Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 61 (2007) 433–448
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Deriving static and dynamic concepts from software requirements using sophisticated tagging ☆

Günther Fliedl [a,*], Christian Kop [a], Heinrich C. Mayr [a], Alexander Salbrechter [a], Jürgen Vöhringer [a], Georg Weber [a], Christian Winkler [b]

[a] *Institute of Business Informatics and Application Systems, University of Klagenfurt, Austria*
[b] *Institute of Linguistic and Computational Linguistic, University of Klagenfurt, Austria*

Received 20 June 2006; accepted 20 June 2006
Available online 18 July 2006

Abstract

Natural language requirements specifications form the basis for the subsequent phase of the information system development process, namely the development of conceptual schemata. Both, the textual as well as the conceptual representations are not really appropriate for being thoroughly captured and validated by the 'requirement holders', i.e. the end users. Therefore, in our approach the textual specifications are firstly linguistically analyzed and translated into a so-called conceptual predesign schema. That schema is formulated using an interlingua which is based on a lean semantic model, thus allowing users to participate more efficiently in the design and validation process. After validation, the predesign schema is mapped to a conceptual representation (e.g. UML). The sequence of these translation and transformation steps is described by the ''NIBA workflow''. This paper focuses on the information supporting a step by step mapping of natural language requirements specifications to a conceptual model, and on how that information is gained. On particular, we present a four-level interpretation of tagging-output.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Conceptual modeling; Natural language processing; Requirements engineering; Tagging; Shallow parsing

1. Introduction

A crucial factor for the success of any information system development project is the comprehensive and consistent engineering of the future users' requirements. Deficiencies in requirements elicitation and analysis are costly to repair in later development states. Natural language requirements specifications often form the basis for the next phase, namely the development of conceptual schemata. Much effort has been spent so far in developing concepts, methods and tools for requirements engineering. It is quite natural, that amongst those

☆ This work was partly funded by the Klaus Tschira Stiftung Heidelberg. The aim of the project NIBA is natural language processing to support requirements engineering and conceptual modeling.
* Corresponding author.
E-mail address: guenther.fliedl@uni-klu.ac.at (G. Fliedl).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.06.012

there are also natural language processing (NLP) based approaches (see e.g. [2,3,14,19]). Typically, these approaches focus on transforming natural language requirements specifications into conceptual schemata and thus predetermining the target conceptual model. Sometimes however a normative specification language is used [15]. Usually in requirements elicitation end-users have to communicate with the designers using either natural language or conceptual schemata in order to negotiate and validate the requirements specifications. Both types of communication have advantages but also restrictions. End users are not always able to understand conceptual schemata, but on the other hand, natural language specifications are often ambiguous.

Since neither textual nor conceptual representations are really appropriate for being thoroughly understood and validated by the 'requirement holders', the NIBA approach introduces a more appropriate interlingua schema. Like the other approaches it is based on linguistic analysis. The textual requirements specifications are however transformed into a so-called conceptual predesign schema, which is formulated using an end user oriented model. This allows the user to participate in the design and validation process more efficiently [11,13]. After the end user validation, the predesign schema is mapped to a conceptual design schema using the model (language) of the developer's choice.

The requirements specifications, which are the starting point for the NIBA approach, depend strongly on flexible sentence constructs. We therefore had to find a solution for tackling such kind of sentence constructs. Semantic tagging, e.g. labeling words by semantic classifiers (sem-tags like "tvag2", see [5]) seems to be a promising approach for extracting general patterns from conditional sentence constructs which, in a subsequent step, can be interpreted and mapped to a conceptual model pattern. Classical tagging approaches use standardized (POS) tag sets. However, this kind of standardized tagging, which is performed for instance by Treetagger, Brilltagger, Q-tag, etc. (see [1,16,20]), seems to have the following weaknesses:

– the tags provide merely lexical-categorial information,
– ambiguity can be made explicit only in a restricted sense,
– only limited chunking (normally chunking of simple NPs) is possible.

Furthermore, POS-tagging currently focuses on the English language [12,20] in the first place, and developing taggers for foreign languages poses a lot of challenges. With respect to the German language, its morphological richness and its (relatively) free word order are a source of problems. The complexity of words, the ambiguity of word endings and the big amount of serialization variants rarely allow for a straight forward interpretation of the input material.

To overcome these linguistic problems a tagging and interpretation approach has been elaborated [5] and implemented within the NIBA framework. In this paper, we introduce a new key approach, namely a four-level interpretation of tagging-output stored in XML-files. For the purpose of pragmatic interpretation we additionally assume that in the context of typical scenario texts even simple sentences are linguistic encodings of if–then relations (where the if-part is empty). The tools used for tagging and interpretation as well as their underlying strategies are presented within this paper. In Section 2 the underlying basic linguistic theory is outlined and some notions of the modeling language are described, which we use as an interlingua between natural language and common conceptual modeling languages (like, e.g. UML [17]). Section 3 introduces the extended tagging approach and its implementation in NIBA-TAG. Section 4 discusses the interpretation strategies of the tagging outputs to derive dynamic concepts as well as static concepts. In Section 5 we demonstrate how the tagging and interpretation components may be further used in the conceptual predesign step. After that the paper gives an overview of the mapping strategies to transform conceptual predesign schemata to conceptual schemata. The paper ends with an overview about further research and development.

2. Underlying theories

2.1. The basic linguistic theory

As has been described in a series of precedent publications (see, e.g. [5]), NTMS (**N**atürlichkeits-**T**heoretische **M**orpho**S**yntax) proved to be a flexible fundament for the linguistic subtasks of the NIBA project. NTMS is a grammar model based on Generative Syntax in the Chomsky style [4]. Sentence related phenomena

are represented by trees expressing constituency and dependency. These trees are projections of lexical base-categories. Each tree has just one lexical head and, accordingly, just one dominant heritage line. Given our framework, we tag the root elements with NTMS-defined morpho-syntactic and (domain-) semantic features for categories, types and subtypes of lexical base elements. Most of the used features have a pure semantic function (e.g. [countable], [animate], etc.).

Our focus of linguistic investigation lies on deep analysis of the morpho-syntactic relevance of verbal features like [''tvag2''] which decodes bivalency of the verbal head and agentivity of the involved subject. Features like these are related to specific configurations of Theta-roles, which can be presented as Predicate-Argument-Structures (PAS). Verbclass-specific PAS are core components of the typical NTMS lexicon entries for verbs (up to now we classified about 16.000 German verb entries). Lexicon elements bear a code valence and the argument position of the indicated theta-roles. Thematic role configurations can be derived from the sem-tags [6].

2.2. The conceptual predesign model (KCPM) – an interlingua for model mapping

Within the context of mapping requirements specifications to conceptual models an interlingua approach, which applies a lean semantic model as an interface between natural languages texts and conventional conceptual models (like the ER-model, UML), turned out be advantageous for both: the business owners (end users) being enabled to validate requirements models on a transparent and tolerable level of abstraction, and the mapping process itself the complexity of which may be reduced substantially by the use of an interlingua. The model we propose is called KCPM (Klagenfurt Conceptual Predesign model) and described in detail in e.g. [9].

KCPM consists of a small set of modeling notions for static (structural) and dynamic UoD aspects. The latter also cover the needs of business process modeling. The main notions are **thing-type**, **connection-types**, **cooperation-type**, **operation-type**, **pre-** and **post-condition**. Thing-types represent important domain concepts. A thing-type is a generalization of the classical conceptual notions class and value type, so that, e.g. both *customer* and *customer name* are modeled as (different) thing-types. Typical things (instances of thing-types) are natural or juristic persons, material or immaterial objects, abstract notions. In textual requirements specifications they usually are referred to by noun phrases. Connection-types represent relationships between thing-types. Verb phrase and prepositional phrase usually give hints for connection-types. Operation-types are used to model functional services that can be called via messages (service calls [8]) between objects that are seen as systems. As such they may be perceived of as a generalization of the notions use-case, activity, action, method, service etc. Each operation-type is characterized by references to so-called thing-types which model the **actors** (acting and calling actors of the respective operation-type) and service parameters.

An *acting actor* is the thing-type which a given operation-type has been assigned to as a service. It is capable of carrying out instances of the operation-type. A **caller** is a thing-type that can initiate the execution of an operation-type instance by calling up the corresponding service of the respective acting actor. In the case of business process modeling a *customer* can initiate a service of an enterprise (e.g. *to answer to a letter of complaint*). In that case on the top level the *enterprise* is the acting actor (the thing-type which is responsible for executing the operation-type *answer to a letter of complaint*). The *letter of complaint* and *the answer* are the incoming and outgoing service parameters.

To sum up, UoD dynamics emerge from (acting) actors performing operations invoked by (calling) actors under certain circumstances (**pre-conditions**) and thus creating new circumstances (**post-conditions**). This is captured by the KCPM concept of **cooperation-type**, a term we adopted from object oriented business process modeling [10]. A particular cooperation in that sense is an elementary step of a business process to which one or more actors contribute by (concurrently) executing operations (services). Thus for our complaint letter example a cooperation-type could have the following pre-condition, operation-types and post-conditions:

- pre-condition: *letter of complaint about a product failure comes in*,
- involved operation-types with contributing (acting) actors: *secretary checks customer status, clerk searches for amount of loss caused by the product failure*,
- post-conditions: *costumer is important and amount of loss >10.000 Euro*.

3. Sophisticated tagging

Our tagging methodology is based on the key concepts of NTMS as described in Section 2.1, and includes [7,5]

- the assignment of POS and semantically motivated subclass tags,
- morphological parsing with suffix identification, stemming, lemmatizing and compound splitting,
- the chunking of phrases (different types of word grouping),
- relating of phrases to syntactic and semantic default-functions.

Consequently, much effort had to be spent into the definition of context rules that act as disambiguation triggers during the tagging process. Based here-on, a tagging tool called NIBA-TAG has been developed and integrated into the NIBA Toolset.

NTMS based Part-of-speech tags divide words into morpho-syntactically and semantically motivated categories, based on how they can be combined to form sentences. For example, some types of auxiliaries can combine with participles and not with infinitives; articles can combine with nouns, but not verbs. Part-of-speech tags also supply information about the semantic content of a word. For example, nouns typically express things, and prepositions express relationships between things. In addition, verbs encode different concepts of meaning. Sometimes they refer to features of an entity (in the case of ergative verbs), in other cases they describe an action carried out by the respective subject. The largest verb class is represented by bivalent relational agentive verbs.

A merely morphological classification of verb tags as is commonly practiced by most taggers proves unsuitable for conceptual orientation which is crucial in requirements engineering. Therefore, a sub-classification of verb tags from a semanto-logical point of view turns out to be indispensable. The example output below contains different sorts of parameter allocation and is derived from the following example sentence:

Ein einlangender Beschwerdebrief wird an die Beschwerdeabteilung weitergeleitet. (*An incoming letter of complaint is forwarded to the complaints department*)

```
<n3 position="0">
  <q0 number="1" numeral="Ein" position="0" type="numeral" id="0" lowerCase="ein">Ein
  </q0>
  <a0 position="1" id="1" lowerCase="einlangender">einlangender
  </a0>
  <n0 position="2" id="2" lowerCase="beschwerdebrief"
  lastcomponent="1">Beschwerdebrief
  </n0>
</n3>

<v0 referTo="werden" mode="pass" form="ind" position="1"
  verbclass-number="1" tense="present" person="3"
  lowerCase="wird" id="3" verbclass="AUX">wird
</v0>
....
<v0 referTo="weiterleiten" form="partizip" position="3" lastposition="1"
  verbclass-number="7" lastcomponent="1" tense="perfect" partikel="weiter"
  lowerCase="weitergeleitet" id="7" verbclass="tVag/2">weitergeleitet
</v0>
```

In this XML-output the input words are related to wordclass-specific features like Verbclass = AUX, person = 3 or mode = pass(iv) for the auxiliary element *wird*; verbclass = "tvag2", partikel = "weiter", tense = "perfect" for *weitergeleitet*, a German participle of the particle verb *weiterleiten*.

Consider now a subsequent example sequence:

Wenn ein Beschwerdebrief in der Beschwerdeabteilung eintrifft, dann wird dem Beschwerdebrief eine Prioritätsstufe zugewiesen und einem Sachbearbeiter zugewiesen (*if a letter of complaint comes in, then the letter of complaint gets a priority level at it is assigned to a clerk*):

```
...
<v0 referTo="zuweisen" form="partizip" position="9" verbclass-number="8" tense="perfect"
partikel="zu" lowerCase="zugewiesen" id="14" verbclass="tV/3">zugewiesen</v0>
...
<v0 referTo="geben" form="inf" position="14" lastposition="1" verbclass-number="8"
lastcomponent="1" tense="present" person="3" lowerCase="gegeben" id="20"
verbclass="tV/3">gegeben</v0>
</sentence>
...
```

The verbs *zugewiesen* and *gegeben* are participial forms of the trivalent verbal infinitives *zuweisen*, *geben*. They both decode treefold relations between an ACTOR (classified as AG), a THEME (a process involved non agentive entity classified as TH) and a GOAL (GO). These three involved roles (slot filling general semantic components of an action) are not necessarily visible, but in any case relevant for further interpretation.

4. Analysis of the tagging output

4.1. Deriving dynamic concepts

NIBA-TAG outputs are processed by the NIBA Dynamics interpreter which derives operation-types, conditions and cooperation-types. Fig. 1 shows the main window of the interpreter with its four different views.

The left upper part of the window lists all the sentences of a text, the right hand upper part presents a structural template for each of these sentences based on the assumption that each sentence can be defined as consisting of a condition part and its implicational consequence. If the condition part is not stated explicitly then

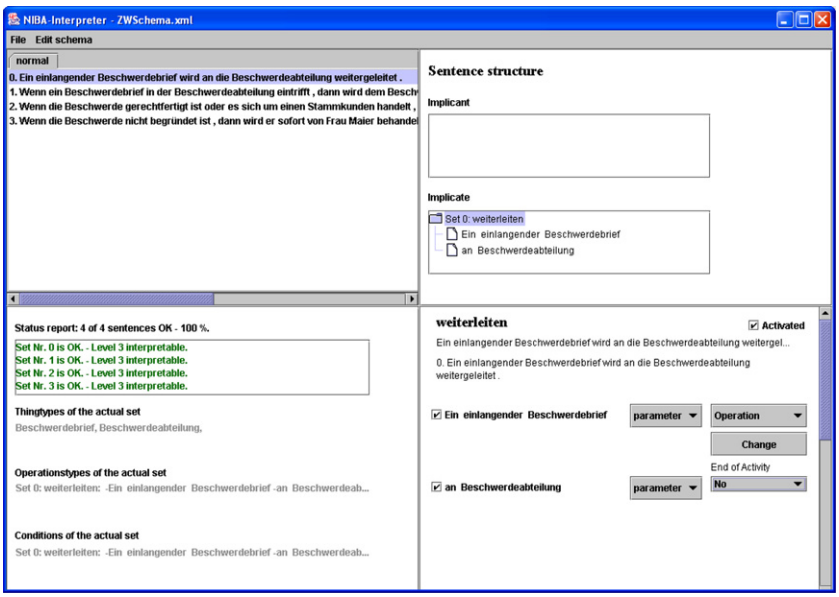


Fig. 1. Main window of the interpreter.

it is empty. In the condition part as well as in the implication part the verbs are collected together with those noun and prepositional phrases that could be candidates for verb arguments of the resp. verb. The identification of the verb arguments by the interpreter is based on the resp. NTMS verb class.

The interpreter assigns syntactic functions and semantic roles to those word groups which are identified as syntactic phrases by NIBA-TAG (see Section 3). This is a quite difficult task for German sentences because of the mentioned (morpho-) syntactic problems in German (free word order etc.; see Section 3). The interpreter makes linguistically motivated default decisions. The assignments of the syntactic functions and semantic roles are used for the interpretation process. The results are shown in the right upper and lower part of the window. The interpretation process presupposes decisions about the interpretability of the tagged sentences. Currently a first and simple model of four levels (level 0–level 3) was introduced:

Level 0: No interpretation of the sentence is possible at all. This means that the sentence is written in such a way that the interpreter cannot find any structure within the sentence which can be used for interpretation (see level 1).
Level 1: At least one of the two parts *implication* and/or *condition* is found.
Level 2: Verbs are found, but cannot be associated with arguments.
Level 3: Candidates for verb arguments are recognized for the verb (if there is only one in the sentence), or for at least one verb (if there are more of them in the sentence).

If a sentence can be interpreted on level 3 then it is possible to derive KCPM notions. This can be controlled in the right lower corner of the window. For each verb and its arguments in the implication section (upper right part) the end user receives a default interpretation. If the verb is an agentive verb then it is mapped to an operation-type. The noun which is the syntactic subject of the sentence is mapped to the acting actor. The nouns which could be the syntactic objects are mapped to parameters. If the verb is not an agentive verb then it is mapped to a condition. All the arguments of the verbs in the sentence are listed as candidates for involved thing-types. If there is more than one, the user has to select one of them to be the involved thing-type. For example, if the sentence *a person owns a car* is taken as a condition, then both *person* and *car* are candidates for the involved thing-type of that condition but only one of these nouns can be chosen as an involved thing-type (e.g. person). The rest of the sentence *owns a car* is then treated as the property of that thing-type.

All interpreter results are understood to be default, i.e. the user can always overrule default decisions. For example, the user can change the kind of each thing-type (parameter, calling actor, acting actor) if he thinks that the default assumption does not fit well.

Furthermore, currently, the tool distinguishes between sentences (sentence parts) that are useful for interpretation and sentences which are not. In fact, the tool assumes that every sentence should be interpreted. However, there is a check box "Activated" which is "on" by default. If the user disables this check box, then the interpretation result will not be transferred into in the final schema.

In some cases (where the sentence fits with some given implicational sentence patterns e.g. if/then constructs) the tool can also derive cooperation-types. The user then has the possibility to relate conditions and operation-types to logical operators (or, and, xor). Where possible, the tool gives hints about which of these operators should be chosen, otherwise, the user has to do this manually.

4.2. Deriving static concepts

Beside the interpretation of dynamic concepts a tool for simple extraction of static concepts was developed. In this section it will be shown that sophisticated tagging extensively supports static interpretation.

The chunking mechanism of the tagger-tool and the NTMS verb classification support the extraction of thing-types and connection-types. Interpretation is done in the following way:

Thing-types are detected by looking at noun phrases. These phrases have <n3> tags generated after chunking.

Verb classification can handle the problem of finding the number of arguments of verbs. Thus it can be concluded that the number of verbal arguments are converted to thing-types involved in the same connec-

tion-type. This approach is used in sentences with one main verb. In this case, the interpreter firstly searches for the main verb. If a ternary verb is filtered out, the interpreter tries to looks for up to three argument phrases in the sentence. However, if in the sentence containing a ternary verb only two arguments will be found, then the interpreter generates two thing-types and establishes binary connection-type between these two arguments. The reason is simple; in most cases the second (indirect) object of a ternary verb is often deleted. See for example: *X sends the letter*. The verb *to send* is ternary since it needs a recipient. Nevertheless in some cases the recipient is not necessary and thus it is not named in the sentence explicitly.

If a sentence has more than one main verb – e.g. it sentence contains a main clause and a subordinated clause then the same strategy as in the dynamic interpreter is used. It must find the borders of a main respectively a subordinated clause and examine each sentence part within this border. In particular the interpreter must

1. Find tags like <conn0> which is the tag for words like *wenn* (*if*) or *dann* (*then*). Of course in this cases it is also necessary examine if the word is an *if* or a *then*.
2. Split the sentence into parts within these borders.
3. If each sentence part has one verb then the interpreter can determine the connection-types according to the arity of the verb as described above.

However, for static interpretation a simpler approach can be used. It is based on the chunking mechanism of the tagger. Therefore, only the third level of the XML-tree is important and thus examined.

```
<text>
  <sentence>
    <n3>
      <det> ... </det>
      <n0> ... </n0>
    </n3>
    <p2>
      <p0> ... </p0>
      <n0>...</n0>
    </p2>
    <v0>
      ...
    </v0>
  ...
  </sentence>
  <sentence>
    ...
  </sentence>
</text>
```

If the tags are written as a string, ignoring the closing tag </..>, then the result is <**n3**><**p2**><**v0**>... In this case the interpreter can search for some simple combinations. Connection-types can be extracted from a combination of a noun phrase with a prepositional phrase e.g. ... *the man with the hat* ... If a classical tagging approach is used, the relationship between *man* and *hat* is given only implicitly and the interpreter has to analyze the words between the two related concepts. The NTMS notation is encoded by the following tags <det><n0><p0><de t><n0>. However, using chunking it is possible that these related concepts will become neighbors <n3><p2>. Thus searching for such a pattern <n3><p2>, <p2><n3> or <n3><n3> within a tag string can be a first hint for connection-types. It must be said that not all nouns inside <n3><p2>-neighbors automatically belong to the same connection-types since the prepositional phrase might belong to the verb and not the previous noun phrase, however, it is a good first hint e.g. compare the following German sentence: *Die Person beobachtet den Mann mit der Sonnenbrille* (*The person watches the man with the sunglasses*).

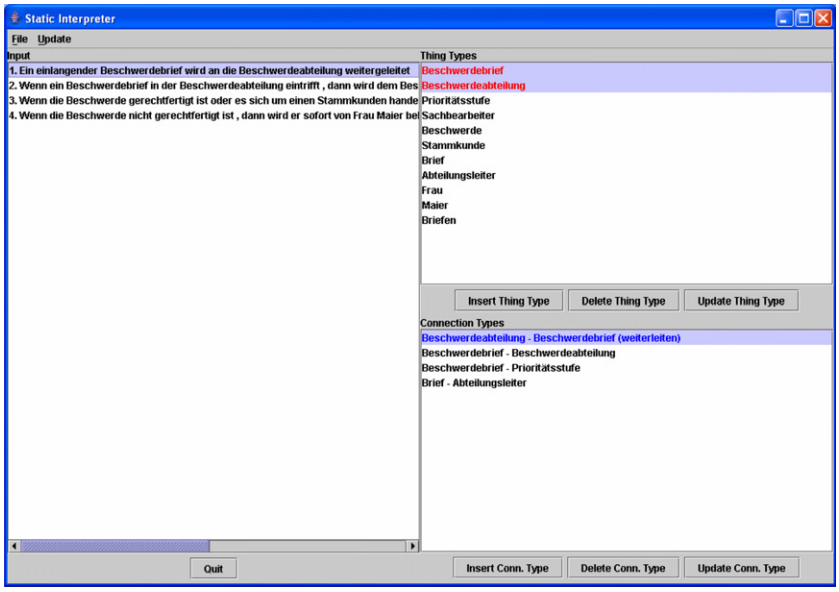


Fig. 2. Static interpreter window.

Furthermore, some sequence rules relevant for German, can be used for determining possible combination of argument phrase and bivalent verbs. In a subordinated clause, the main verb is always on the last position of the clause. Thus all the involved arguments are positioned before the verb. This means that for bivalent verbs in subordinated clauses the following patterns can occur:

- <n3><n3><v0>: This can happen in a subordinated clause with a direct object and subject in front of a bivalent verb.
- <n3><p2><v0>: This can happen in a subordinated clause if an object denoted by a preposition and the subject is before the bivalent verb.
- <p2><p3><v0><v0>: This is a special case where the second v0 is an auxiliary and the first v0 is the main verb.

If a sentence in German starts with a subordinated clause, then this clause replaces the subject from it's first position and the subject can be found after the verb. Thus patterns like <v0><n3><n3>, <v0><n3><p2>, etc. will give further hints for connection-types.

In all these cases the verb establishes a connection-type between the thing-types found inside the <n3> or <p2>. Since the above mentions combinations <n3><n3>, <p2><n3> and <n3><p2> for prepositions also appear in patterns with a verb, these can be also interpreted as connection-type patterns derived from a verb.

The main idea of the interpreter tool is to offer a list of thing-types and a list of connection-types to the user. The window structure enables the user to edit connection-types and thing-types (see Fig. 2).

5. Supporting the conceptual predesign

Tagging and *Interpretation* as described in the previous sections are embedded as the first two steps (components) into a workflow supporting the derivation of KCPM entries and, in a subsequent step, supporting the mapping of these entries to a conceptual model (activity diagrams, state charts and class diagrams). This section focuses on two tools that support the user in manipulating the concepts of the conceptual predesign level. The tools reflect the fact that there are always users preferring graphical representations and others preferring a tabular notation. In particular,

– the graphical KCPM schema editor was developed for supporting the drawing of dynamic concepts,
– the web-based KCPM glossary editor manages the links between the schema elements (e.g. specific thing-types, connection-types cooperation-types, operation-types and conditions) and the corresponding sentences in the requirements documents.

Independent of the form, both views are used for schema verification and modifications by the end users and their consultants.

5.1. The graphical editor

Following the interpretation phase, the mapping result may be viewed and refined graphically (and thus, implicitly, the initial requirements specifications). To keep this step as close as possible to the initial texts, the operation-types are named by the corresponding text phrases which have to be replaced by more generalized terms in addition to that. The left hand side of the corresponding window (see Fig. 3) features a tree view such supporting the presentation of several levels of abstraction with operation-types at the top level. These operation-types may represent services of high level thing-types thus having an associated cooperation-type schema (which in turn describes the behavior of the corresponding subsystem). Since each cooperation-type again may consist of one or several operation-types, a hierarchical navigation and presentation is supported.

The right-hand side of the editor window features a graphical representation of the corresponding KCPM schema entries. In particular, the user may edit the cooperation-type schema of each operation-type, e.g. by relating them with pre- and post-conditions (circle). Fig. 3 shows those cooperation-types which were already completed manually by interweaving the cooperation-types appropriately based on matching identical pre- and post-conditions.

5.2. Web-based KCPM glossary editor

Besides of the graphical view KCPM also was developed to support a glossary view for those kinds of users that are familiar with different types of glossaries, tables and forms. From our practical experience, however, we know that glossaries are scarcely suitable for dynamic modeling but they were widely accepted for static modeling and listings of operation-types (system services). Thus, both alternatives had to be supported by appropriate tools. This view has been implemented by means of a web interface. Fig. 4 shows the glossary of a couple of thing-types which are used in the example. The following properties are listed to the user from left to right:

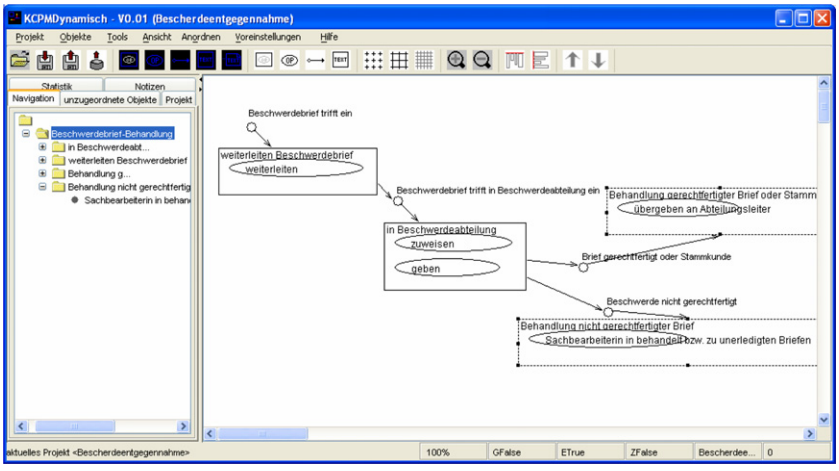


Fig. 3. Dynamic editor tool.

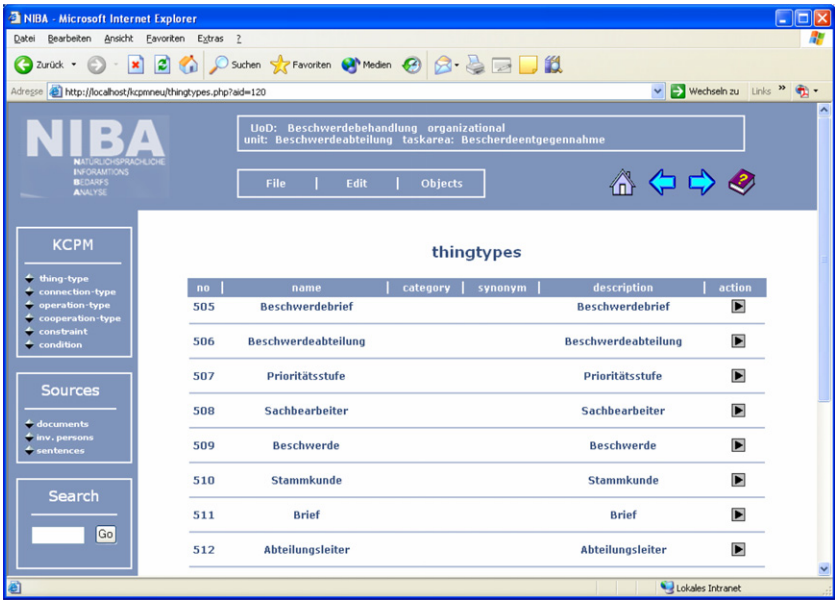


Fig. 4. Web-based KCPM glossary editor – thing-type glossary.

- Each thing-type has a system internal identifier (no.).
- For each thing-type the name of the thing-type is listed (e.g. the name for the thing-type with the no. 505 is *Beschwerdebrief* or *letter of complaint*). The name was derived from a specific noun in the sentences.
- Next there is a category where the user can make some assumptions about the usage of the thing-type in a conceptual schema (e.g. will it become a class or a value type).
- The property "synonym" next to category is a reference to another thing-type if the thing-type is the synonym of the thing-type referenced.
- Usually in the description property there is a short definition of the meaning and usage of the notion denoted by the thing-type. However, if the thing-type is automatically derived from a sentence, currently the name of the thing-type is stored in the description property as a place holder.

At the end of each line there is an "action"-column. If the user presses the arrow he can execute certain edit actions on the notion in that line (update and delete properties). If the designer has questions concerning this thing-type he can also store questions and related answers (see Fig. 5).

The web based glossary view also documents the relations between the KCPM schema entries and their requirements text sources which is important as a first step to guarantee the traceability between modeling notions and requirement sources.

Moreover, the requirements sources are presented separately (see section 'Sources' at the left-hand side of Fig. 4). Since the KCPM notions were described extensively in the previous sections we will not stress them here (section 'KCPM' at the left-hand side). Instead we will concentrate on the requirements sources and their manipulation.

Actually, we distinguish three different kinds of sources:

- Sentences,
- Documents and
- Involved persons (inv. Persons).

Sentence is the smallest unit of requirements source in our approach. It can be inserted manually but mainly it will be automatically generated (referenced) during the interpretation process (see Section 4).

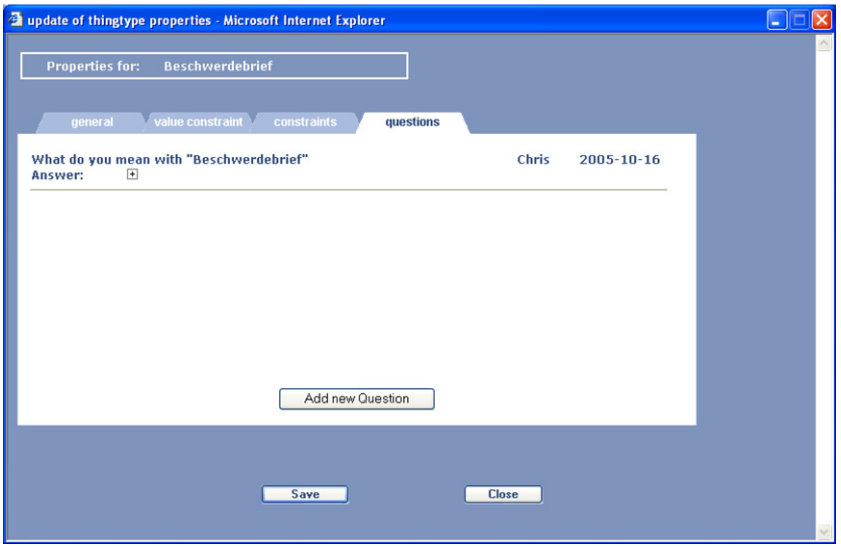


Fig. 5. Properties window for thing-types.

Whole requirements **documents** are referenced if there is no need to relate KCMP notions in a finer granularity. Actually there is no further restriction on what a document may be and how it should look like. Concerning the document format, it can be a Word textual file (e.g. a Word Document, plain text document, RTF or PDF document). The user also may store a graphical representation of that document (e.g. a screenshot), a multimedia file like a picture, a movie or an audio file. The content of a document may be structured or unstructured. This allows the user to relate KCPM notions to any kind of document. Relating a notion in this way means that the notion appears in that document explicitly or implicitly.

The source **involved person** is used if the user directly expresses his requirements in KCPM terms. This could be the case if e.g. the end user checks the forms and glossaries and notices that something is missing. Instead of writing again textual documents, the designer is able to reference this person directly.

6. Mapping to the conceptual level

So far we have described the steps of natural language analysis including its interpretation (Sections 3 and 4) as well as the support on the conceptual predesign level. In order to build an information system it is of course also necessary to transform the results to a conceptual design schema (e.g. UML). This step was already extensively described in previous work (see [9,13,18]). Therefore, in the following, we only give a short overview of the mapping process.

Depending on the target model, dynamic concepts are mapped in specific ways. If the target model is an activity diagram then the main mapping principles are the following:

- an operation-type will become an activity,
- a condition will become a transition between activities,
- if a cooperation-type includes more than one operation-type, the cooperation-type itself becomes an activity. All the operation-types inside the cooperation-type will become subactivities. Cooperation-types mean that the operation-types within can be executed independently from each other. Thus these mapped operation-types will also become independent subactivities by forking and joining (Fig. 6).

The transformation of the predesign schema into activity diagrams is done automatically by a certain component [18] of the graphical editor.

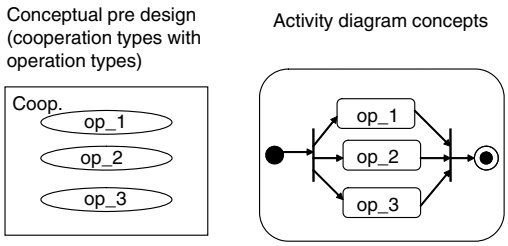


Fig. 6. Example of mapping parts of a dynamic KCPM schema to an activity diagram.

The mapping of predesign concepts to a state chart can be done in four steps. In a first step the user has to select a thing-type. This is the concept for which a state chart will be generated. Then all conditions where this thing-type is involved in are presented as state candidates (second step). The paths between these conditions are candidates for transitions between states (third step). In the last step all the derived concepts must be transformed to a state chart.

Mapping of static concepts (thing-types, connection-types) to a conceptual object model is based on the idea that much information about the target concept (class, value type) is available inside the predesign schema. If for example a connection-type is a generalization or aggregation then the thing-types that are involved in the connection will become classes. Furthermore, predesign schemata can be graphically represented similarly to networks. The leafs of these "networks" (i.e. the concepts that have only one neighbor) are candidates for value types.

7. Conclusion

The approach presented here uses combined tagging and interpretation for processing requirements texts. We call this a step by step generation of the predesign notions thing-type, operation-type, condition as well as cooperation-type based on linguistic patterns. The advantage of this method is a multilevel representation of the information which allows for user feedback on all stages of requirements text processing and a subsequent automatic predesign (KCPM) schema construction.

The KCPM methodology itself was already applied several times for practical (informal) requirements analyses. The most recent project was the creation of a requirements specification document in the domain of health care, namely the specifications for a cancer statistics database system. In this project, the users were medics, medical practitioners and hospital administrative personnel, who had no knowledge in conceptual design. Our KCPM approach and the resulting schemata proved to be particularly applicable for the communication of these kinds of users with the system designers.

Certain tools of the NIBA toolset (i.e. he static and dynamic editors) are currently evaluated by external project partners. Using the tools ourselves we already determined certain strengths and weaknesses, which will be addressed in future research. We also plan to do a more thorough statistical evaluation of our toolset and method by performing a statistical study. We want to deploy student comparison groups to work on a small-scale software project with and without our method.

Current research of our group includes the extension of our tagging- and interpretation-steps in order to allow the interpretation of English natural language texts. We also research the integration of KCPM models and ontology based software engineering. Research of combining the KCPM approach with existing MDA-methods and tools already yielded first results.

Acknowledgments

The authors also would like to thank all the students involved for their substantial implementation work.

Appendix

```
<?xml version="1.0" encoding="Windows-1250" ?>
- <text taggedBy="nibaTAG 1.11" position="0" id="0" lastcomponent="1">
- <sentence position="0" delimiter="." id="0">
- <n3 position="0">
<q0 number="1" numeral="Ein" position="0" type="numeral" id="0" lowerCase="ein">Ein</q0>
<a0 position="1" id="1" lowerCase="einlangender">einlangender</a0>
<n0 position="2" id="2" lowerCase="beschwerdebrief" lastcomponent="1">Beschwerdebrief</n0>
</n3>
<v0 referTo="werden" mode="pass" form="ind" position="1" verbclass-number="1" tense="present" person="3"
lowerCase="wird" id="3" verbclass="AUX">wird</v0>
- <p2 position="2">
<p0 position="0" id="4" lowerCase="an">an</p0>
<det0 form="" position="1" id="5" lowerCase="die">die</det0>
<n0 position="2" id="6" lowerCase="beschwerdeabteilung" lastcomponent="1">Beschwerdeabteilung</n0>
</p2>
<v0 referTo="weiterleiten" form="partizip" position="3" lastposition="1" verbclass-number="7" lastcomponent="1"
tense="perfect" partikel="weiter" lowerCase="weitergeleitet" id="7" verbclass="tVag/2">weitergeleitet</v0>
</sentence>
- <sentence aux-exists="1" position="1" delimiter="." id="1">
<conn0 position="0" id="0" lowerCase="wenn">Wenn</conn0>
- <n3 position="1">
<q0 number="1" numeral="ein" position="0" type="numeral" id="1" lowerCase="ein">ein</q0>
<n0 position="1" id="2" lowerCase="beschwerdebrief" lastcomponent="1">Beschwerdebrief</n0>
</n3>
- <p2 position="2">
<p0 position="0" id="3" lowerCase="in">in</p0>
<det0 form="" position="1" id="4" lowerCase="der">der</det0>
<n0 position="2" id="5" lowerCase="beschwerdeabteilung" lastcomponent="1">Beschwerdeabteilung</n0>
</p2>
<v0 referTo="eintreffen" form="ind" position="3" verbclass-number="2" tense="present" partikel="ein" person="3"
lowerCase="eintrifft" id="6" verbclass="eV">eintrifft</v0>
<conn0 position="4" type="conj" id="7" lowerCase=",">,</conn0>
<conn0 position="5" id="8" lowerCase="dann">dann</conn0>
<v0 referTo="werden" mode="pass" form="ind" position="6" verbclass-number="1" tense="present" person="3"
lowerCase="wird" id="9" verbclass="AUX">wird</v0>
- <n3 position="7">
<det0 form="" position="0" id="10" lowerCase="dem">dem</det0>
<n0 position="1" id="11" lowerCase="beschwerdebrief" lastcomponent="1">Beschwerdebrief</n0>
</n3>
- <n3 position="8">
<q0 number="1" numeral="eine" position="0" type="numeral" id="12" lowerCase="eine">eine</q0>
<n0 position="1" id="13" lowerCase="prioritätsstufe" lastcomponent="1">Prioritätsstufe</n0>
</n3>
<v0 referTo="zuweisen" form="partizip" position="9" verbclass-number="8" tense="perfect" partikel="zu"
lowerCase="zugewiesen" id="14" verbclass="tV/3">zugewiesen</v0>
<conn0 referTo="und" position="10" type="conj" id="15" lowerCase="und">und</conn0>
<pronoun form="" position="11" id="16" lowerCase="er">er</pronoun>
<v0 referTo="werden" mode="pass" form="ind" position="12" verbclass-number="1" tense="present" person="3"
lowerCase="wird" id="17" verbclass="AUX">wird</v0>
- <n3 position="13">
<q0 number="1" numeral="einem" position="0" type="numeral" id="18" lowerCase="einem">einem</q0>
<n0 referTo="Sachbearbeiter" form="sg" position="1" id="19" lowerCase="sachbearbeiter"
lastcomponent="1">Sachbearbeiter</n0>
</n3>
<v0 referTo="geben" form="inf" position="14" lastposition="1" verbclass-number="8" lastcomponent="1" tense="present"
person="3" lowerCase="gegeben" id="20" verbclass="tV/3">gegeben</v0>
</sentence>
+ <sentence aux-exists="1" position="2" delimiter="." id="2">
+ <sentence position="3" delimiter="." id="3" lastcomponent="1">
</text>
```

References

[1] E. Brill, A simple rule-based part of speech tagger, in: Proceedings of the Third Conference on Applied Natural Language Processing, ACL, 1992.
[2] E. Buchholz, H. Cyriaks, A. Düsterhöft, H. Mehlan, B. Thalheim, Applying a Natural Language Dialogue Tool for Designing Databases, in: Proc. Int. Workshop on Applications of Natural Language to Databases (NLDB'95), 1995.
[3] J.F.M. Burg, Linguistic Instruments in Requirements Engineering, IOS Press, Amsterdam u.a., 1997.
[4] G. Fliedl, Natürlichkeitstheoretische Morphosyntax, Aspekte der Theorie und Implementierung, Habilitationsschrift, Gunter Narr Verlag, Tübingen, 1999.
[5] G. Fliedl, Ch. Kop, W. Mayerthaler, H.C. Mayr, Ch. Winkler, G. Weber, A. Salbrechter, Semantic tagging and Chunk-parsing in dynamic modeling, in: F. Meziane, E. Métais (Eds.), Proceedings of the Ninth International Conference on Applications of Natural Language Processing and Information Systems, NLDB2004, Salford UK, Springer LNCS 3316, 2004, pp. 421–426.
[6] Ch. Fillmore, J.M. Petruck, J. Ruppenhofer, A. Wright, FrameNet in Action: the case f Attaching, International Journal of Lexicography (2003).
[7] G. Fliedl, G. Weber, Niba-Tag – A Tool for Analyzing and Preparing German Texts, in: A. Zanasi, C.A. Brebbia, N.F.F. Ebecken, P. Melli (Eds.), Management Information Systems, vol. 6, Data Mining 2002 Bologna: Wittpress September 2002, pp. 331–337.
[8] W. Hesse, H.C. Mayr, Highlights of the SAMMOA framework for object oriented application modelling, in: G. Quirchmayr, E. Schweighofer, J.M. Bench-Capon (Eds.), Proceedings of the Ninth International Conference of Database and Expert Systems Applications (DEXA'98), Lecture Notes in Computer Science, Springer Verlag, 1998, pp. 353–373.
[9] C. Kop, H.C. Mayr, An interlingua based approach to derive state charts form natural language requirements, in: M.H. Hamza (Ed.), Proceedings of the Seventh IASTED International Conference on Software Engineering and Applications, Acta Press, 2003, pp. 538–543.
[10] R. Kaschek, C. Kohl, H.C. Mayr, Cooperations – an abstraction concept suitable for business process reengineering, in: J. Györkös, M. Krisper, H.C. Mayr (Eds.), Conference Proceedings ReTIS'95, Re-Technologies for Information Systems, R. Oldenbourg Verlag, Wien, München, 1995, pp. 161–172.
[11] C. Kop, H.C. Mayr, Conceptual predesign – bridging the gap between requirements and conceptual design, in: Proceedings of the Third International Conference on Requirements Engineering ICRE'98, Colorado Springs, April, 1998.
[12] M. Marcus, B. Santorini, M. Marcienkiewicz, Building a large annotated corpus of English: the Penn Treebank, Computational Linguistics (1993).
[13] H.C. Mayr, Ch. Kop, A user centered approach to requirements modelling, in: Proc. Modellierung 2002, Lecture Notes in Informatics LNI p-12, GI-Edition, 2002, pp. 75–86.
[14] A. Moreno, N. Juristo, R.P. van de Riet, Formal justification in object-oriented modeling – A linguistic approach, Journal of Data and Knowledge Engineering 33 (2000) 25–47.
[15] E. Ortner, B. Schienmann, Normative language – approach a framework for understanding, in: B. Thalheim (Ed.), Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, 1996.
[16] H. Schmid, Probabilistic Part-of-Speech Tagging using Decision Trees, 1994. Available from: <http://www.ims.uni-stuttgart.de/ftp/pub/corpora/tree-tagger1.pdf>.
[17] St.R. Schach, An Introduction to Object-oriented Analysis and Design with UML and the Unified Process, McGraw Hill, Boston, Mass, 2004.
[18] A. Salbrechter, H.C. Mayr, Ch. Kop C, Mapping pre-designed business process models to UML, in: Proceedings of the Eighth IASTED International Conference on Software Engineering and Applications, ACTA Press, Cambrigde, USA, 2004, pp. 400–405.
[19] A.M. Tjoa, L. Berger, Transformation of requirements specifications expressed in natural language into an EER model, in: Proceeding of the 12th International Conference on ER-Approach, Airlington, Texas USA, 1993.
[20] D. Tufis, O. Mason, Tagging Romanian texts: a case study for QTAG, a language independent probabilistic tagger, in: Proceedings of the First International Conference on Language Resources and Evaluation (LREC), 1998, pp. 589–596.

Günther Fliedl is a Prof. in Computational Linguistics. He supervises courses on topics related to Natural Language Processing at the University of Klagenfurt. His research activities are focused on theoretical and implementational aspects of Computational Linguistics. He is engaged in the analysis of morphological, lexical, syntactic and semantic knowledge of natural language. The NLP-activities include *lexicon development*, *parser construction* and implementation of tagger functionality. As a member of the research group NIBA (''Natural Language based Requirements Analysis'') he was primarily concerned with the optimization of ***parsing methods***. He used methods based on a Chomsky oriented theoretical framework called NTMS (''Natural Theoretic Morphosyntax''). For being able to manage multifunctional ***tagging*** jobs like identification of unknown words and chunk parsing he was involved in the development of a tool called NIBA-TAG. This is some kind of multilevel natural language tagger for German doing normal POS-tagging, lemmatizing and morphological tagging.

Fliedl is member of program committees of international conferences like the NLDB.

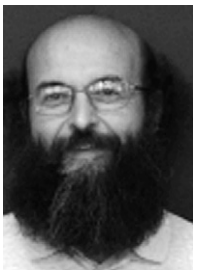
Christian Kop studied Applied Computer Science at the University of Klagenfurt. He works as a research assistant at the institute of business informatics and application systems at this University. In 2002 he received his Ph.D. His research interests includes information systems analysis and design, and questions of natural language processing support for information systems analysis. He is editor of a newsletter within a section of the German Informatics society.

Heinrich C. Mayr received his doctorate in applied mathematics from the University of Grenoble (France) in 1975. Between 1975 and 1983 he was an assistant professor at the University of Karlsruhe, Germany and a lecturer and visiting professor at several other German universities in the domain of database technology and information systems. From 1984 to 1990 he was a CEO of a German software company responsible for the section ''Business Information Systems''. Since 1990 he is a full professor of informatics at the University of Klagenfurt, Austria. His current research includes information systems design methodologies, natural language processing in requirements analysis, customer relationship management (CRM), knowledge management, software quality assurance and distance education. Amongst others he is vice president of CEPIS (the umbrella association of all European informatics organizations), past president of the GI, main publisher of the Lecture Notes in Informatics (LNI), head of the Industriestiftungsinstitut eBusiness, former dean of the Faculty of Economics, Business Administration and Informatics at the University of Klagenfurt and since April 2006 rector of the University of Klagenfurt.

Alexander Salbrechter received his MS in Applied Informatics in 2001 at the University of Klagenfurt. Afterwards he worked as a project assistant and university assistant at the department of business informatics and application systems. Currently he is engaged in designing and implementing modern bank specific application systems. His research interests include the design and implementation of systems with respect to modern pattern oriented OO-aspects and the analysis and design of business processes.

Jürgen Vöhringer received his degree in Applied Informatics at the University of Klagenfurt in 2003. Since 2004 he works as a project assistant at the institute for business informatics and application systems, where he participated in the implementation of the NIBA toolset and was involved in requirements engineering projects. Currently he focuses his research on schema integration on the pre-design level and on the use of ontologies for recognition and resolution of linguistic integration conflicts.

Georg Weber started his study of Applied Informatics at the University of Klagenfurt in 1994. Currently he is working on his master thesis which researches the linguistic analysis on requirements texts. Additionally he works as a senior engineer at Infineon Technologies. There he is involved in software development and system and availability monitoring. He is also a member of the NIBA team were he is responsible for the development of the German NIBA Tagger.



Christian Winkler studied Romance Languages, English, and General/Applied Linguistics at the Universität Klagenfurt. He has been teaching in many fields since 1994. Under the leadership of Willi Mayerthaler, he has worked together with long-time companion Günther Fliedl in several projects resulting in a long list of publications. Main interests: Comparative Linguistics (Phonetics, Morphology, Syntax, Semantics), Computational Linguistics, Rule Based (Natural) Language Acquisiton.