Elaborating analysis models with tool support

Gregor Buchholz and Peter Forbrig

University of Rostock, Institute of Computer Science, Albert Einstein Str. 21,
18059 Rostock, Germany
{gregor.buchholz, peter.forbrig}@uni-rostock.de

Abstract. Integrating models as essential elements into the software development process is supported by numerous methods and tools but the creation of such models still bears a considerable challenge. This paper proposes a structured modeling of tasks and activities during the requirements analysis in order to pave the way for the very early utilization of models. A tool implementation demonstrates the elaboration of models based on scenarios.

Keywords: Task Models, Scenario Analysis, Model Based Development

1 Introduction

1.1 Motivation

A considerable number of methods and tools for the integration of task models into software development and usability testing can be found. The underlying paradigms (model-based and model-driven development) include the use of models in all development stages [4]. Models for design have to be deduced from requirements analysis results which would be much more convenient if the analysis stage itself purposed the formalization of its output as a model. While it might be a feasible objective to model user intentions, goals and tasks at a high level it becomes more and more difficult and laborious as the level of detail grows [7]. We will introduce a tool supported method to create such models during the requirement analysis phase.

1.2 Related Work

Deriving descriptions of user and system tasks from recorded scenarios has been conducted and documented before. The *CelLEST* process [9] reconstructs a navigation model of an existing application as state graph with screenshots as states and user inputs as state transitions, aiming at the documentation of existing systems. In [5] *ActionStreams* is proposed for the creation of task models based on user behavior. The users' activities are observed and recorded for a comparatively long time ("a little more than a day") and a grammar production represents the autonomous learning of the system. An example for the analysis of interaction sequences to deduce domain knowledge for learning systems via *Sequential Pattern Mining* (SPM) is the

framework described in [3] that has been proved in the tutoring system *RomanTutor*. The *ProM* framework [2] features a wide variety of algorithms to discover and mine processes and a lot more.
In addition to the construction of a formal correct model our approach includes the semantic meaningful abstractions of actions to task.

2 Collaborative Model Building as Learning Activity

The goal of the analysis phase is to define the problem to be solved and the desirable outcome is a clear and precise description of tasks for which software support has to be built. That elaboration of a common understanding of the problem is challenging and methods and artifacts are needed that help to arrange the acquired pieces of knowledge, to reveal missing information and to support discussion.

2.1 Gathering Domain Knowledge

This work supports the idea of the employment of task models as domain-independent modeling technique during all phases of software development. The selection of techniques for the first requirements elicitation is rather unaffected by the intention of task modeling. As usual, functional as well as non-functional requirements are wanted. That includes the identification of user roles, business objects, context information, use cases etc. and also descriptions of scenarios.

2.2 Structuring Domain Knowledge

The formal specification is started with the main goal of the problem to be solved on an abstract level which is then refined into more concrete tasks. Temporal relations between tasks can be modeled with the operators known from CTT (*Concurrent Task Trees*) [8]; missing operators will be deduced later. Each model can be assigned to a role as a separation into distinct models for different roles may be useful.

The requirements elicitation not only unveils abstract tasks but will also comprise more concrete actions. These actions can be assigned as sub-elements to tasks, so as to build groups and specify which section an action belongs to.

2.3 Scenario-based Exemplifying

In the next step the models with tasks and actions are used to generate examples of scenarios. The developed tool supports the role-based recording of actions as well as a file import functionality to include traces produced by other systems, such as a log file from an existing software system.

2.4 Construction of Detailed Models

A tree with abstract tasks ("Manage account", and "Use account" in Fig. 1), a set of actions assigned to these tasks, and a set of action traces exemplifying the fulfillment of the tasks (see Fig. 1) are the basis for model generation. The resulting tree structure is a candidate for suitable representation of processes the system is to support (see the tree in Fig. 1). The tool features three steps of model generation:

Pattern recognition identifies frequently occurring sequences of actions in the traces based on the assumption, that actions observed together may be steps of a more abstract task. For finding such patterns the *Apriori* algorithm, *PrefixSpan*, and *BIDE+* are implemented. The recognition results in groups of potentially related actions. A reviewing of these groups comprises a reasonableness check and labeling.

The goal of **hierarchy creation** is to combine the manually *top-down* decomposed tasks with the generated parts created *bottom-up*, only loosely coupled to the upper part. In short, this step merges analytic and synthetic parts using a *LearnModel* algorithm modification without objects assigned to tasks and with the notation and operators of CTT until all actions from the protocols are assigned directly (one edge connecting both nodes) or indirectly (one or more nodes between the nodes) to a task.

In the process of **identifying temporal relations** initially all sibling nodes are considered to be in an *enabling* relation following their first occurrence in the first trace of the analyzed trace set they occur respectively. The protocol analysis corrects this assumption and *order independence, iterations* and *concurrency* are suggested. Each time an operator is modified all operators up to the root are modified, if violated. This process can be repeated any number of times, selecting different sets of task traces and with varying settings for operator detection. Thus, different rival model versions can be created and discussed.

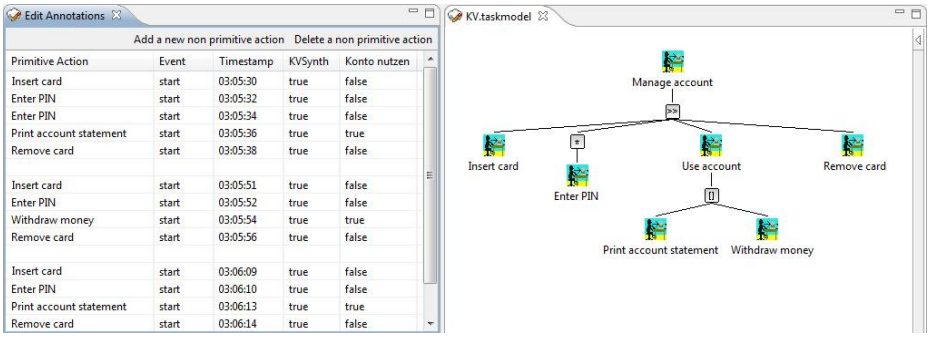

Fig. 1. Sample scenario traces and resulting model.

2.5 Evaluating the Progress and Starting of new Iteration

The models can be simulated by the provided tool. Therefore, one or more clients connect to the server; a stored model is selected and opened in the simulation module with a specific role. Then, the execution of actions is simulated by activating the actions in the wanted order according to the previously detected scenarios. If any needed action is not accessible (that is, it is not in the enabled task set) or an action is

enabled but known to be forbidden at that time, the model can be corrected. These simulations' protocols can serve as input for the next model generation iteration.

3 Conclusion and Future Work

This paper describes a method to establish models as an integral part of requirements analysis. Traces of scenarios can be combined with the structured knowledge about tasks to formalize the findings about the current situation. The approach employs modified data mining and association algorithms to derive relations between actions and tasks and build a hierarchical task structure. Results of the process have to be interpreted as a model of an existing situation. The approach and the tool support may help in simplifying the transfer of previously informal knowledge to structured and formal presentations.

Next steps will be the improvement of tool for annotations and references to other artifacts. Furthermore, it has to be investigated in which way the implemented tool can contribute to test prospected scenarios. In this situation, not the generation but the validation of models will be in focus.

References

1. Colombo, P., Khendek, F., Lavazza, L.: Requirements Analysis and Modeling with Problem Frames and SysML: A Case Study, In: Modelling Foundations and Applications, Springer, Berlin (2010)
2. van Dongen, B., Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, van der Aalst, W.M.P.: The ProM framework: A New Era in Process Mining Tool Support. In: Ciardo, G., Darondeau, P. (eds) Application and Theory of Petri Nets 2005, LNCS vol. 3536, pp 444--454. Springer, Berlin (2005)
3. Fournier-Viger, P., Nkambou, R., Nguifo, E. M.: A Knowledge Discovery Framework for Learning Task Models from User Interactions in Intelligent Tutoring Systems, Proc. MICAI 2006, pp. 765--778, Springer (2008)
4. Johnson, S.K., Brown, A.W.: A Model-Driven Development Approach to Creating Service-Oriented Solutions, Proc. ICSOC 2006, pp. 624-636, Springer (2006)
5. Maulsby, D.: Inductive Task Modeling for User Interface Customization, Proc. IUI 1997, pp. 233--236, ACM (1997)
6. Mazón, J.-N., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses, In: Advances in Conceptual Modeling & Foundations and Applications, pp. 255--264, Springer, Berlin (2007)
7. Paris, C., Lu, S., Linden, K.V.: Environments for the Construction and Use of Task Models, In: The Handbook of Task Analysis for Human-Computer Interaction, pp. 467--482, Lawrence Erlbaum Associates (2004)
8. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications, Springer (1999)
9. El-Ramly, M., Stroulia, E., Sorenson, P.: Recovering Software Requirements from System-user Interaction Traces, Proc. SEKE 2002, pp. 447--454, ACM Press (2002)
10. Seyff, N.: Exploring how to use scenarios to discover requirements, In: Requirements Engineering, pp. 91--111, Springer, London (2009)