The Four Levels of Use Case Description*

Barbara Paech

Institut für Informatik, Technische Universität München
Arcisstr.21, D-80290 München

Abstract. Use cases are a popular means of requirements engineering for object-oriented systems. However, there is a lack of guidance on the contents and position of use cases in the overall software development process. In the paper we identify four levels of information to be included in the use case description, based on existing methods in requirements specification and human-computer-interaction. This gives methodical guidance for the development of use cases and also for the integration of use cases and the analysis object model.

1 Introduction

Scenario-based description techniques have gained considerable popularity during the last few years - in research [Car95] and industry [WPJH98]. They are mainly used to describe the interaction between the users and the software system. Because of their exemplary nature they are relatively easy to produce and understand, both by software engineers and domain experts. Often they are accompanied by user interface prototypes [WPJH98], but they can be just a textual description of usage visions. Therefore they are particularly helpful in eliciting and negotiating requirements. In addition, they are used for documentation of requirements. This is also true for their most popular variant: Jacobsons *use cases* [Jac92]. Use cases are classes of interaction between a user and the system [Jac95]. Each use case corresponds to one major system function. They are described in the *use case model* depicting the name of the use cases, their `extends`- and `uses`-relationships and the actors involved, as well as by text capturing the behaviour of the individual use cases. As part of the method OOSE, use cases constitute the *requirements model*. They are the major input to the *analysis model* which consists of the *entity, control* and *interface objects*, their relationships and responsibilities. This model abstracts from design level considerations like the detailed behaviour of operations of the objects.

Since object models are not suitable for capturing functional requirements, use cases are very popular as a means of requirements engineering for object-oriented systems. However, a recent survey on scenario usage in industry has revealed a lack of guidance on the contents and the position of use cases in the overall software development process [WPJH98].

* This work was carried out within the Forschungsverbund Software Engineering, supported by the Bayerische Forschungsstiftung

In this paper we concentrate on the contents of use cases and their relation to the analysis model. We identify different levels of information to be included in the use case description. This gives methodical guidance for the development of use cases, but also for the integration of use cases and the analysis model. In particular, we show how to develop the analysis model incrementally along with the different levels of use cases.

The identification of the different use case description levels is based on existing methods in requirements specification and human-computer-interaction (HCI) [Pre94]. For the former, we include variants of the *entity/event modeling* of the standardized method SSADM [DCC92] to capture the data effects of the software system functions. For the latter, we include a variant of *task analysis* [Dia89] to ensure the adequacy of the system functions for the users tasks, a variant of *usability engineering* [Nie93] to ensure the ease of interaction between the users and the software system, and some kind of *dialogue specification* to capture the information presented to the user and the user control over the execution of the system functions. Task analysis and usability engineering are the major source of information for identification of the system functions, while entity/event modeling and dialogue specification make explicit the behaviour of the system functions as input to the development of the analysis model.

For each level of use case description we give an exemplary graphical description technique. These diagrams serve as an illustration of the kind of information to be captured at each level. Depending on the needs of a software development project, they can be used in addition to the use cases to emphasize important aspects of the system functions.

This paper is structured as follows: in section 3, we describe the four different levels of use cases, their content and appropriate modeling techniques. In section 4, these models are integrated with the analysis models. All of this is exemplified with the ubiquituous library example. Two use cases for this example are given in section 2. Related work is discussed along the way.

2 Example Use Cases

The discussion in this paper will be based on the following example use cases for `book return` and `book search` in a library.

Book Return The course of events starts when a reader hands a book to the librarian to return it to the library. The librarian enters the book number. The system retrieves the title and author of the book as well as the reader identity, for the librarian to acknowledge that the correct book is returned from the correct reader. In reaction to the acknowledgement of the librarian the system updates book and reader data and checks whether the book has been reserved. If so, an Email message to the owner of the reservation is created and shown to the librarian. The acknowledgement of the librarian triggers the sending of the Email. Finally, the success of the whole transaction is notified to the librarian.

Book Search A student is searching for literature on requirements engineering. Therefore she chooses relevant key words from the list and issues the corresponding query. However, the list of results is too long. Thus, she decides to look only for books not older than three years. She opens the attribute list and fixes the possible years. The corresponding query yields 40 books. One after the other she opens the books on the screen and looks through the contents. The interesting ones she marks. Then she reserves the marked books and sends the list to the printer. She also checks which of the interesting books are available. Since only a few of them are available right now, she decides to look again at some of the books which did not mark. Therefore, she activates the last query (together with the results) and searches through the results again.

3 Use Case Description

OOSE [Jac95] - as most other object-oriented methods - presupposes a textual requirements specification. This is transformed into the requirements model consisting mainly of use cases, possibly accompanied by a domain object model which captures the most important application objects, and by interface descriptions capturing what the user will see on the screen. These models are the major input to the analysis model consisting of the interface, entity and control objects. The analysis model typically only shows the objects and their associations, but not their operations. Jacobson recommends to incorporate operations only at the level of the design model, because they often depend on the implementation environment. In both cases, the use cases are the starting point for deriving the operations of the objects.

Along the transition from the requirements specification to the analysis object model, we have found four major classes of decisions to be taken. These classes of decisions have been identified by a thorough literature survey in the realm of requirements engineering and HCI [Pae98], as well as in example industry projects. Each of them is well studied in the literature. We mention typical approaches in the following and give examples for typical modeling techniques.

Work Division For each use case one has to identify the division of work between the users and the system. This includes the automation of tasks previously performed by the users, but also major changes to the work processes. E.g. book return could be handled - alternatively to the use case of section 2 - fully automatic by a system which offers an interface to deposit the book.

Application-specific System Functions For each use case the data used and affected has to be determined. This is the main focus of traditional analysis methods like SSADM. Functions working on the application data are called *application-specific* in contrast to system functions which support the detailed work organisation of the users.

Work-specific System Functions The usability features of each use case have to be determined. Thus, in addition to the application-specific functionality, *work-specific* system functions have to be identified which allow the users

to organize the computer-based work more comfortably. A typical example is the undo-function, more specific for the library is a function allowing to store and retrieve queries to the book data base.

Dialogue The dialogue steps for acquiring the input to the system functions from the user and delivering the output to the user, as well as the user control over the function execution have to be determined. This is the main focus of user interface modeling techniques.

In Jacobsons use case approach, these decisions are not seperated. The use cases are produced in one step and all information is captured in one textual description. Therefore, it is difficult to check whether all important decisions have been taken, as well as to trace the decisions to the analysis object model. In contrast, we propose to develop each level separately. Starting from a preliminary use case text, each level is examined and the text is extended accordingly. If one level is very complex, a separate model is developed.

In the rest of this section we discuss each level in more detail. By way of example, we introduce a graphical description technique for each level. Because of the explicit use of the graphical models, we do not incrementally develop the use case text. The text of the two use cases in section 2 represents the final version after examination of each level.

3.1 Work Division

Most analysis and design methods do not explicitly acknowledge the fact that system design is also work design. Only methods inspired from HCI, e.g. TRIDENT [BHLV95] or TASK [BJ94] give explicit support for work design. This typically includes some kind of *task analysis* [Dia89] which gathers information about the user tasks, the objects and some attributes like repetitions, duration, priority and the like. Task analysis models concentrate on the user actions, only. In the spirit of use cases we propose to describe both, the user and system tasks, in one model. As a graphical description technique use a variant of UML activity diagrams [BRJ97] with one swimlane for the system and one or more swimlanes for the users and their external partners. We call these diagrams *Work Process Diagrams*. As an example, figure 1 shows the activity diagram for the book return.

Boxes represent activities. We have extended the notation of the activity diagrams to allow activities which need not be given in detail to be involved in a sequence of communications. This is denoted by a dotted line within the activity box. Thus e.g. the `Accept Book`-activity of the counter person is divided into two subactivities. The first one takes the book from the reader and issues the return command to the software system. The second one takes the output from the software system and acknowledges the return to the reader. The little diamond represents decision. Outgoing arrows are labeled with the different possibilities. In the example, there are two possibilities: either the returned book had been reserved or not. In the former case, a notification is send that the book is now available. In both cases the success of the function execution is signaled to the

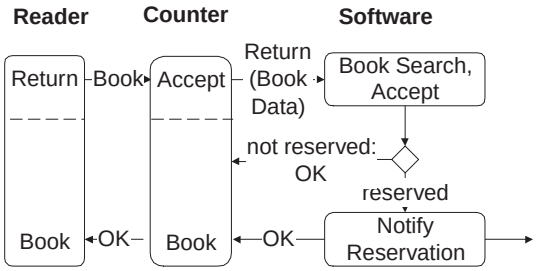

Fig. 1. Work Process

user at the counter who in turn communicates this to the reader. In comparison with the book return use case of section 2 one can see that at this level we include neither the detailed data effects nor the information presented to the user or the detailed user commands.

The major difference to UML activity diagrams is that we label the arrows with data flow, not object flow. The emphasis is here on the dependencies between activities and their distribution between users and the system, not on the objects affected.

At this level of use case description, one can experiment with different possibilities of work division between the user and the system. The similarity of work process diagrams to business process descriptions, e.g. [Sch92], allows for an easy transition from strategic business process models to work place descriptions. To allow the developer an evaluation of the work design, each user activity should be described in more detail [BJ94] regarding aim, degree of freedom, causes, preconditions, subtasks, postconditions, input, ressources and output.

Then the complete set of user activities can be analyzed by methods and check lists from work psychology to determine the adequacy of the work design for human labour [Uli94, DVZ+93].

3.2 Application-specific system functions

Models of system functions and their data effects have been heavily used by modern structured methods like SSADM [DCC92]. Inspired by *Jackson System Development* [Jac83], they provide models to capture the interplay between data and system functions like entity life histories or effect correspondence diagrams. Through the advent of object-oriented methods this level of function description has been abandoned in favour of sequence or collaboration diagrams capturing the operation calls between different objects and state transition diagrams capturing the sequences of operation calls to a specific class. While these models are adequate to capture design decisions, they are too detailed for requirements capture and analysis. The reason is that *one* data-flow model of data effects corresponds to *several* models based on object communication. As an example,

consider figure 2 and figure 3. The first shows an activity diagram for the book return use case. This kind of diagrams is called *function process diagram* in the following.

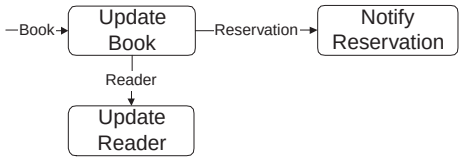

Fig. 2. Function Process Diagram

It shows that book and reader data is updated and the reader information flows from the book update activity to the update reader activity, as well as the reservation information to the notification activity. The book information must be input to the book return function.

Figure 3 shows two possible UML collaboration diagrams describing the data effects of book return through object communication. On the level of object communication, data flow has to be transformed into control flow. For the book return function, there are essentially two possibilities: either the book object triggers the reader and the reservation object or there is another control object sequencing book update, reader update and notification. In general, there are much more possibilities.

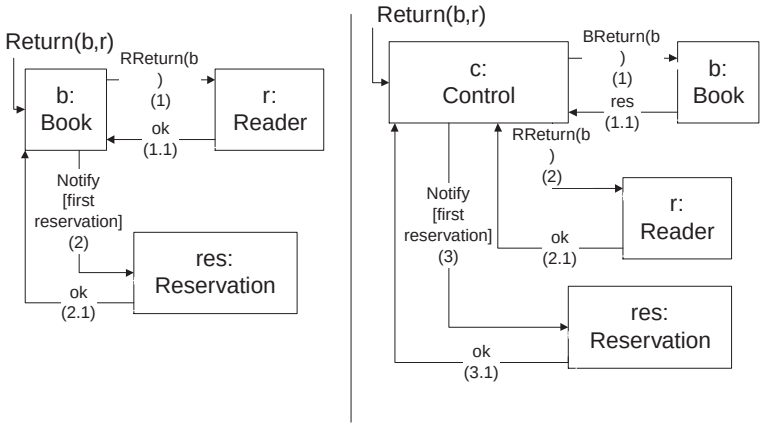

Fig. 3. Collaboration Diagrams

This example demonstrates that to describe data effects on the level of object communication one has to make decisions about the detailed object interaction. However, these decisions might depend on overall design and implementation decisions like the programming language used. Also, the decisions about the data effects are buried under the communication details, making the collaboration diagrams more difficult to read than function process diagrams.

At this level of use case description data effects are explicit und sequencing dependencies between them. One can experiment with different ways of navigation between different data updates. Function processes detail the activities of the software system identified in the work processes to the level of the data effects. One should note that the kinds of activity diagrams used for work and function processes are quite different. The first describes data flow between different actors, the second data flow within the software system. UML does not make a difference between activity diagrams with or without swimlanes.

3.3 Work-specific system functions

Analysis and design methods typically only support the identification of application specific system functions relating to the application data. However, to be useful in every day work the system also has to offer work specific functions. *Usability engineering* [Nie93, Car95] offers a whole range of techniques, how to identify and design this work specific functionality. Typically, no diagrammatic models are involved, since the focus is on identifying usability features and less on sequencing of activities or data effects. This level of use case description is best supported by use of usage scenarios, mock-ups or interface prototypes for envisioning the work-specific functionality, and the use of walk through, observation, ethnography and the like to evaluate real system usage. The book search use case of section 2 is a typical example of a scenario focusing on work-specific functionality. It makes explicit how the user may construct queries by choosing from a list of keywords or filling in an attribute list. Also, the storage of queries for later retrieval is described.

This level of use case description is especially useful for weakly-structured system functions like search which require a rich set of usability features. [Suc95, HB95] give examples for similar requirements elicitation techniques, especially in the context of designing new technology.

3.4 Dialogue Modeling

The levels described above have identified the application- and work-specific functionality and determined the data effects of the system functions. To be able to derive interface objects from the use cases, use cases must also capture the information visible at the interface and the user control. In HCI these aspects are captured in some sort of dialogue models. There is a whole range of dialogue models which mostly aim at automatic generation of user interfaces from the models [Sze96]. We use a variant of *interaction diagrams* developed by Denert [Den92]. Figure 4 shows the interaction diagram for the book return use case.

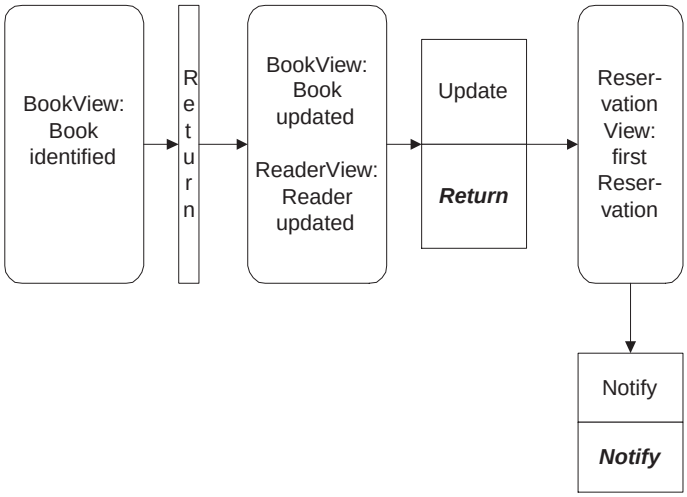

Fig. 4. Dialogue Model

Boxes with rounded angles represent user interface states in terms of data views to be seen at the screen and a rough description of the information seen in this view. Boxes with sharp angles represent user actions. If the user action triggers a system functions, this is shown in italics in the lower half of the box. The arrows sequence the information presented to the user and the user actions.

In the example, the return function request is issued from a view showing the book to return. As a response, the data updates to reader and book are shown, but only through an explicit update request the update is made permanent by means of the return system function. After that the reservation to be notified is shown and the notification is triggered by an explicit user request. Thus, at this level the choice between an object-or function-centered user interface is made. Here we have choosen the object-centered style, where first the object is selected and then the functions.

In comparison with the complete use case description of section 2, one can see that at this level we include all the detailed user actions as for example the acknowledgement of the update.

4 Integrating Requirements and Analysis Modeling

Because of the separation of requirements and usage concerns into different levels of use case descriptions, it is also possible to develop the analysis object model incrementally, where each use case level corresponds to a certain level of abstraction of the object model. In the following we will show the interplay between the object model and the use cases in the library example.

4.1 Application Data Model

The activities of the work process models refer to the application data. This information can be captured in the *application data model* (also called domain model) as recommended by Jacobson. We use the term application data model to emphasize that the entities are not viewed as active objects communicating with each other, but only as passive data (entities) processed by the activities. In particular, relationships between the entities do not represent communication links but only dependencies between the entities which can be observed in the environment.

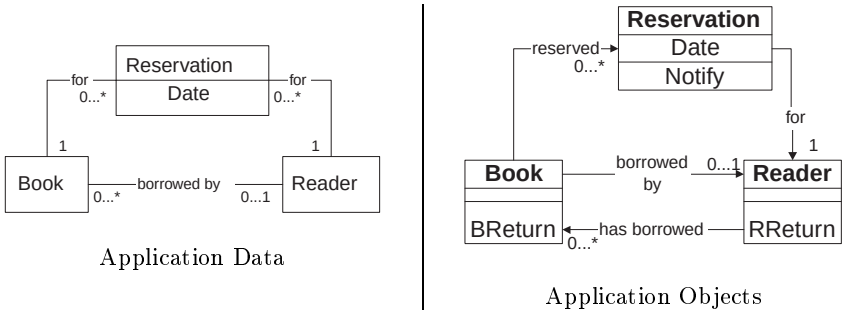

Fig. 5. Application Data and Objects

The left half of figure 5 shows the data relevant for the book return use case. We use the notation of UML static structure diagrams. Boxes represent entities and links represent associations. The latter can be adorned with multiplicities.

4.2 Application Object Model

For function process modeling, one has to fix navigability between the entities. This is necessary to be able to determine which entities are affected by which function. So e.g. for the book return function process, the borrowing association between reader and book in the application data model can be detailed into a one- way reference from book to the reader or the other way round or into a two-way reference.

As exemplified in the right half of figure 5, in the example we have choosen the two-way-reference. According to UML this is represented by arrow heads of the links. The associations linking reservation with book and reader, respectively, are resolved into one-way-references. The activities of the function process diagrams can be associated with the corresponding entities (called `BReturn, RReturn` and `Notify` in the example). Thus, at this level the entity diagram is transformed into an object model with navigation information and operations which only affect single classes.

4.3 Work Object Model

The identification of work-specific functions leads to the addition of work-specific objects and their activities. As shown in figure 6, for the book search function a session object is added which consists of session items. The latter are tuples of queries and results, where a result is a possibly empty list of books.

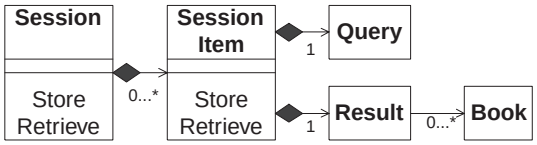

Fig. 6. Work Objects for Book Search

4.4 Control Objects

Up to now, operations which affect several objects are not represented in the object model. It is a design decision whether to associate such functions with entity objects or whether to include specific control objects corrresponding to the use cases. Because of the explicit representation of such functions through the function processes, there is no need to take this decision during requirements capture. If taken, it can be documented in the object model through the addition of control objects, their references to entity objects and the operation corresponding to the system function. The left half of figure 7 shows the extension of figure 5 by an control object for book return.

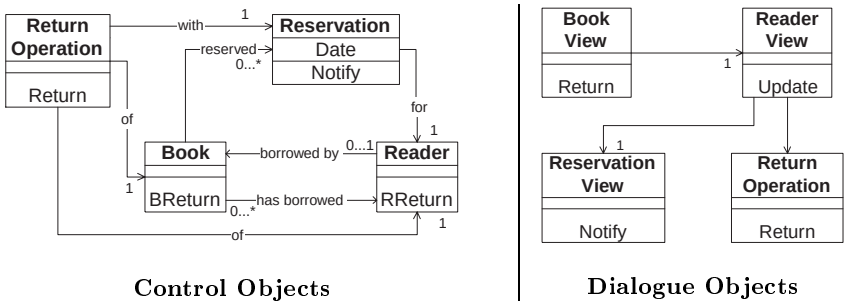

Fig. 7. Control and Dialogue Objects

4.5 Dialogue Objects

In the dialogue models, views on the entity objects visible to the user are mentioned. As described in [Zie97], these views can be systematically derived from the entity objects. Typically, one needs different kinds of attribute views to represent entity objects or parts of them which are input to system functions, container views for sets of objects, function views to represent control objects and notification objects. In the user interface these views are grouped into windows. However, this grouping decision can be postponed to the transition from the analysis models to the design model. At the level of the analysis model, the object model shows the individual views. The user actions are incorporated as operations of the view objects. Thus, the dialogue object model shows reference relationships between the interface objects and possibly control objects. In the right half of figure 7 we have incorporated the control object for book return. If no decision has been made about the allocation of the system function, then a dummy object is used. In comparison with the global dialogue description of figure 4 the object model makes explicit the navigation between different views.

5 Conlusions

We have proposed a separation of different requirements and usage concerns into different levels of use case description taking into account the wide variety of approaches for human-computer-interaction modeling. For each level we have identified an appropriate diagrammatic description technique. We also have shown, how to incrementally develop the analysis object model according to the different use case levels. In contrast to mesage sequence diagrams which are typically used to depict use cases [Jac92, RKW95, RAB96], we propose to use data-flow based process models for work and function process models, text for usage scenarios and state transition diagrams for dialogue models. Message sequence charts focus on object communication which is unnecessarily detailed for requirements modeling purposes. The decision about communication can be postponed to the transition from the analysis model to the design model.

Thus, our work can also be seen as an extension of object-oriented methods by intermediary models abstracting from communication details to allow for better focus on the major requirements decisions.

Acknowledgements

We thank Bernhard Deifel for helpful comments to a draft version of the paper.

References

[BHLV95] F. Bodart, A.-M. Hennebert, J.-M. Lehereux und J. Vanderdonckt. A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype. In *Interactive Systems: Design, Specification and Verification.* Springer, 1995.

[BJ94] A. Beck und Ch. Janssen. TASK - Technik der Aufgaben- und Benutzerangemessenen Software-Konstruktion. Technischer bericht, IAT - Institut für Arbeitswissenschaft und Technologiemanagement, January 1994.

[BRJ97] G. Booch, J. Rumbaugh und I. Jacobson. The Unified Modeling Language for Object-Oriented Development, Version 1.1, 1997.

[Car95] J.M. Carroll, Hrsg. *Scenario-based Design*. John Wiley & Sons, 1995.

[DCC92] E. Downs, P. Clare und I. Coe. *Structured Systems Analysis and Design Method: Application and Context*. Prentice-Hall, 1992.

[Den92] E. Denert. *Software-Engineering*. Springer Verlag, 1992.

[Dia89] D. Diaper. *Task Analysis for Human-Computer Interaction*. Ellis Horwood Limited, 1989.

[DVZ+93] H. Dunckel, W. Volpert, M. Zölch, U. Kreutner, C. Pleiss und K. Hennes. *Kontrastive Aufgabenanalyse im Büro*. Teubner, 1993.

[HB95] K. Holtzblatt und H.R. Beyer(ed.). Special Issue on Requirements Gathering - The Human Factor. *CACM*, 38(5), 1995.

[Jac83] M. Jackson. *System Development*. Prentice-Hall, 1983.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[Jac95] I. Jacobson. The Use-Case Construct in Object-oriented Software Engineering. In J.M. Carroll, Hrsg., *Scenario-based Design*, Seiten 309–336. John Wiley & Sons, 1995.

[Nie93] J. Nielsen. *Usability Engineering*. Academic Press, 1993.

[Pae98] B. Paech. *Aufgabenorientierte Softwareentwicklung - Modellierung von Unternehmen, Arbeit und Software*. submitted as Habilitationsschrift, TU München, 1998.

[Pre94] J. Preece. *Human-Computer Interaction*. Addison-Wesley, 1994.

[RAB96] B. Regnell, M. Andersson und J. Bergstrand. A Hierarchical Use Case Model with Graphical Representation. In *ECBS*. IEEE, 1996.

[RKW95] B. Regnell, K. Kimbler und A. Wesslen. Improving the Use Case Driven Approach to Requirements Engineering. In *Symposium on Requirements Engineering*, Seiten 40–47. IEEE, 1995.

[Sch92] A. Scheer. *Architecture of Integrated Information Systems: Foundations of Enterprise Modelling*. Springer Verlag, 1992.

[Suc95] L. Suchman(ed.). Special Issue on Representations of Work. *CACM*, 38(9), 1995.

[Sze96] P. Szekely. Retrospective and Challenges for Model-Based Interface Development. In *Design, Specification and Verification of Interactive Systems*. Springer, 1996.

[Uli94] Eberhard Ulich. *Arbeitspsychologie*. Schäffer-Poeschel Verlag, Stuttgart, 1994.

[WPJH98] K: Weidenhaupt, K. Pohl, M. Jarke und P. Haumer. Scenario Usage in System Development: A Report on Current Practice. In *Int. Conference on Requirements Engineering*. IEEE, 1998. to appear.

[Zie97] J. Ziegler. ViewNet - Konzeptionalle Gestaltung und Modellierung von Navigationsstrukturen. In R. Liskowsky, B.M. Velichkovsky und W. Wünschmann, Hrsg., *Software Ergonomie 97*, Bd. 49 of *German Chapter of the ACM - Berichte*, Seiten 343–350. Teubner, 1997.

This article was processed using the LaTeX macro package with LLNCS style