Task models for safe software evolution and adaptation

Jan Van den Bergh, Deepak Sahni, Karin Coninx

Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{firstname.lastname}@uhasselt.be

Abstract. Many industrial applications have large and complex interfaces that grow incrementally over time. Typically, these interfaces will be used by people with different user profiles. The combination of these facts demands a software methodology and tool support that ideally allow consistency checks and configuration in order to avoid a system to become unusable. In this paper, we present an approach in which task models are used throughout the design and development cycle up to the final application. The task model is not only used at design time, but is also used to check for potential problems with e.g. consistency during configuration of the final application.

Key words: customization, task model, evolution, consistency, model-based user interface design

1 Introduction

[1] The task model has been one of the central models in several model-based and model-driven development approaches [?,?,?]. This central role has been based on the fact that it offers a structured overview of the tasks that have to be performed by a user. Furthermore, a notation such as ConcurTaskTrees also allows for checking properties such as reachability, reversibility and mutual awareness [?] although the currently tools do not have support for these capabilities.

When considering approaches for large industrial applications, adoption of a model-based development cycle meets some resistance even while offering these benefits. This is especially the case when the application is not safety critical, but can still be mission critical. This resistance has multiple reasons. For instance, many methodologies [?,?,?], require other abstract models, such as the abstract user interface model, besides the task model. This makes it more difficult to integrate the approach in current development practices. Not only is it difficult to integrate the approach in current design and development practices, it also requires the development team to learn a set of new languages or notations.

[1] The original publication is available at `http://www.springerlink.com/content/m472874242t76m01/`

In this paper, we propose a methodology that takes the "one model, many interfaces" phrase introduced by Paternò [?] to the extreme. The methodology uses only one abstract model, the ConcurTaskTrees model, and integrates with user-centered software engineering practices. This model is not only used during analysis and design phases, but also during prototyping and even after product development to generate the navigation structure of the application's user interface. Another innovative use is that the temporal operators are also used during customization to avoid logical errors in the execution due to the removal of (seemingly) irrelevant tasks.

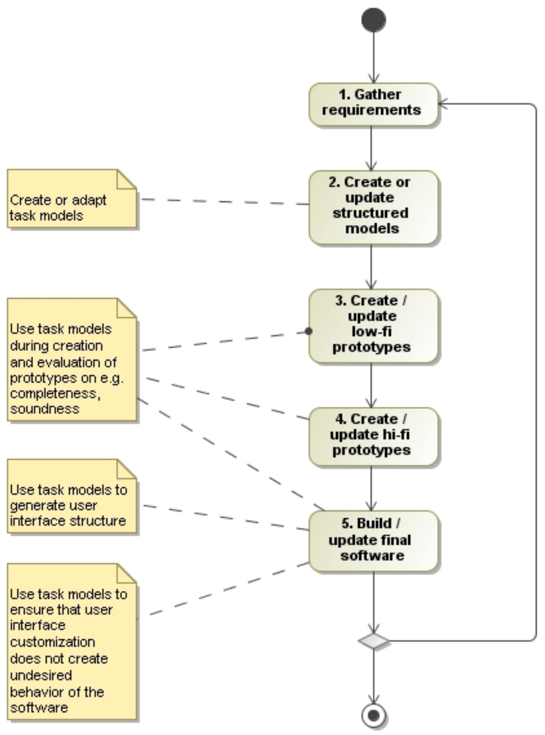
1. Gather requirements
Create or adapt task models
2. Create or update structured models
Use task models during creation and evaluation of prototypes on e.g. completeness, soundness
3. Create / update low-fi prototypes
4. Create / update hi-fi prototypes
Use task models to generate user interface structure
5. Build / update final software
Use task models to ensure that user interface customization does not create undesired behavior of the software

Fig. 1. A MuiCSer process instantiation and the envisioned use of the task model

2 A one-model user-centered process

We will use the MuiCSer process framework [?] as a basis for discussing how we envision the usage of the task model at different stages in a user-centered design and development process that supports user interface evolution. The different phases of the MuiCSer process instantiation are shown at the right side of Fig. **??**,

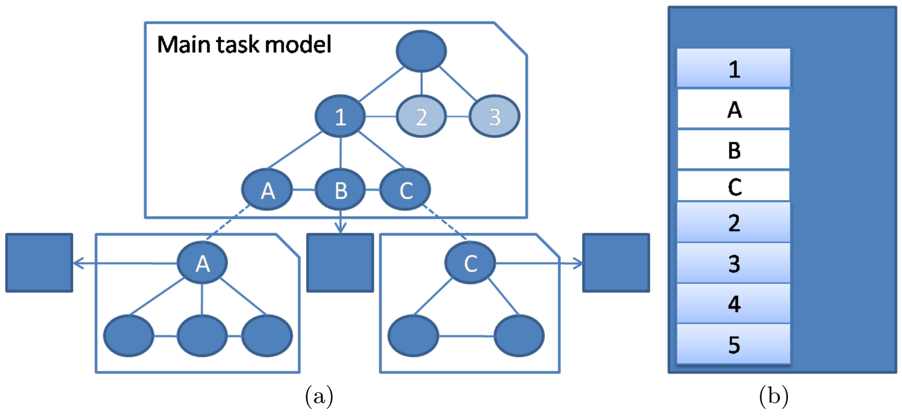

Fig. 2. (a) Task model distribution over files (pentagons) and links with software components (boxes) and, (b) the corresponding navigation structure

while the usage of the task model is shown in the form of notes at the left-hand side.

The task model is actively used for the first time in phase two, in which structured models are created or updated (analysis). This phase in the proposed process is supported by e.g. the Teresa tool [**?**]. The following two phases in the MuiCSer process framework deal with the creation of low-fidelity and high-fidelity prototypes. Using the methodology proposed in earlier work [**?**], these prototypes can be checked for completeness and soundness with the task model using a small tool. Many tools used for prototyping [**?**] offer the XML serialization required for this approach to work. In the fifth phase mentioned shown in Fig. **??**, the task model is used for two additional purposes in addition to checking for completeness and soundness as during the prototyping phases three and four. The task model is used to generate the navigation structure of the application and it is used to customize the user interface.

Fig. **??** shows a schematic overview of the task model and links it with the software components in our proof-of-concept implementation. There is a main task model file, which contains at least three layers, and possibly several separate files further detailing subtasks. Each child of the root task (1, 2, 3, . . .) represents a category in the navigation system of the application, while the root's grandchildren (tasks A, B and C) are used as the lowest level entries of the navigation system, as can be seen in Fig. **??**.

Tasks A, B and C in Fig. **??** are connected to the software components of which the system is comprised. Notice that these tasks can still have subtasks. This property is important as it will be used for the customization of the final product (during the last phase in Fig. **??**). The role of task models in the customization capabilities will be further detailed in the following section.

3 Customization through task models

Customization of the user interface, and especially the modification of the tasks that are available in a specific deployment of the application, is an important issue in complex industrial applications. The needs of the companies deploying these applications greatly differs and even within one company the needs can greatly differ depending on the person that is using the application.

As described in the previous section, the task model is the piece of information that links all different components of the user interface. If is therefore a logical choice to also use it to enable customization of the user interface and more specifically to remove or to disable parts of the user interface.

The hierarchical structure and temporal relations of a task model such as ConcurTaskTrees describe dependencies between task. One example of such a dependency occurs when information is exchanged between different tasks [**?**]. When such dependencies are present, it has of course a great impact on the capability to remove tasks (and thus parts of the user interface) from the system.

Table **??** shows the effect of removing one task on the other tasks in the task model for the different binary temporal operators. We used the following guideline within our customization algorithm: when a task or the correctness of the task model depends on the result, including the successful termination, of another task, the former task will be removed when the latter is removed. Based on this guideline the following factors were determined to cause a dependency between tasks:

information exchange When information exchange is explicitly modeled in the task model, we assume that the information exchanged by the tasks is crucial for a correct execution of the tasks and therefore they share a mutual dependency.

deactivation When a task deactivates another task, removing the latter task completely changes the meaning of the task model and may even cause an otherwise correct task model to never end. E.g. when the infinite iteration operator is applied to the deactivated task.

enabling When the task that precedes an enabling operator is removed, the task following the operator is also removed, because it has to much influence on the meaning of a task model. E.g. removing a login-task could result in unauthorized users having access to some functionality.

4 Discussion

This paper discussed a process that uses a single model throughout development: the task model, which is directly linked to parts of the user interface. This has the advantage that the traceability problem of requirements is minimized, which is considered to be an important property of a development process [**?**]. The process outlined in this paper has been used as part of a research and development project [**?**]. During this project the last two phases of the process

Operator		Removed Task	Effect
Name	Notation		
Choice	T1[]T2	T1 T2	T2 stays T1 stays
Concurrency with information exchange	T1 |[]| T2	T1 T2	T2 removed T1 removed
Independent Concurrency	T1 ||| T2	T1 T2	T2 stays T1 stays
Order Independence	T1 | = | T2	T1 T2	T2 stays T1 stays
Deactivation	T1 [> T2	T1 T2	T2 stays T1 removed
Suspend Resume	T1 | > T2	T1 T2	T2 stays T1 stays
Enabling	T1 >> T2	T1 T2	T2 removed T1 stays
Enabling with information exchange	T1 [] >> T2	T1 T2	T2 removed T1 stays

Table 1. Binary temporal operators of ConcurTaskTrees and effect when removing one of the involved tasks in our customization algorithm

have been carried out on prototypes instead of on a real system. Proof-of-concept implementations of the proposed tools have been realized and used during the project. This enabled us to do a preliminary evaluation of these tools during which we noticed that there are several enhancements that need to be made before these tools could be used in a real-life setting.

A first enhancement is better integration between the task model tool and the design tool. A designer should not be forced to modify XML by hand to link the task model to a user interface component. A plugin for the design environment can improve this situation.

A second concern is the customization of the software. Informal tests of the customization feature revealed that there is a definite need for the user to see what the impact will be of selecting/deselecting a task from the task model. Due to the dependency rules discussed in the previous section, removal of a single leaf-task can result in the removal of a significant part of the task model (and thus also the applications features in the user interface). The user should be informed about the consequences of his actions before he takes them. Some aspects of the work on feature models, such as that by Czarnecki et al. [**?**] may also be useful in realizing the user interface customization using task models.

Another point of future work, besides further validation and refinement of the approach, is to investigate whether the customization feature can be combined with the idea of executable task models proposed Klug and Kangasharju [**?**].

Finally, during the project mentioned before, we were constantly reminded that there should be more clear advantages of using task models before they will be actively used for large industrial applications. Transparent integration of additional model checking capabilities in task modeling tools could influence the decision to use a task-based approach such as the one proposed in this paper as part of regular industrial development. Further research regarding formalization of task models and related tool support therefore seems warranted. Also a direct relation between the task model and usability properties of the final user interface and automated checking thereof would be highly appreciated.

Acknowledgments

This research was performed in the context of the IWT project Warhol of Punch Graphix in cooperation with researchers of IBBT-CUO (KULeuven). Part of the research at the Expertise Centre for Digital Media is funded by the ERDF (European Regional Development Fund) and the Flemish Government.

References

1. Wolff, A., Forbrig, P.: Deriving user interfaces from task models. In: Proc. of MD-DAUI'09. Volume 439 of ceur-ws.org. (2009) `http://ceur--ws.org/Vol--439/paper8.pdf`
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers **15** (2003) 289–308
3. Mori, G., Paternò, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Transactions on Sofware Engineering **30** (2004) 507–520
4. Paternò, F., Santoro, C.: Integrating model checking and hci tools to help designers verify user interface properties. In: DSV-IS 2000. (2001) 135–150
5. Sottet, J.S., Ganneau, V., Calvary, G., Coutaz, J., Demeure, A., Favre, J.M., Demumieux, R.: Model-driven adaptation for plastic user interfaces. In Baranauskas, M.C.C., Palanque, P.A., Abascal, J., Barbosa, S.D.J., eds.: INTERACT (1). Volume 4662 of Lecture Notes in Computer Science., Springer (2007) 397–410
6. Paternò, F., Santoro, C.: One model, many interfaces. In Kolski, C., Vanderdonckt, J., eds.: CADUI, Kluwer (2002) 143–154
7. Haesen, M., Coninx, K., Van den Bergh, J., Luyten, K.: Muicser: A process framework for multi-disciplinary user-centred software engineering processes. In Forbrig, P., Paternò, F., eds.: TAMODIA/HCSE. Volume 5247 of Lecture Notes in Computer Science., Springer (2008) 150–165
8. Van den Bergh, J., Haesen, M., Luyten, K., Notelaers, S., Coninx, K.: Toward multi-disciplinary model-based (re)design of sustainable user interfaces. In Graham, T.C.N., Palanque, P.A., eds.: DSV-IS. Volume 5136 of Lecture Notes in Computer Science., Springer (2008) 161–166
9. Klug, T., Kangasharju, J.: Executable task models. In: Proceedings of TAMODIA, ACM New York, NY, USA (2005) 119–122
10. Sousa, K., Mendonca, H., Vanderdonckt, J.: User Interface Development Lifecycle for Business-Driven Enterprise Applications. In: Computer-Aided Design of User Interfaces VI. Springer (2009) 23–34
11. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice **10** (2005) 143–169