Tool Supported Extraction of Behavior Models[1]

Christian Kop[+], Jürgen Vöhringer[+], Martin Hölbling, Thomas Horn, Christian Irrasch, Heinrich C. Mayr[+]

+Institute of Business Informatics and Application Systems
Alpen-Adria Universität Klagenfurt
[chris | mayr]@ifit.uni-klu.ac.at

Abstract: Information system projects often suffer from incomplete or inadequate requirements specifications. In our opinion, which is supported by practical experience, these problems result from the fact that the models usually applied for requirements analysis are to abstract as to be easily understood and validated by the business owners, i.e. the end users. In addition to that, validation is often hampered by the fact that traditional modeling approaches do not relate the particular model elements to their corresponding requirements sources. We, therefore, propose an approach that uses a lean and thus more transparent requirements model, which is intermediate in the sense that it has to be mapped, after validation, to one of the traditional conceptual models. Clearly, introducing such an additional step into the information system development process induces increased effort which has to be reduced by appropriate tool support. This paper concentrates on specific aspects and tool support for extracting behavior models out of requirements texts.

1 Introduction

Usually, information system development projects start with textual requirements descriptions which then are manually transformed into a conceptual schema based on a particular conceptual model (e.g., using the UML). For analyzing behavioral aspects on the natural language level, so-called scenarios are very popular (see e.g. [Ro98b]). In practice, scenarios are used for different purposes (see, e.g. [We98]). [Le00], [Ro98a] and [Zh00] propose specific scenario construction processes to overcome problems in managing, refining and integrating textual scenarios.

Nevertheless, information system projects often suffer from incomplete or inadequate requirements specifications even if extensive conceptual modeling is carried out. Due to our experience, these problems often result from the fact that traditional conceptual models (and their instantiations in the form of conceptual schemes) are too abstract as to be easily understood and thus thoroughly validated by average users, who are the stake-holders of the relevant concepts, notions and requirements. This is especially true for non-technical environments (for a more detailed discussion see [MK02]). Additionally,

[1] 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.

the end user rarely is provided with references to the corresponding requirements sources of the schema elements, which turn out to be a further barrier for schema validation.

We, therefore, propose a lean modeling language that supports an end-user oriented view, and which is used as an inter-lingua enhancing automatic transformation from the textual sources and to the conceptual schema. We call that approach *Conceptual Predesign (CP)* since it is based on a semantic (meta)-model and since it precedes the usual conceptual design. The model itself (*KCPM: Klagenfurt Conceptual Predesign Model*) offers a set of semantic concepts for modeling static and dynamic aspects of a Universe of Discourse (UoD). It is based on an ontology 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). KCPM Schemes are represented in either glossary like or diagrammatic form.

Of course, such a schema development process has to be supported by tools to reduce the additional effort caused by introducing the interlingua. Therefore, an integrated tool chain has been developed within the last years. Currently, the following steps are supported by at least one tool or a tool component:

1. Sentence analysis: NIBA-TAG tagger [FW02] which is based on the NTMS approach [Fl99].
2. KCPM notion derivation: NIBA Interpreter Tool which derives interlingua concepts (thing-type, operation-type, conditions and cooperation-types) from the tagger output.
3. Schema validation and modification: KCPM Schema Editor, to be used by the end-users and their consultants. Currently, two different schema representations are implemented. One tool works with a graphical, the other one with glossary like schema representation. The latter also manages the links between the KCPM schema entries and the corresponding sentences in the requirements sources.
4. Schema mapping: KCPM-UML mapping component maps a pre-design schema into a conceptual one (here UML Class and Activity Diagrams).

The paper is organized as follows: Section 2 introduces the main KCPM concepts for behavior modeling. Based here-on, sections 3-5 describe those tool (components) that handle behavioral UoD aspects (Dynamic Interpreter, Graphic Editor and a web based KCPM Glossary Representation). Details on the underlying theory as well as on tool support for handling static UoD aspects may be found, e.g., in [Fl93, FW02, MK02, FK04, SM04].

2 KCPM

Comparing the different approaches to conceptual behavior modeling (e.g. *state charts*, *sequence diagrams*, *use cases*, *activity diagrams* etc.), a small set of basic principles may be identified:

- there are *actors* capable to execute *tasks/services*,

- task execution involves the manipulation of some objects (including message sending),
- the execution of a task depends on some pre-conditions,
- the execution of a task results in some post-conditions,
- thing-types can are related to pre- and post conditions. Each condition has a related thing-type.

In the context of requirements elicitation and analysis for business information systems it seems to be useful to concentrate on these principles. Therefore, KCPM is restricted to two main concepts for behavior modeling: **operation-type** and **cooperation-type**.

Operation-types are used to model functional services that can be called via messages (service calls). As such they may bee seen 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* (executing instances of the resp. operation-type), *service parameters* and the *receivers* of the service results. The notion of thing-type is the central KCPM concept for structure modeling [MK02] and may be seen as a generalization of the usual conceptual notions *class* (or entity set) and *value type*[2]. Typical things (instances of thing-types) are natural or juristic persons, material or immaterial objects, abstract notions. In textual requirements specifications they are usually referred to by noun phrases.

UoD dynamics emerge from *actors* (thing types) performing operations under certain circumstances (*pre-conditions*) and thus creating new circumstances (*post-conditions*). This is captured by the KCPM concept of **cooperation-type**[3] which is formally (simplified) defined as a triple *<prc,{(A,O)},poc> where prc* and *poc* are sets of conditions (possibly combined by logic expressions), and {(*A,O*)} is a set of actor, operation-type pairs. Figure 1 illustrates this definition: Cooperation-types are represented by rectangles, operation-types by ellipses. The latter are drawn into the rectangle of those cooperation-types they are contributing to. Conditions are represented by filled circles. c is a pre- (post-) condition of cooperation-type CT, if an arrow points from c to CT (CT to c). Each condition may be pre- and/or post-condition of one or more cooperation-types. In this way, a network of cooperation-types may be constructed. For each operation-type and condition we depict the 'involved' thing-types: parameters and actors<A>.

---

2 Value type corresponds to value set in the ER-model or to typed expression in UML. A related KCPM thing-type may be user defined (e.g. Color, Customername etc.) without restrictions (e.g. to simple types like Number, Date etc.).

3 This term has been taken from object orientated business process modeling [KK95]. A cooperation in that sense is an elementary step of a business process to which one or more actors contribute by (concurrently) executing operations (services).

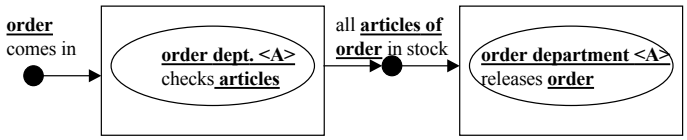

Figure 1: cooperation-types, operations-types, pre- and post conditions in a graphical notation

3 The Dynamic-Interpreter

Using the output of the tagger NIBA Tag, our Dynamic-Interpreter derives operation-types, conditions and cooperation-types based on the framework presented in [FK04]. It produces default interpretations which the user can overrule. Fig. 2 shows the main window of the interpreter with its four different views. The examples sentences are all in German since the computer supported sentence analysis currently only works for German sentences.

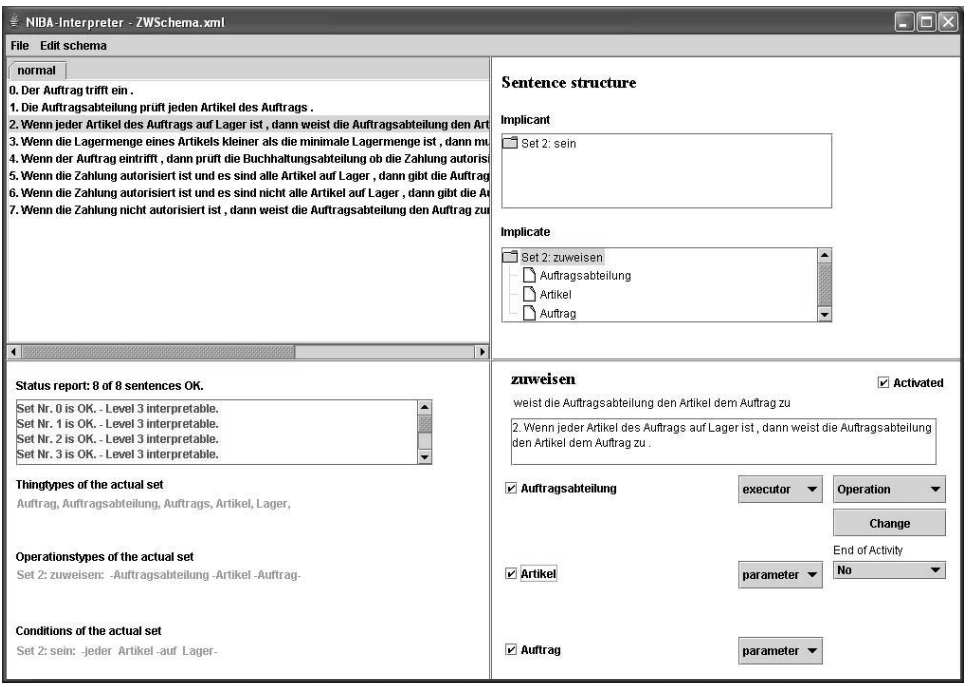

Figure 2: main window of the Dynamic Interpreter

In the left upper part all the sentences of a text are listed. On the right hand side (upper part) a basic structure is presented for each sentence. Each sentence can consist of a condition part and an implication part. If the condition part is not stated explicitly then it is empty. In the condition part as well as in the implication the verbs are collected

together with those noun phrases and prepositional phrases that could be the candidates for the verb arguments for each verb.
The left lower part of the window features some statistical data, among those the level to which an interpretation process has advanced. From that, the user can conclude the reasons in cases where an interpretation does not succeed. We currently distinguish a four such levels:

Level 0: No interpretation of the sentence is possible at all.
Level 1: The interpreter is able to find the condition part or at least the implication part of a sentence with condition.
Level 2: The interpreter is able to find verbs but cannot find their arguments.
Level 3: The interpreter is able to find candidates for verb arguments for at least one verb.

If a sentence can be interpreted on Level 3 then the derived KCPM notions are listed in the right lower corner of the window. If the verb is an agentive one then it is mapped to an operation-type. The noun which is a candidate for the syntactic subject of the sentence is mapped to the executing actor. The nouns which could be the objects are mapped to parameters.
If the verb is not an agentive one then it is mapped to a condition. KCPM allows relating conditions to a thing-type. E.g., if a phrase like "*the order comes in*" happens to be treated as a condition then 'order' may be defined as the involved thing-type of this condition. This will have an effect on the mapping of KCPM notions to conceptual models (e.g. state charts). The tool lists all candidates for involved thing-types: if there is more than one, the user has to choose one.

If the interpretation of a sentence is undesired for any reason, it may be skipped by disabling the check-box "Activated". In this case the interpretation result will not be mapped to the schema.

If a sentence fits with some given sentence patterns (e.g. if/then constructs) the tool can also derive cooperation-types. Otherwise, the user has to generate cooperation-types manually, applying the special function "edit cooperation-types" of the menu "Edit Schema". In this case, a list of all conditions and operation-types is presented in a separate window from which the user may select those he needs for defining a cooperation-type. It is also possible to relate logical operators (and, or, xor) to pre-conditions, post-conditions and operation-types. Clearly, this editing function may be also used for completing cooperation-types which were only partly derived by the interpreter (see figure 3).

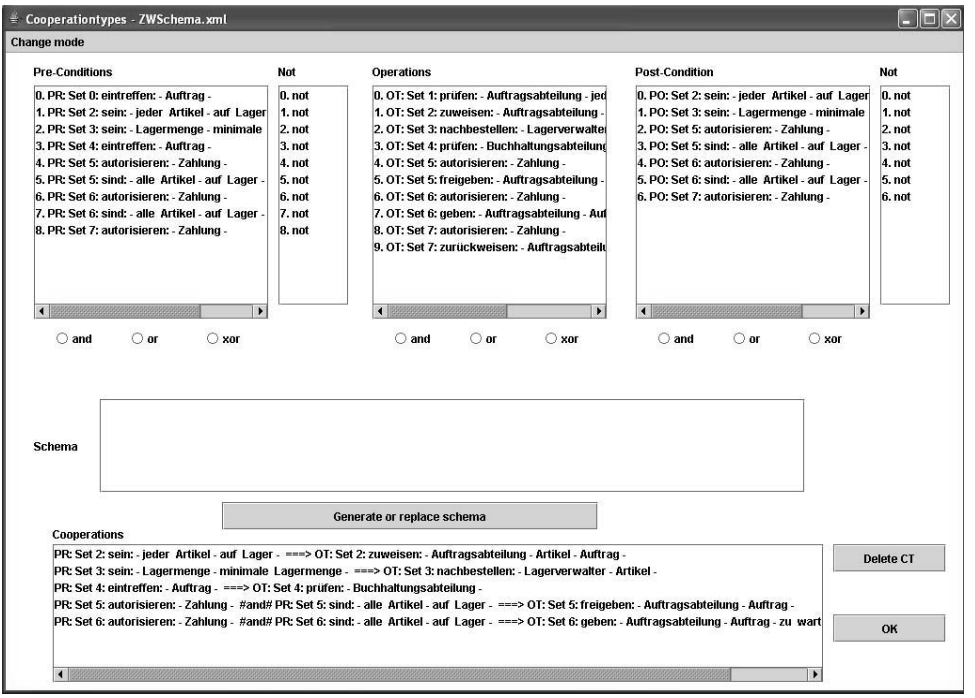

Figure 3: cooperation-type editing window

4 The Dynamic-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, clearly, will be replaced by more generalized terms later-on; in addition to that, to each KCMP schema entry a unique identifier is associated automatically). The left hand side of the corresponding window (see figure 4) 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). Figure 4 shows those cooperation-types which were automatically extracted from the initial sentences by the dynamic interpreter. This schema has to be completed by interweaving the cooperation-types appropriately based on matching identical pre- and post-conditions (a task that will be tool supported in future, too).

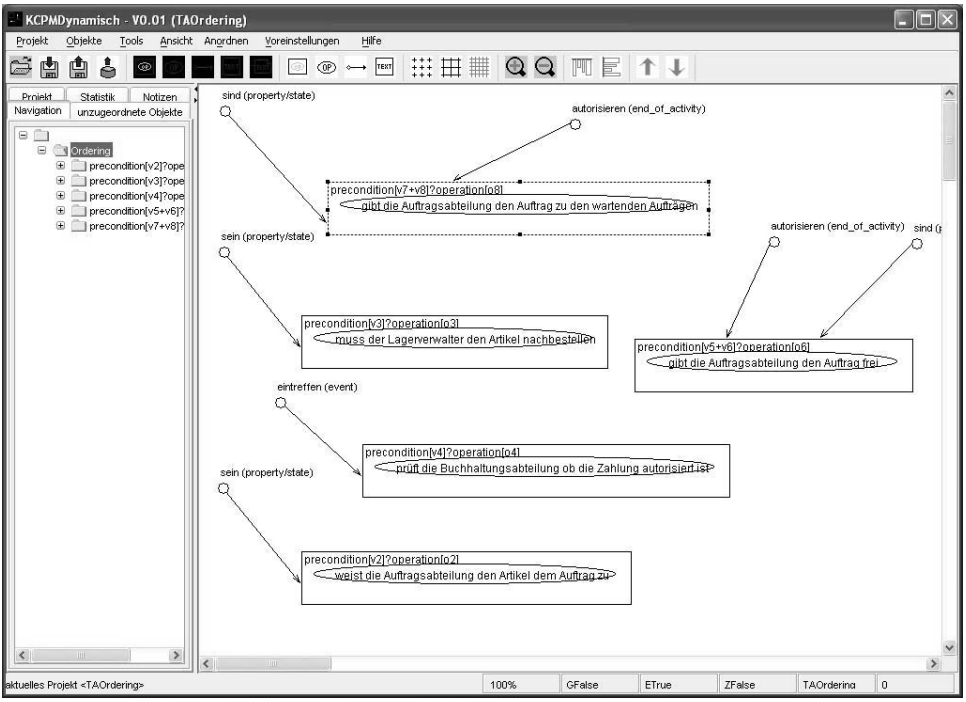

Figure 4: Dynamic Editor Tool

5 Web Interface and Traceability

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. This view has been implemented by means of a web interface (see figure 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. From our practical experience [St00], 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 forms had to be supported by appropriate tools.

The web interface presents the requirements sources separately (see section ‘Sources’ at the left hand side of figure 5). 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 there manipulation.

Actually, we distinguish three different kinds of sources:

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

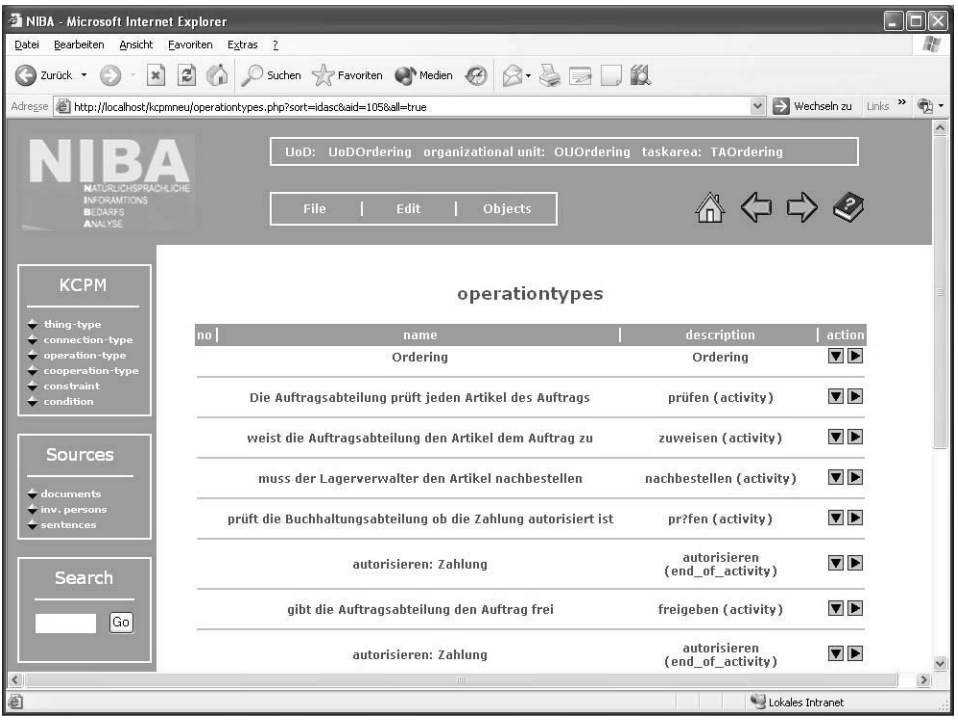

Figure 5: Web based glossary view for extracted operation-types

Sentence is the smallest unit of requirements source in our approach. It can be inserted manually but mainly will be automatically generated (referenced) during the interpretation and mapping process (see section 3).

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" reflects the fact that requirements often are given without any written sentence or document, they are just said. Especially when extracting

ontologies from the mind of the experts a document or something "written" does not always exist. To assure traceability also for such kinds of requirements sources, the feature was added to our tool. 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 enduser checks the forms and glossaries and notices that something is missing. Instead of writing some textual descriptions that have to be transformed to KCPM, the designer is able to reference this person directly.

To conclude, the web application supports the collection of KCPM notions as well as the collection of requirements in a glossary fashion. KCPM notions can be related to requirements sources thus allowing the user to see where the notions (resp. their need) come from.

6 Conclusion

Within this paper a tool chain was presented which is based on the premise, that computer supported analysis of requirements specifications is a step by step approach starting with core linguistic analysis by tagging and parsing [FW02]. From the output of such tools basic modeling notions can be extracted by interpretation and applying heuristic rules. These notions should be part of an intermediate modeling language (combined with an adequate representation concept) the user is able to understand and thus to validate. For that reason, KCPM was developed, a lean modeling language featured with a graphical and glossary like representation concept. Furthermore, traceability of schema entries to their corresponding requirements sources is supported as a further validation aid.

References

[Fl99] Fliedl, G.: „Natürlichkeitstheoretische Morphosyntax, Aspekte der Theorie und Implementierung." Habilitationsschrift, Gunter Narr Verlag, Tübingen, 1999.

[FK04] Fliedl, G.; Kop, Ch.; Mayr, H.C.; Winkler, Ch.; Weber, G.; Salbrechter, A.: Semantic Tagging and Chunk-Parsing in Dynamic Modeling. In: Meziane F.; Metais E.; (eds.) Proceedings of the 9th International Conference on Applications of Natural Language Processing and Information Systems, NLDB2004, Salford UK, Springer LNCS 3316 pp. 421 – 426.

[FW02] Fliedl, G., Weber, G.: Niba-Tag - A Tool for Analyzing and Preparing German Texts. In: Zanasi, A.; Brebbia C.A.; Ebecken, N.F.F.; Melli, P. (Hrsg.): Data Mining 2002 Bologna: Wittpress September 2002 (Management Information Systems, Vol 6), S. 331 – 337

[KK95] R. Kaschek, C. Kohl, H.C. Mayr: Cooperations - An Abstraction Concept Suitable for Business Process Reengineering, in Györkös, J. et.al. (eds.) *Re-Technologies for Information Systems, Proc. ReTIS'95*, R. Oldenburg Verlag, Wien, 1995.

[MK02] Mayr, H.C.; Kop, Ch.: A User Centered Approach to Requirements Modeling, *Proc. Modellierung 2002*, Lecture Notes in Informatics LNI p-12, GI-Edition, 2002, pp. 75-86.

[Le00] Leite, J.C.S.; Hadad, G.D.S.; Doorn, J.H.; Kaplan, G.N.: A Scenario Construction Process, *Journal of Requirements Engineering*, *Vol. 5 No. 1*, 2000, Springer Verlag, , pp. 38–61.

[Ro98a] Rolland, C. et al., A proposal for a Scenario Classification Framework, *Journal of Requirements Engineering, Vol. 3 No. 1,* 1998, Springer Verlag, pp. 23–47.

[Ro98b] C. Rolland, C. Ben Achour, Guiding the Construction of Textual Use Case Specifications, *Data & Knowledge Engineering Journal*, *Vol. 25 No 1-2,* 1998, North Holland Elsevier Science Publ., pp. 125–160.

[SM04] Salbrechter, A.; Mayr, H.C.; Kop, C.: Mapping Pre-designed Business Process Models to UML In: Hamza M.H. (Hrsg.): Proc. of the 8th IASTED International Conference on Softeare Engineering and Applications Cambrigde USA: ACTA Press 2004 (400-405).

[St00] Stark, M.: Geschäftsprozessmodellierung im konzeptuellen Vorentwurf, Diploma Thesis, University of Klagenfurt, 2000.

[We98] Weidenhaupt, K.; et al., Scenarios in System Development–Current Practice, *IEEE Software*, *Vol. 15 No. 2*, 1998, pp. 34–45.

[Zh00] Zhu, H.; Jin L.: Scenario Analysis in an Automated Tool for Requirements Engineering, *Journal of Requirements Engineering*, *Vol. 5 No. 1*, 2000, Springer Verlag, pp. 2 – 22.