Formal User-centered Requirements Engineering*

Barbara Paech
Institut für Informatik, Technische Universität München
Arcisstr.21, D-80290 München
paech@informatik.tu-muenchen.de

Introduction

In this extended abstract we discuss the combination of formal and user-centered requirements engineering. Formal methods like Z[Wor92], VDM[AI91], LOTOS,ESTELLE and SDL[Tur93] focus on requirements specification in terms of an abstract behaviour description of the software system to be build. User-centered requirements engineering is concerned with identifying the users of the software system and understanding the tasks to be supported by the system. Here we propose

- a *formal enterprise model* and a *formal interface model* as a means of closing the gap between the informal description of users and tasks and the formal description of the software system and
- the use of *scenarios* as first-class modelling elements as a means of deriving the enterprise and interface model systematically.

This work is part of the SYSLAB-project which aims at a methodology integrating formal and pragmatic software development [Pae95]).

Formal user-centered requirements engineering

In the following we sketch the formal models and a suitable requirements engineering process.
The enterprise is viewed as interacting *areas* where each area corresponds to a complex task (often, but not necessarily associated with an organisational unit of the enterprise). Thus the enterprise model consists of a task model which describes the areas and their subtasks and an object model which describes the data relevant to these tasks. The behaviour of the tasks is given by state-transition diagrams whose transitions are labelled with triples (inputs, subtask,outputs). As an example consider the management of a simple production company in figure 1. It interacts

*This work was carried out within the project SysLab, supported by Siemens Nixdorf and by the Deutsche Forschungsgemeinschaft under the Leibniz program

with the customers (channels `cm,mc`), the production (channels `pm,mp`) and the store (channels `sm,ms`)

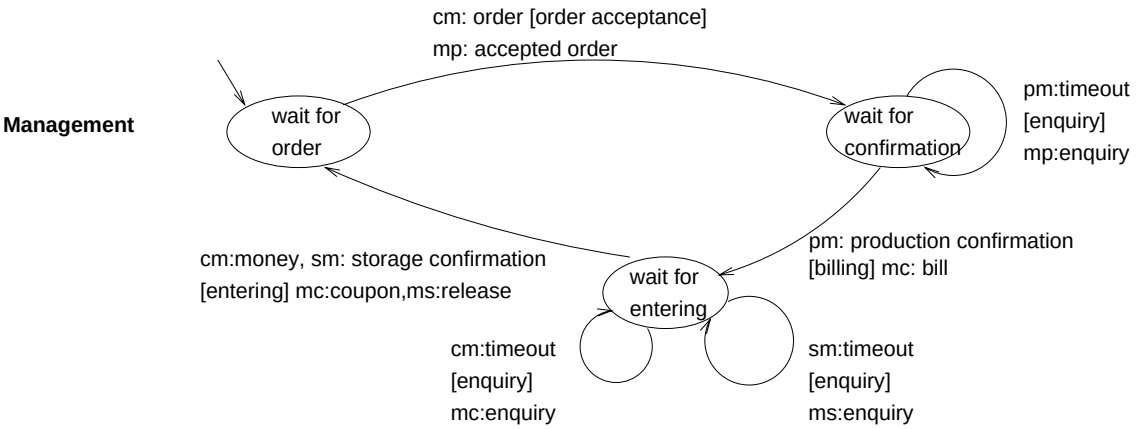

Figure 1: Task behaviour description

The interface model describes the interaction between the user and the software-system. On one hand the *services* of the software-system are identified, where services are the subtasks allocated to the software-system. The behaviour of the software-system is described in terms of the services. Here again state-transition diagrams are used. On the other hand the objects relevant to the services are collected in the interface object model.
These models are derived from business process scenarios and usage scenarios, which specify exemplary dependencies between tasks and exemplary interactions between users and the software system, respectively.
Altogether we propose the following process for the engineering of functional requirements

1. In a mixed bottom-up and top-down manner business process scenarios are identified and described on different refinement levels.

2. On each refinement level from the scenarios the enterprise object model and the full-behaviour descriptions for the tasks in terms of state-transition diagrams are derived.

3. At a certain refinement level business process scenarios are transformed into usage scenarios. Further usage scenarios for individual workplaces are added.

4. As for business processes usage scenarios are described on different refinement levels and analyzed by deriving the interface object model and the state-transition-diagrams for the software-system and the tasks supported by the system.

Related Work

Techniques for modelling processes of the application system have been used in most pragmatic software development techniques (e.g. SSADM[DCC92]). However, these models have been used informally to achieve an understanding of the application. In our approach the emphasis is on a thorough analysis of the enterprise. Both, business process scenarios and task behaviour descriptions are first class modelling elements which

- can be incrementally developed (e.g. through refinement)
- analyzed (e.g. through simulation).

Most pragmatic software development methods also use some kind of scenarios to determine the interface of the software system (e.g. FUSION [CAB+94]). But again they are mostly used informally. To our knowledge the only approach making heavy use of usage scenarios is OOSE[Jac92] which is often called "the use-case approach". Use cases are also maintained throughout the development process as independent modelling elements. Thus to some extent our work can be understood as giving a formal basis to Jacobsons' approach.
Similar to the approach of the TASK methodology ([BJWZ94]) we view the modelling of tasks as the key element of combining requirements analysis methods and user-centered design. The formal interface model is the basis for a detailed user-interface design. The task description by state-transition diagrams can directly be used for prototyping purposes. Another advantage of formality is the possibility of proving important properties of the specification like consistency, safety or liveness.

Conclusion

We have discussed the role of formal enterprise and interface models within the process of user-centered requirements engineering. The approach outlined above is far from complete. The methodological steps leading from the scenarios to the full behaviour descriptions are not fully understood. In [Bro95] the mathematical foundations are sketched. This formal basis should be hidden from the software developer. However, it is essential for powerful tool-support allowing for a thorough validation of the behaviour descriptions.

References

[AI91] D. Andrews and D. Ince. *Practical Formal Methods with VDM.* Series in Software Engeneering. McGraw-Hill, 1991.

[BJWZ94] A. Beck, C. Janssen, A. Weisbecker, and J. Ziegler. Integrating object-oriented analysis and graphical user interface design. In *Software En-*

gineering and Human-Computer Interaction, LNCS 896, pages 127–140. Springer Verlag, 1994.

[Bro95] M. Broy. Mathematical system models as a basis of software engineering. In *LNCS 1000*. Springer Verlag, to appear, 1995.

[CAB+94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development - The FUSION method.* Prentice Hall, 1994.

[DCC92] E. Downs, P. Clare, and I. Coe. *Structured systems analysis and design method: application and context.* Prentice-Hall, 1992.

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

[Pae95] B. Paech. A methodology integrating formal and informal software development. In M.Wirsing, editor, *ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*, pages 61–68, 1995.

[Tur93] K.J. Turner(ed.). *Using formal description techniques - an introduction to ESTELLE, LOTOS and SDL.* John Wiley & Sons, 1993.

[Wor92] J.B. Wordsworth. *Software Development with Z.* Addison-Wesley, 1992.