ (e.g. entity type, attribute) if extractable from the text. If it is not extractable it will be determined during the mapping process from the predesign to the design scheme.

quantity: the expected number of instances of the thing-type, e.g., derived from a sentence like

(S5) *we have 50 cars.*

examples: denominators of things observed in the UoD for which the resp. thing-type is introduced; they are derived from sentences like (S6) *Mr. Miller, living in Klagenfurt, is a typical customer.*

Value domain: natural language description of the legal value domain, e.g. from sentences like

(S7) *all rent rates are between 100\$ and 200\$.*

synonym: references synonyms (i.e. the resp. glossary entry)

description: narrative text on the resp. thing-type.

requirement reference: a pointer to the (natural language) documents/sentences where the resp. thing-type is referenced.

Relevant connection-type meta attributes

name: may be derived from one of the perspective names (see below).

description: narrative text describing the connection-type.

Connection-type determiner: references, if it exists, an „objectification“ of the connection-type by means of an existing thing-type.

additional cardinality: cardinalities which cannot be expressed in one perspective (see below).

per perspective

perspective name: name of the perspective - normally the verb of the specifying sentence in its active or passive form.

involved thing-type: thing-type connected via the responsible perspective.

optional: If there are situations where the involved thing-type needs not to be instantiated as part of a connection, this value is set to *True*.

cardinality (min, max): relation qualifier similar to numerous conceptual models: Each cardinality has a lower and upper bound. The lower bound can range between zero and a definite number. The upper bound must range between the lower bound (must have at least the value „1“) and an indefinite value „N“. If necessary, also number sets are allowed.

perspective determiner: circumscribes the involved thing-type of this perspective. E.g. think of a connection-type between thing-types *Author* and *Book* having perspectives with the names *wrote* and *is_written_by*, respectively. Then there could be a perspective determiner *writer* of perspective *wrote*.

The figures 2 and 3 show the corresponding glossary entries for the example sentences (*S1*) to (*S7*).

UoD-area: rentals			UoD: car renting		department: --			
id#	name	classification	quantity description	examples	value domain	synonym	description	requiremt. source
D001	customer	thing-type		Mr. Miller				S1, S2, S6
D002	car	thing-type	50					S1, S2
D003	truck	thing-type	20					S3
D004	customer-no	thing-type						S4
D005	rent-rates	thing-type			100\$ - 200\$			
D006	name	thing-type			String			

Figure 2: part of a thing-type glossary

UoD-area: rentals			UoD: car renting		department: --				
c-id#	name	connection type determine	de- scription	perspective					
				perspective#	involved thing-type	name	min, max	perspective determiner	
C001				p001a	D001 , customer	rent			S1, S2
				p001b	D002 , car	is_rent_by			
C002				p002a	D003 truck	Specialization			S3
				p002b	D002, car	Generalization			
C003				p003a	D001, customer	is_identified_by			S7
				p003b	D004, customer number	identifies			
C004				p004a	D002, car	has			S8 ...
				p004b	D005, rent-rates				
C005				p005a	D001, customer	has			...
				p005b	D006, name				

Figure 3: part of a connection-type glossary

3. An overview of the mapping process

For the derivation of a conceptual scheme from glossary entries we propose an iterative process consisting of several phases and steps (see figure 4). It starts with a check of the glossaries w.r.t. completeness and inconsistencies (for details see section 4). If the entries are consistent and if both parties (designer and end-user) agree that the listed entries reflect all relevant UoD concepts and their interrelationships, then a stepwise mapping to the conceptual scheme is initiated. In the first step of this 2nd phase, mapping rules are applied which help to associate concepts of the target model to the glossary entries. These rules will be discussed in section 3.2.

In the case of UML, e.g., thing-types are mapped to classes or (attribute) value-types², connections to associations or attributes or generalizations or aggregations. The next

² 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*.

steps of the mapping phase aim at a refinement of the target model by restructuring and completion.

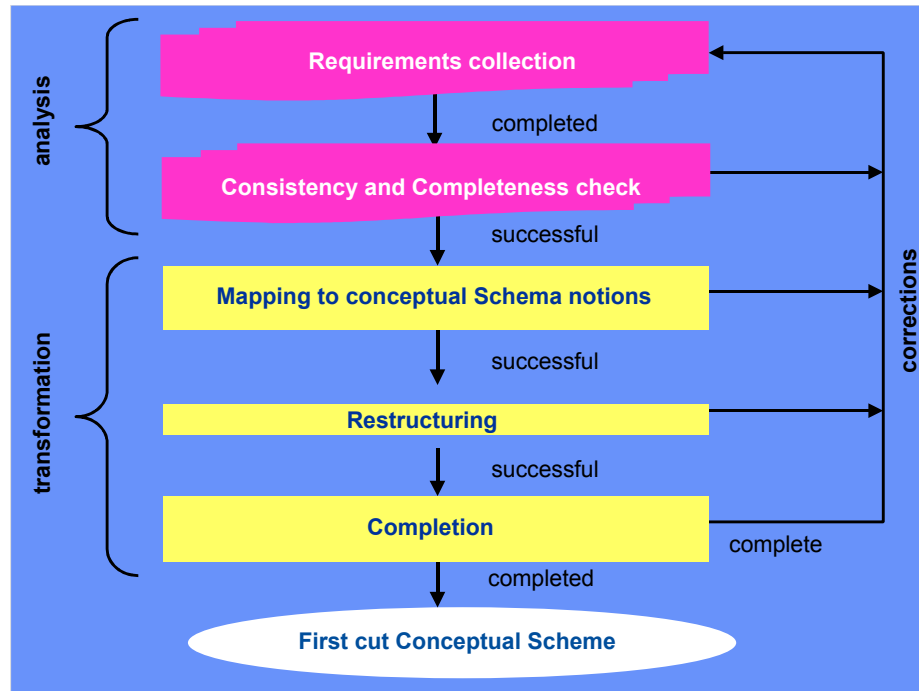


Figure 4: the mapping process

Clearly, using this approach, the mapping will not always result in a complete and fully specified conceptual schema. In the case of UML, e.g., we can generate classes, attributes and value-types, roles, associations, association classes, multiplicities and assertions, which are the most common concepts within a wide range of conceptual models. However, since KCPM is designed as an interface to the end-user, it does not provide for concepts which are of interest for implementation purposes but not from the user's point of view. Consequently, the mapping result is a first cut schema which the designer has to complete according to his better understanding of the UoD and according to some implementation decisions.

3.1. Consistency check

This step can be formalized to a large extent and thus again be deferred to a computerized tool. We use the term 'consistency' with a rather general meaning, covering 'malformedness' and 'redundancy'.

Uncritical redundancy

If two or more connection-types share the same involved thing-types, then some of these connection-types may be redundant. This is not true for all cases but depends on the semantics of the connection-type [17]. Consider for instance the connection-types *employee_works_at_project* and *employee_manages_project*. Although connecting the same thing-types, these two connections have different meanings. Thus, it is the designer's task to decide on what to do (consider as synonyms and thus redundant or not).

Critical redundancy:

Synonym concepts should be prevented from being mapped to the conceptual level except from one of them (the 'representing concept').

Uncritical malformed entries:

Orphan thing-types should be avoided if possible. If orphan thing-types exist, at least a warning should be given, but the designer need not correct this.

Critical malformed entries:

- Cycles of special connection-types (e.g. aggregation or generalization).
- A thing-type should not be an involved thing-type and connection-type determiner of the same connection-type.
- A thing-type should not be an involved thing-type and perspective determiner in the same connection-type.

3.2. Mapping rules

The mapping of KCPM notions to conceptual ones 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. A *proposal* means, that the proposed mapping is more likely than another one. Thus, the designer may accept the proposal or take another decision.

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 the 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 rules 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 an 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.

Based on these rules it is rather easy to formulate similar rules for the mapping of connection-types. We will not present them in detail here, but hint at the fact that the mapping of generalization and aggregation connection-types is trivial, and that all other rules (mapping to associations or attributes) are indirect in the sense that the targets depend immediately on what has happened to their involved thing-types.

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 $T_1 \dots T_n$ is mapped to an Association if at least two of the involved thing-types have been mapped to classes previously

Figure 5 shows some meta rules for the mapping process indicating 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

Figure 5: meta mapping rules

As can be seen in figure 5 a 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*).

3.3. Restructuring and completion

The restructuring and the completion step consist in basic consistency checks at the conceptual level like, e.g.

- check for orphan classes [18], i.e. classes which are not related to other classes,
- check for classes with identical attributes. If such classes exist one has to decide if the classes are synonymous or if they establish a generalization hierarchy.

As a consequence of such checks changes on the predesign level, i.e. within the glossaries, might become necessary (“restructuring”).

Another candidates for restructuring could be connection-type determiners, if the target conceptual model does not offer a concept like UML does with its *association class*. In another conceptual models equivalent structures have to be found.

We also have to be aware of tool restrictions. E.g. the rational rose tool (Release 2001.03.00) is not able to model n-ary associations, although it is a feature of the UML specification draft 1.3 [19]. Within the restructuring step therefore we have to handle these n-ary associations and create a work-around. The best way is, to transform a n-ary association into an association class, although the semantic of the two modeling concepts is different.

“Completion” means for example the adding of an identifying value-type to a class, if there is none and this is requested from an implementation point of view.

4. A simple example

We are now going to illustrate the previously outlined theory in the light of an example. 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. By using this example we will step through all phases of the transformation process given in section 3.

4.1. The starting point (Natural language source)

Let us suppose an UoD were we have to deal with publishing companies (Verlage). The description of structural aspects is as follows (in german):

*(1) Verlage verlegen Bücher. (2) Autoren schreiben Bücher. (3) Ein Buch kann von mehreren Autoren geschrieben werden. (4) Ein Autor hat einen eindeutigen Namen, ein Geburtsdatum und eine Adresse. (5) Ein Buch hat einen eindeutigen Titel und einen Preis. (6) Jedes Buch wird von genau einem Verlag verlegt. (7) Ein Verlag hat eine eindeutige Bezeichnung und eine Adresse. (8) Ein Verlag beschäftigt Mitarbeiter. (9) Ein Mitarbeiter hat eine eindeutige SVNr., einen Namen und ein Geburtsdatum.*³

There are two ways for processing that kind of sentences: Firstly, KCPM modeling concepts could be assigned manually and transformed into glossary entries. Secondly, using

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

a natural language parser and appropriate transformation rules, the semantics extraction could be done to a large extent automatically⁴.

4.2. Automatic collection of KCPM entries and consistency check

For this purpose we use a parser based on Natural Theoretic Morpho-Syntax [5,20]. This parser classifies words into word categories and phrases into phrase categories (e.g. noun = n0, nounphrase = n3). Based on these outputs we are able to interpret the sentences in the sense of associating KCPM modeling concepts (see figure 5).

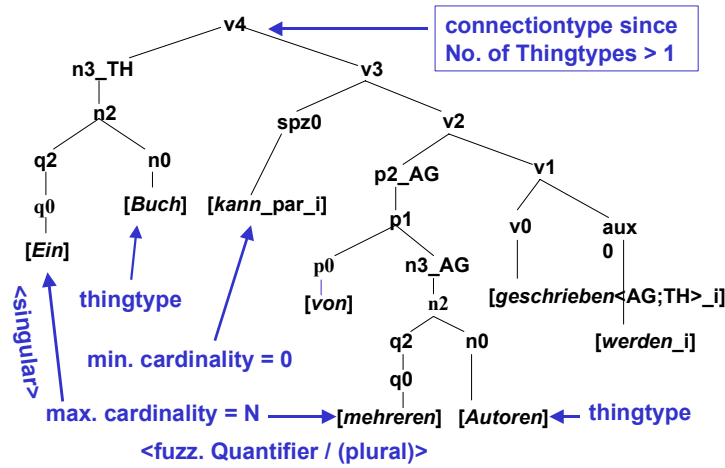


Figure 5: Syntax Tree for sentence (3) of the example

According to the parser output, nouns will be modeled by thing-types. Quantifiers (q0) and some auxiliaries (*kann*) give hints for the assignment of minimal and maximal cardinalities. If more than one thing-types are detected within a sentence then a connection-type may be detected from this sentence. Note also, that nouns like “Address” (Adresse) or “birthdate” (Geburtsdatum) will be seen at that stage as thing-types, too. Compared to other approaches (e.g.: [22]) this strategy saves rules for deciding which noun should mapped to a class and which should not. In our opinion, this decision is not simply a question of one sentence. Therefore, most of the rules given in other approaches do not work properly. E.g., based on one sentence it is not generally possible to assign to the Verb “haben” (have) a Class/Attribute relation as it works in our simple example.

⁴ This second approach is the aim of the project NIBA which we are running since several years in co-operations with the Institute of Computer Linguistics at the University of Klagenfurt. (NIBA is an acronym for the German project title ‘Natürlichsprachige Informationsbedarfsanalyse’, natural language based requirements analysis.) However a complete treatment of the linguistic details would mean going far beyond the scope of this paper. Therefore, only outline some general concepts in section 4.2. References to NIBA results are listed in the bibliography.

We also believe, that in the early phase of requirements collection it should be neither a question for the developer nor for the end-user to decide on such details. Instead both parties should concentrate on the question: Which noun is important and hence a relevant UoD notion and which noun is not. To conclude, this is why we prefer the more general notions of thing-type and connection-type instead of class, attribute, value-type (type expression) and association.

The KCPM notions derived from the sentence may be compared with already existing entries in the KCPM database (the glossaries). This consistency check helps to find malformed or redundant information (e.g. redundant connection-types – see section 3.1). The abstraction induced by KCPM enhances this step, since it allows for an easy detection of redundancies that would go beyond type borders in other approaches (i.e., redundancies between classes and attributes).

4.3. Mapping to UML

After having finished the consistency and completion phase (which may request iterative steps), the KCPM glossary entries are mapped to UML notions according to the rules given in section 3.2:

Assume that all entries (connection-types and thing-types) together generate a network of information as given in figure 6.

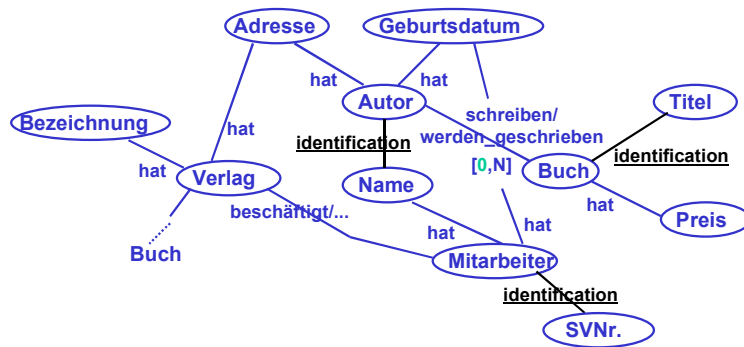


Figure 6: Graphical representation of KCPM glossary entries

Further assume that the phrases “*Autor hat einen eindeutigen Namen*”, “*Buch hat einen eindeutigen Titel*” and “*Mitarbeiter hat eine eindeutige SVNr.*” express that name (*Name*), title (*Titel*), social insurance no. (*SVNr.*) identify author (*Autor*), book (*Buch*), employee (*Mitarbeiter*), respectively. Thus they led to special semantic connection-types between those thing-types.

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 (*Name*), title (*Titel*), social insurance no. (*SVNr.*), author (*Autor*), book (*Buch*), employee (*Mitarbeiter*). We can also apply **rule 6** to book

(Buch). If we have context information and know that every string which ends up with “...Name”, “...Titel”, “...Bezeichnung”, “...SVNr.”, “...Preis”, “Datum” 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 *Buch*, *Mitarbeiter*, *Autor* and the value-types *Name*, *Titel*, *SVNr*, *Bezeichnung*, *Preis*. Now it is time to apply indirect rules. E.g., indirect rule 14 may be applied to map publishing comp. (*Verlag*) to a class, since it is connected to at least one value-type (see: *Bezeichnung*) and at least one class (see: *Buch*, resp. *Mitarbeiter*). No rule supports us to classify the thing-type address (*Adresse*). 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 7. 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.).

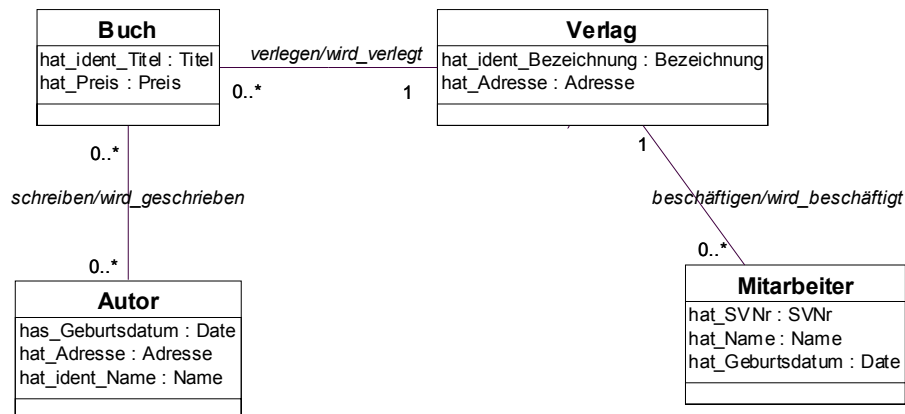


Figure 7: First cut UML-Schema

4.3. Restructuring, Completion, Example Conclusions

Once we have determined the UML schema, we have to perform consistency and quality checks on the conceptual schema level. For instance, we can ask why author (*Autor*) and employee (*Mitarbeiter*) share the same attributes and value-types. This can lead to a revised version of the above schema. In this version an abstract class person (*Person*) could introduced as a generalization of employee (*Mitarbeiter*) and (*Author*).

During the completion step, the schema may be extended by new entries of all kinds.

5. 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 [21] 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 6) 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.

6. 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 ([4], [5], [6], [7], [8]).

Within this paper we have focused on the mapping of the static part of an UoD model. Our general idea is twofold. First, a requirements analyst can collect UoD concepts without thinking in terms of classes, attributes, associations, value-types, because such terminology also depends on the conceptual model used and technical reasons (tool

support for a notion). Second, the transformation is done on different levels. Each level (steps) is responsible for a specific task. The consistency check is responsible for checks which are independent from the conceptual model. The mapping provides a first modeling language transformation (transformation of basic notions). The restructuring and the optional completion level (step) assumes that the target model is known and helps the designer to generate the target model. 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

- [1] Booch, G.: Object-Oriented Analysis and Design with Applications. Benjamin/Cummings Publ. Comp., 1994.
- [2] Ceri, S. (ed.): Methodology and Tools for Database Design. North Holland 1983.
- [3] Coad, P.; Yourdon, E.: Object-oriented Analysis, Prentice Hall, 1991.
- [4] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: Dinge und Zusammenhänge: NTS-gestützte Ableitung konzeptueller Vorentwurfsschemaeinträge aus natürlichsprachlichen Anforderungsdefinitionen. In: Erich Ortner, Bruno Schienmann, Helmut Thoma (eds): Natürlich-sprachlicher Entwurf von Informationssystemen - Grundlagen, Methoden, Werkzeuge, Anwendungen. Proc. GI-Workshop, Tutzing, 28.-30. Mai 1996.
- [5] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: „NTS-Based Derivation of KCPM Cardinalities: From Natural Language to Conceptual Predesign“. In: R.P. van de Riet, J.F.M. Burg, A.J. van der Vos (eds): Applications of Natural Language to Information Systems. Proc. Of the Second International Workshop June 26-28, 1996, Amsterdam.
- [6] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler, Ch.: „Das Projekt NIBA Zur automatischen Generierung von Vorentwurfsschemata für die Datenbankentwicklung“. Papiere zur Linguistik Nr. 55 (Heft 2, 1996) Gunter Narr Verlag Tübingen
- [7] 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 June, 1997, Vancouver, Canada
- [8] 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.
- [9] 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.
- [10] Kristen, G.: Object Orientation, the KISS Method - From Information Architecture to Information Systems. Addison Wesley, 1994.

- [11] Lochovsky, F. H. (ed.): Proceedings of the 8th Int. Conference on Entity Relationship Approach. North Holland Publ. Comp. Oct. 1989.
- [12] 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.
- [13] Nijssen, G.; Halpin, T.A.: Conceptual Scheme and Relational Database Design – A fact oriented approach. Prentice Hall Publ. Comp. 1989.
- [14] Rumbaugh, J.; Blaha, M.; Premelani, W.; Eddy, F.; Lorensen, W.: Object oriented modeling and design, Englewood Cliffs, NJ, Prentice Hall, 1991.
- [15] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language, Documentation Set 1.0, Rational Software Corporation, Santa Clara 1997.
- [16] Sommerville, I.: Software Engineering. Addison Wesley Publ. Comp., 3rd edition.
- [17] Storey V.C.: "Relational database design based on the entity relationship model". In: Data & Knowledge Engineering, Vol. 7, 1991, pp 47-83.
- [18] Tazovitch, B.: "An Expert System for Conceptual Data Modeling". In: [Loch89] pp. 205-220.
- [19] OMG-UML Specification Draft Version 1.3. Notation Guide, 1999
- [20] Fliedl, G. Mayerthaler, W. Winkler, Ch.: Lexikon der Natürlichkeitstheoretischen Syntax und Morphosyntax, Stauffenburgverlag, Tübingen, 1999.
- [21] Michael Stark Geschäftsprozessmodellierung im konzeptuellen Vorentwurf Master Thesis, Universität Klagenfurt.
- [22] 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.): 12th International Conference on Entity Relationship Approach. Arlington, Texas. New York, Springer 1993, pp 106-217.