Abstract
User interface design patterns could increase the usability while reducing the software development effort. In this paper we explore design patterns for the derivation of the presentation from task and domain models.

Key words: user interface design, design patterns, HCI

1 Introduction
Little budgets and low usability awareness in organizations has always been an obstacle in promoting the effective use of user and task models. On another hand, the widespread use of software engineering methodology paying little attention to usability issues but providing with effective integration between subsequent design stages creates a strong appetite for productivity in the development process. Therefore designers rarely consent to pay an extra effort by using task models as driving specifications towards usable designs.

This paper aims at investigating how user interface design patterns could increase the usability while reducing the software development effort. The basic idea of our approach is to identify typical interaction structures in the task and domain models and to elaborate on a pattern language that captures the essential mappings between the various models that are used in the development process. A special emphasis is put on the relation between the task and presentation models.

2 Usability patterns and design patterns
Pattern languages are representations that focus on the interaction between physical form and social behaviour and express design solutions in an understandable and generalizable form [2]. The original model for patterns was the work of Christopher Alexander and his colleagues [1] who applied pattern languages in architecture and urban design. The emphasis was put on the characteristics of the physical form of the built environment, which can afford certain social behaviour.
There are several ways of using patterns. The report of Bayle et al [2] enumerate five of them: capture and description of key characteristics in a context sensitive way, generalization across places and situations, prescriptive solutions to commonly encountered problems (guidelines), rhetorical use among members of the design community and predictive reflection on the impact of changes.

Patterns are also used in software engineering community as means to cope with the complexity and diversity of the development process. In this respect they show a concern for more adaptive and pragmatic design, able to satisfy specific requirements while preserving software re-use and thus reducing the development effort.

A similar requirement could be found in user interface development. Design of user interfaces is constrained by several models: user model, task model, application-domain model and platform model. First generation of model-based approaches focused on domain models that were exploited to derive appropriate interaction objects.

Like the patterns of Alexander, UID patterns could help in deriving characteristics of the presentation environment, which can afford certain cognitive behavior. Using the information contained in the domain model helps in deriving what information the user has to manipulate in order to perform his task. Using the task model makes it also possible to derive how to present and organize this information in a usable way.

3 Exploiting the task model
In a previous work [1] we proposed a layered task model as a foundation for task based design. Task models are starting with a functional, device independent description of the user’s task that shows task decomposition, temporal ordering between tasks and additional constraints such as pre- and post-conditions. Further transformations of the task model produce a device-dependent task model, which shows how the user is actually interacting with a given interface.

The functional task decomposition stops at unit task level. Unit tasks have been defined as the lowest level task the user really wants to perform [3]. From the point of view of task analysis, unit tasks are showing “what-to-do” knowledge, regardless how this work will be actually carried on. Since the task will be accomplished using a given technology, the representation is said to be device independent.

Next step is the operationalisation of unit tasks up to the level of basic tasks. This task decomposition requires considering the technological means by which the user will accomplish his goals.

A basic task is defined as the lowest level task that is using a single interaction object or a single external objects or serves a communicational goal. In this respect, the stopping criteria for task decomposition are an interaction object (for interaction
tasks, application tasks and cognitive user tasks), an external object (for manual user
tasks), and a distinct step in communication (for communication user tasks). Decomposition of basic tasks up to action level is beyond our task-based approach.

4 Design patterns for user interface derivation

Interaction objects are embodying basic interaction techniques. They are covering both presentation and dialog model at a basic level, which we may term as lexical level. In this respect, interaction objects are the basic constituencies of the user interface.

Interaction object groups, which have one or more information, control AIO and one function control AIO (sometimes two, but the user could choose only one of them at a given time – for example OK vs. Cancel) provide with a first level of structuring the interface. Interaction object groups could be used as basic building blocks for the presentation model in a task-based approach.

An example is given in Figure 1 where searching either by identification number or by name is used to identify an old client. The task model shows the actual behaviour of the user: if the client does not know his id or if the name is misspelled, then a new client will be recorded.

We can further decompose the tasks in Figure 1 using CTT notation [4] like in Figure 2, in order to obtain the full operational structure for the task “identify-by-
We can put it all together and propose a design pattern. In order to propose a pattern language we will start by describing the following: problem, context, forces, solutions and comment.

Problem
Functions could be chained such as the same data is used as prompting for future actions and feedback for previous actions. Ideally, the user should be provided with semantic feedback showing the effect of his actions to the application data.

Context
The same operational structure could be used in several sub-tasks. It usually happens when there are several functions performed upon the same object. For example, the client address is used as feedback after two search methods and for data entry for a new client or when the address changes.

The user is performing a search function. If successful, the attribute data of the found object is displayed in the interface. Then the user could act upon the displayed data in order to perform further actions. This situation is typical in database applications when we first search for a record and then edit it.

Forces
There are several search keys. For example, the user can search using a client id, a personal id number or the name. Some search methods could be faster other could be more easy to use. If a method fails, possible because of a data entry error, the user might want to try another.

There are a huge number of possible groupings of AIOs in the interface. Grouping of interaction objects could be done according to semantic criteria provided by the data model or in a task based approach. Semantic criteria help to perceive the data structure including relationships. A task-based approach minimizes user actions.

There is always a tradeoff between the information density and the articulator tasks for navigating between different dialog units.

Solution
First level of AIO grouping should mirror the operational task structure. Assign a static interaction object denoting the semantics of data or function to each AIO. This design step is performed in a bottom-up approach. Higher-level groups are based on the goal hierarchy. Assign a static interaction object denoting the task goal to each AIO group. Use up to three levels of grouping in a dialog unit. Allocation of dialog units should be done in a top-down approach based on the task model.

Comment
This pattern applies mainly for the presentation part of the interface and helps in organizing the information on the screen in a way that provides with user guidance. It can be integrated in more complex patterns.

5 Conclusion
In this paper we investigated design patterns, which are based on the information provided by task and domain models. On the basis of our previous work and this investigation we propose a method to develop pattern languages in two steps:
- identifying patterns by using information from task and domain models and recording them in an initial pattern language combining formal definition with informal description;
- formalization of the pattern language by expanding the initial pattern template.

References
4. Paternò, F. Model-based design and evaluation of interactive applications. Springer Verlag, 1999