Selective Modeling to Support Task Migratability of Interactive Artifacts

Anke Dittmar and Peter Forbrig

University of Rostock, Department of Computer Science,
Albert-Einstein-Str. 21
18051 Rostock, Germany
{anke.dittmar,peter.forbrig}@uni-rostock.de

Abstract. Selective modeling is suggested as a technique that encourages designers to mix exploratory, analytical, and empirical design activities in interaction design. The co-development of models and prototypes of interactive systems is proposed to support a better balance between formal and explorative design approaches. Models serve to inform design decisions but also to analyze emerging alternatives of prototypical implementations.

Task migratability is a usability design principle that describes how control for task execution is transferred between system and user. Refined flexible task allocation is rarely achievable through pure top-down decomposition as used in many model-based approaches. The paper shows at the example of HOPS models how selective modeling can be applied to develop prototypes in a deliberated evolutionary way by using models to express different viewpoints and to explore design options at different levels of granularity.

Keywords: User-Centered Design, Model-Based Design of Interactive Systems, Exploratory Design, Tools for Design, Modeling, Prototyping, Design Rationale.

1 Introduction

Task migratability describes the ability of an interactive application to pass control for the execution of a task so that it becomes either internalized by the user or the application or shared between them [1]. The application of this usability principle to the design of interactive systems promotes the implementation of dynamic function allocation. Safety-critical systems such as the control of aircrafts, ships or chemical plants are complex and well-accepted examples for a need of dynamic function allocation. Throughout this paper a simple and mundane example is used for illustration. It allows for discussion without requiring specific domain knowledge.

In order to shape an interactive artifact that supports task migration, designers[1] must develop a clear understanding about current tasks, artifacts, habits, and situations. They must create hypotheses about appropriate grades of automation in possible future situations and how to support them by corresponding user interfaces.

[1] In this paper, the term designer refers to all active stakeholders in the design process.

P. Campos et al. (Eds.): INTERACT 2011, Part III, LNCS 6948, pp. 571–588, 2011.

They also need to understand the nature of transition processes: why and how do people change their current practices? One can consider task migratability as a principle that encourages designers to develop a differentiated picture of human activity and the role of artifacts. Of course, this has to be reflected in the design processes themselves. Task migratability is neither well supported by analytical approaches favoring a top-down refinement of models to implementations nor by approaches focusing mainly on the application of empirical means.

The paper suggests *selective modeling* as a technique to intertwine exploratory, analytical, and empirical design activities in a more effective way. This technique can be considered as a deliberate change to assumptions and practices in model-based design to support the integration with other user-centered design approaches.

In selective modeling, designers co-develop prototypical implementations of the interactive artifact and models describing different perspectives on it. For example, envisaged task models help to derive requirements of the interactive system under consideration. Or, exploratory prototypes are built and then empirically tested or analyzed in more depth by the creation of models. In the suggested approach, models and implementations are considered as fragmentary and "open" design representations which are used to shape interactive artifacts. Each representation evokes certain thoughts and actions. Their co-development helps designers to switch between different perspectives and to explore design alternatives in different contexts. It is acknowledged that design activities in a user-centered design process can happen in any order but must be linked by evaluation steps (e.g. [2]).

In particular, the paper investigates how the co-development of models and prototypes supports flexible task allocation. Models and prototypes have specific characteristics and serve different purposes in the suggested approach.

- Models describe selected aspects of current and envisaged tasks, artifacts and roles.
- Models inform the development of prototypes and other models.
- Models serve to analyze prototypes.
- Prototypes are implementations of design ideas and can be tested for utility, usability and user experience.
- Prototypes (and their use) inform the development of models to get a richer task understanding.
- Prototypes and models are deliberately underspecified in their behavior. They are supplemented by other design representations to explore alternative design options.

The general approach is "instantiated" in the paper by the use of Higher-Order Processes Specifications (HOPS), a universal specification formalism with high-level concepts for describing interaction from different viewpoints [3,4]. Tool support makes it possible to animate HOPS models and to map them to Java implementations. Selective modeling is supported by techniques such as model coupling, model-guided prototyping and deliberated underspecification. Throughout the paper the mastermind game serves as example to illustrate suggested ideas.

The paper is structured as follows. In the next section, task migratability is considered in depth and examples are given. Sect. 3 discusses different design approaches in HCI and their underlying understandings of tasks and human activity in

general. Selective modeling is introduced in Sect. 4. It applies a more liberal task understanding on design activities themselves and suggests adaptations to current model-based design practices because flexible task allocation in systems is rarely achievable by pure top-down refinements of task models to implementations. Sect. 5 instantiates the general approach. The HOPS formalism and tool support is applied to the example problem to demonstrate a co-evolution of models and prototypes in order to explore task control options at different levels of granularity. The paper closes with a summary and future work.

2 Task Migratability

Although almost a truism, proper task allocation is still one of the most important aspects of human-computer interaction. Task migratability is a usability principle that supports dynamic task allocation. It "concerns the transfer of control for execution of task between system and user. It should be possible for the user or system to pass control of a task over to the other or promote the task from a completely internalized one to a shared and cooperative venture" [1]. Systems that allow dynamic allocation of tasks make possible more adequate responses to actual conditions. On the one hand, task migratability is based on *task conformance* describing the degree to which a system covers the users' tasks and represents their task understanding [1]. On the other hand, designers need to acquire a deeper understanding of what the tasks of the users might be in certain situations and how to support their achievement.

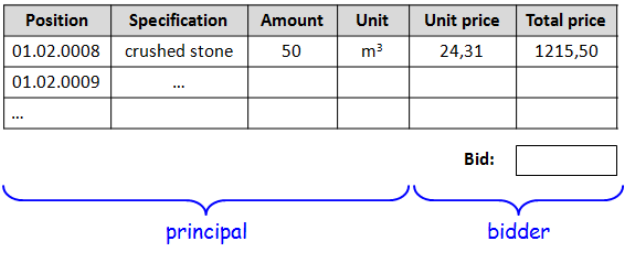

Fig. 1. A tender specifications created by the principal and priced by a bidder

User interfaces of applications that offer different task allocation options (configurations) are typically more complex. This not only concerns the representation of the functionality of each configuration and the representation of relevant information for the users to achieve their actual goals. It also concerns the representation of the actual and of possible configurations and the design of control transfer.

Sometimes a poor representation of a certain configuration may easily cause problems. Sometimes, subtle modifications to the user interface may already be sufficient to support different configurations. For example, an interactive application for managing tender specifications as depicted in Fig. 1 may support different modes for calculating bids. In one mode the application calculates the sum of the total prices

of each position in the specifications automatically. In another mode the sum is calculated as well but can be modified by the user. In the first mode the bid entry may be implemented as output field, in the second as input/output field.

2.1 Current and Envisaged Worlds

"Design requires two models of the world: a current one and a future one. Design is a goal-directed activity involving deliberated changes intended to improve the current world, so the need to model the future in design is unquestionable. In practice, models of possible future worlds need to be based on models of the current world" [5].

No matter how simple design problems may appear at first glance, a deeper analysis of current practices may reveal more subtle issues. To illustrate this point, let us go back to the example of tender specifications as they are e.g. used in the building sector (see Fig. 1). In order to find a builder, a principal creates the specifications for the project. Bidders fill in unit prices, total prices and the bid. Then, the principal organizes a submission where the bids are made public to all bidders. Afterwards he checks the details of each bid for calculation errors and makes a decision. Let us assume that a computer artifact has to replace paper work. It seems to be obvious that the calculation of the total prices and the bid should be done by a computer. However, it was not uncommon that bidders miscalculate with intention in order to cover their real bids in the public submission. Against this background the design problem appears in a different light. Even if not all will be implemented, different task allocation options should be considered at least. A design process which is guided by the principle of task migratability increases the sensitivity of the designers to possible effects of allocation decisions at different levels of granularity on the resulting interactive artifact (in particular, on its user interface) and on the behavior of the users.

Dearden et al. [6] point out that allocation is not a 'zero-sum game'. It rather should be considered as a transformation of work that the human must perform and that does not necessarily result in a reduction of workload. Unintended, in some cases even undesired practices can emerge. An example is given in [7] where a captain of a modern airliner is cited with the words "You never know exactly what will be the result of flipping a switch or setting in a new parameter, so we don't interact with [the automation] during automatic landing. We simply don't know what it will do."

Fortunately, it is not possible to fully predict the future and sometimes concerns about design effects are overdone. For example, the introduction of software systems to manage tender specifications may result in a decline of intended miscalculations by bidders regardless whether these programs allow a manipulation of bids or not. However, this does not release designers from their responsibility for future practices.

It can be regarded as common ground in user-centered design that the development of interactive artifacts has to be embedded in processes of understanding current working practices and envisaging/designing future practices and that such processes are supported by the creation and use of external representations. However, Sect. 3 will show that design approaches can differ in their underlying assumptions about tasks and human activity, about models and how to use them, about modeling techniques and their combination with other design techniques.

2.2 The Example Problem

Fig. 2. a) Mastermind as board game with the hidden code in front of the picture, b) codebreaker making a guess, c) unintended use of the board

Mastermind is a logic game for two players in the roles of the codemaker and the codebreaker. At the beginning the codemaker sets a code (a hidden combination of colored pegs) which the codebreaker has to discover. Each guess of the codebreaker is marked by the codemaker. Usually, a black marking peg is used to indicate that a peg has the right color and the right position, a white peg tells the codebreaker that one peg is of a color which is in the solution, but it is in the wrong position. The game board in Fig. 2 gives the codebreaker a maximum of seven trials to solve the puzzle.

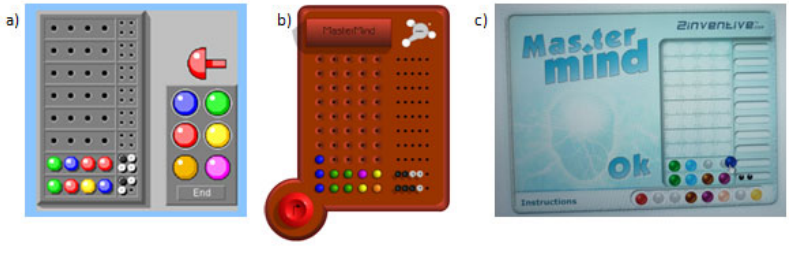

Fig. 3. Online versions of the game[2]

Mastermind is a simple but still popular game. There are a number of online versions available (see Fig. 3). In most of them, the computer acts as codemaker and the user as codebreaker. The example problem we will use throughout the paper is about how to support flexible task allocation in an interactive version of the game. It is used to play with the idea of selective modeling as a technique to combine different means such as fragmentary models and prototypical implementations in a deliberate way in order to get a refined and situated understanding about work and to shape interactive artifacts accordingly. It is also used in the next section to illustrate common ideas in HCI which form the basis for the suggested approach.

[2] http://rezepte.nit.at/online_spiele/mastermind/mastermind.html
http://www.mathsisfun.com/games/mastermind-game.html
http://www.gamesbasis.com/mastermind.html (last accessed January 2011)

3 Roots of Selective Modeling

Typical descriptions about the "history" of HCI describe the 1980s and early 1990s as a formative period with emphasis on cognitive aspects of single user's tasks. Since the 1990s, the focus has changed to supporting cooperation and communication between users (e.g. [8]). This shift has been accompanied by changes in the understanding of goal-directed human activity. In the following, task-based and resource-based conceptions of human-computer interaction are reviewed.

3.1 Task-Based Conceptions

In task-based approaches to HCI, tasks are the unit of analysis and design to describe work in current and envisaged work systems. A task is seen as a mechanism by which intended changes in application domains are achieved. It is assumed that people possess and recruit knowledge about tasks they have previously learned and performed in a given domain. It is furthermore assumed that task knowledge can be analyzed, modeled and predicted [9]. Cognitive approaches such as HTA [10], GOMS [11] or TKS [9] describe task knowledge in terms of hierarchical task decomposition, goals, actions, plans, task domain objects etc. Task analysis refers to the process of identifying and examining the tasks that must be performed by users when they interact with systems. It is used in HCI to provide an idealized, normative model of tasks that should be supported by computer artifacts. In task design, models are used to develop hypotheses of future tasks and their execution.

To illustrate common ways to notate and use task models, Fig. 4 shows a simple analysis model of making a guess in mastermind. The task is hierarchically decomposed and temporal relations are added to describe in which order sub-tasks have to be executed. Let us assume that the model is the result of observing some games that were played with the board of Fig. 1. The codebreakers placed pegs on the board but sometimes removed them again. The codemakers were observed to set marking pegs only. The example may be sufficient to point out that analysis models can only describe small parts of behavior [5]. The presented view depends on who produced the model and for what purposes. For example, the model in Fig. 4 could have been produced to inform the design of an interactive version and the designers may have found it sufficient to look at observable behavior. The gray areas in the figure indicate possible design decisions. First, action sequences <Remove peg, Set peg> as observed in the current task situation are replaced by the operation 'Replace peg' (as e.g. in the version of Fig. 3b). Second, the computer takes the role of the codemaker. However, the task model in Fig. 4 could also be useful for evaluation. If it is used to reflect available user actions in the game version of Fig. 3a) it can reveal that users are not supported in removing pegs from the board.

3.2 Resource-Based Conceptions

Above mentioned approaches put much emphasis on mental representations that people develop and employ to achieve desired states in the world. However, the world should not be seen as "a largely stable collection of objects and events to be observed and manipulated according to the internal mental states of the individual" [12].

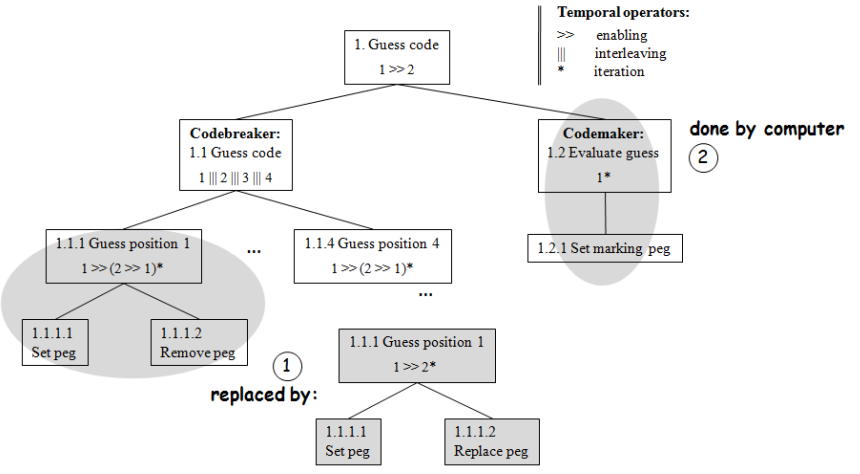

Fig. 4. A hierarchical task model describing observed behavior for a guess in mastermind. Gray areas indicate the transformation to a design model.

Resource-based conceptions of human activity such as 'situated actions' [13], Distributed Cognition [14], or Activity Theory [15] give far more attention to the interplay between actors and artifacts. It is assumed that the resources which are available in an actual situation shape the actors' behaviors. People experience their environment as *activity spaces* and change them deliberately according to their goals. The deeper their understanding of the environment the more fine-tuned are their interventions in order to trigger certain behaviors. As a consequence, plans (e.g. task models) are seen as one resource among others. They can guide but not fully control behavior.

Analysis and design activities that are guided by this attitude acknowledge the deep interplay between internal and external artifacts and the variety of 'control flows'. Of course, this also supports a deeper reflection of task migratability of interactive systems. To give an example, let us look back at Fig. 2. In picture b), the codebreaker uses his fingers to mark some pegs in order to make the next guess. Although the mastermind game was introduced as a logical game in Sect. 2.2 (and so it is perceived by many people indeed[3]), players use besides cognitive skills a variety of external means in order to succeed. They position pegs with certain colors near the board. Sometimes two or more people are the codebreaker and they exchange their thoughts about the right code. When the codebreaker is a novice to the game, the codemaker may help and so on. An interactive mastermind game could enable a single person to play or could allow people in different locations to play with each other. However, designers should also think about the fine-grained details in order to support different usage situations? In [16] human-computer interaction is analyzed as distributed cognition and six different types of information structures are identified that interactive artifacts can represent to users as possible resources for action: plans, goals, possibilities, history, action-effect relations, and states.

[3] On the webpage of the version depicted in Fig. 3b): "Try different color combinations and use your brain to figure it out".

Resource-based approaches give more plausible explanations of effects such as appropriation and unintended use of artifacts because they take a more liberal view on goal-directed behavior than task-based conceptions. The latter term refers to situations where people use artifacts in ways which were not intended by the designers. An example is given in Fig. 2c). The two years old child has no idea about the goal of the game but tries to place pegs on the holes on the board at all. It is more a matter of dexterity. What one can (re-)learn from the child is that there is no need not to use holes in 'free rows' on the board. The analysis of phenomena such unintended use can help to broaden the view on current activity spaces and prevent designers from introducing unnecessary constraints on future activity spaces. As it turned out in the analysis of the mastermind game there are in fact people who use free rows as a resource for holding pegs which they believe are part of the solution. Many available interactive versions such as those in Fig. 3 do not give the users (codebreakers) access to free rows. A more fine-grained transfer of control would be more convenient. Allocation of function is considered as a resource allocation problem in [6].

3.3 Related User-Centered Design Approaches

In User-Centered Design (UCD), many approaches follow a more task-based conception of work. For example, formal task models and top-down refinement are used in most model-based design approaches to derive user interface specifications and prototypes by applying techniques from software engineering [17, 18]. In addition, there is often no distinction between task analysis models and task design models. Consequently, modeling activities easily become specification-driven as we have shown in [3]. In contrast, Scenario-Based Design [19] or Contextual Design [20] encourage designers to ground their design in discussions of different, even conflicting perspectives on current working practices. These approaches rather support resource-based ideas.

Design approaches can furthermore differ in their acceptance and use of formal/semi-formal/informal descriptions and in their assumptions about completeness and correctness of models. They favor analytical over empirical techniques or vice versa. They support design rationale or not, and generally, put different emphasis on different types of design activities. For example, in iterative design empirical techniques such as prototyping, usability tests and iterative development in the field are preferred because it is assumed that the effects of most design choices and emerging artifacts cannot be predicted well enough by analytical means [21,22].

HCI is described as multidisciplinary but also fragmented field [8]. Of course, this fragmentation must have effects on researchers and practitioners. On the one hand, isolation "from some portion of the field's foundations" is often to be found among them as a coping strategy [8]. On the other hand, the need for amalgamating different UCD methods and techniques is well-accepted. To give just two examples: in a warning about iterative design in [1] it is written that "the ideal model of iterative design, in which a rapid prototype is designed, evaluated and modified until the best possible design is achieved…is appealing" but it is also important to be able to overcome bad initial design decisions or to understand the reasons behind usability problems and not just detect the symptoms. It is recommended to use iterative design "in conjunction with other, more principled approaches to interactive system design."

Fischer et al. [23] point out that in Design Rationale "argumentation has been considered in isolation from the activity of solution construction" but that construction and argumentation have to be unified.

Although UCD practices have been refined over the years (e.g. [24]) the question of how to feed back results of evaluation steps into design in an effective way still remains. In this paper, we explore how deliberate changes to assumptions and practices in model-based design support a move towards a resourced-based conception of human-computer interaction and the integration with other empirical and analytic design practices in order to develop more flexible interactive systems.

4 Selective Modeling – Main Ideas

We suggest the term *selective modeling* to describe design practices where designers use models as *resources* for developing interactive artifacts. In other words, we see the design process itself through the glasses of a resourced-based conception of work. Designers deliberately shape their view on the design space by creating different types of representations about tasks, actors or artifacts. They help to develop an understanding about current practices and about possible future usage situations of the system under design that emerges as a "side effect" of the co-evolution of models.

If we apply this view on model-based design the overly dominant role of (envisaged) task models on the creation of systems specifications and prototypes is challenged. Additionally, the understanding of models as selective and fragmentary descriptions of phenomena from certain points of view is more emphasized. Task models describe how to act in the world to achieve a certain goal. Artifact models describe domain objects in terms of attributes and actions serving different purposes in different contexts. Dialog models, in particular, may describe possible usage scenarios of interactive artifacts in terms of visible information and enabled action sequences. A prototype represents concrete human-computer interactions. It is represented by code in a programming language and is experienced in a different way than abstract models. Every representation can be considered as a design artifact which evokes certain responses.

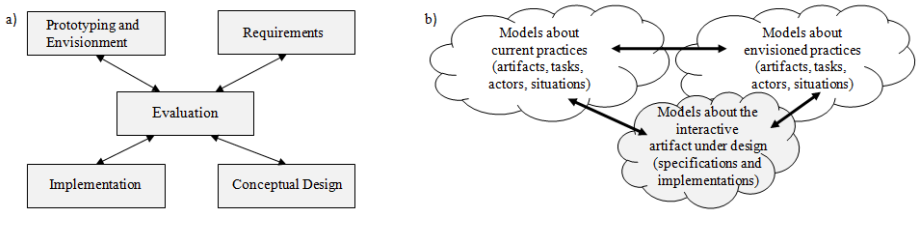

Fig. 5. a) Modified Star Life Cycle [2], b) co-development of design representations

While diagrams such as the one depicted in Fig. 5a) illustrate the interleaving of design activities and the central role of empirical and analytic evaluation steps in UCD, Fig. 5b) focuses on the co-evolution of design artifacts that must be achieved.

The selection and use of models by the designers depend on the objectives and the constraints of the actual design situation. For example, models of tasks, artifacts and roles can be coupled with dialog models in order to explore resource allocation options and to refine the dialog models. A formal systems specification can be evaluated by a corresponding prototype. Models can be used or developed for the assessment of emerging implementations. Changes to these models then can feed back into later design steps and so on. Selective modeling supports the intertwinement of different design activities by encouraging designers to use representations in a focused way.

- Design representations (e.g. formal models and prototypes) describe selected aspects of the design problem from different perspectives and at different levels of granularity.
- Representations are deliberately underspecified in order to avoid premature design commitments, e.g. a premature allocation of resources.
- Different representations are partially coupled to explore design alternatives, to inform design decisions and to enable testing and reflection of implementations. Such couplings drive the co-evolution of representations.

The next section explores these ideas further by using the HOPS formalism.

5 Selective Modeling with HOPS

HOPS (Higher-Order Processes Specification) is a universal specification formalism with tool support for describing interaction from different viewpoints and at different levels of abstraction. A short introduction to HOPS is given in Sect. 5.1. For more details see [3,4,25]. Sect. 5.2 explains how HOPS supports selective modeling techniques. In particular, model coupling, model-guided prototyping and deliberated underspecification are demonstrated at the example of designing an interactive mastermind game.

5.1 Introduction to HOPS

In HOPS, systems are considered as compositions of interacting sub-systems. They are specified by *processes*. The structure of a process is determined by components, operations and sub-processes. *Operations* refer to the smallest units of behavior that are of interest in the actual modeling context. The behavior of a process is defined by a set of sequences of operations. *Sub-processes* of a process P refer to partial behaviors of P. They are useful for creating structures (e.g. task hierarchies) or behavioral variants. Sub-processes can also be used to specify states of components. *Components* as instances of previously defined processes describe sub-systems. Processes without components are called *basic processes*. Their operations are defined by names only. Processes with components are called *higher-order processes*. An operation of a higher-order process describes either new emerging behavior or it is an abstraction of a sequence of operations of components. By using higher-level operations and sub-processes, a process can partially control the otherwise independent behavior of its components.

Basic Processes and Process Animation. The basic process *Guess_peg* given below is defined by three operations (lines 3-5) and three sub-processes *Option1-3*. Each sub-process describes a variant for placing a peg in the mastermind game. The first two options correspond to the variants in Fig.4.

```
 1  PROCESS Guess_peg
 2  OPS
 3    setPeg(s: string),
 4    removePeg,
 5    replacePeg(s: string),
 6  SUB PROCESSES
 7    Option1 IS setPeg(?) ; (removePeg ; setPeg(?))*,
 8    Option2 IS setPeg(?) ; replacePeg(?)*,
 9    Option3 IS setPeg(?); ((removePeg ; setPeg(?)) [] replacePeg(?))*,
10  END PROCESS
```

The behavior of (sub-)processes is specified by partial equations. Temporal operators - as known e.g. from CCT [17] - are used to describe constraints on operation sequences (lines 7-9).

Temporal operators in HOPS:				
	;	sequence	#n	n-fold iteration
	[]	alternative	[…]	option
	|||	concurrency	|>	disabling
	*/+	iteration	[>	interruption

The HOPS tool offers interactive model animation. At each step of an animation run, the user can choose to execute one of the enabled HOPS operations (presented by buttons in Fig. 6 that are attached to the actual state/node in the animation tree). Each path of the animation tree represents the prefix of a valid operation sequence of the model. The user can "jump" between nodes of the tree to further explore specified behavior.

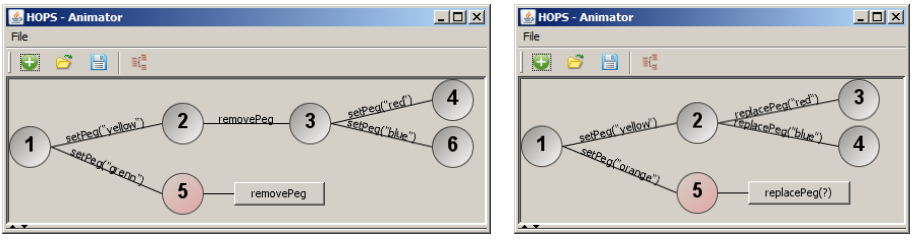

Fig. 6. Interactive animation trees for sub-processes *Option1* and *Option2* of *Guess_peg*

Mapping between HOPS Models and Object-Oriented Implementations. The HOPS tool enables an automated mapping of HOPS process instances to Java objects. If an operation is executed during model animation the corresponding Java method calls are executed as well (see Fig. 7). In the example given below, instances of process *Peg_hole* are linked to Java objects of class *PegHole*, HOPS operation *init* is mapped to the constructor method, operation *setState* to method *setState* and so on. The process describes peg holes on a mastermind board. It has a local variable (line 2) to hold the state: holes are empty or contain a peg of a specific color.

```
1  PROCESS Peg_hole
2  VAR state: string,
3  OPS
4    init IS objId.fCall(new,[this],["PegHole","empty"]),
5    setState(s: string) IS void.fCall(setState,[this],[s]),
6    getState IS state.fCall(getState,[this],[ ]),
7    quit,
8  SUB PROCESSES
9    Peg_hole IS init ; (setState(?) [] getState)* [> quit,
10 END PROCESS
```

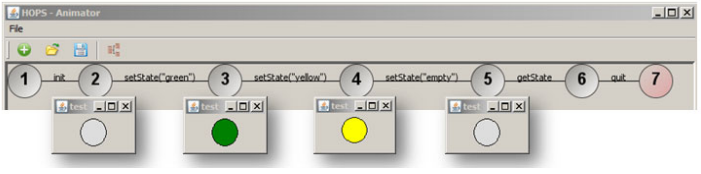

Fig. 7. The animation of operation sequence ⟨init, setState("green"), setState("yellow"), setState("empty"), getState, quit⟩ controls the associated Java implementation of a peg hole

Higher-Order Processes contain components that are themselves process instances. In Fig. 8, process *Peg_dialog* contains two components: *sm* is an instance of process *Peg_hole* - embedded in a test frame, *tm* is an instance of *Guess_peg* that behaves like its sub-process *Option2* (encircled 3 in Fig. 8). The bottom panel of the animator tool in the figure visualizes the hierarchical component structure (component tree). Sub-processes and operations of higher-order processes describe the interaction of their components. Higher-level operations describe new emerging behavior or conflate sequences of components' operations into new atomic behavioral units to increase the level of abstraction in a description. Operation *replacePeg* in Fig. 8 (encircled 1) is e.g. an abstraction of sequence *⟨tm.replacePeg, sm.setState⟩*.

The HOPS notation makes it possible to describe a system from different viewpoints and to couple these views by using higher-order processes. In Fig. 8, component *tm* reflects a task perspective while component *sm* describes a part of an artifact. HOPS supports both bottom-up as well as top-down thinking. On the one hand, the behavior of a process is determined by its components. On the other hand, it has partial control over the components. Partly redundant models can be used to enable description of emerging constraints and of distributed control (see e.g. [4]). For example, higher-level operation *replacePeg* in Fig. 8 is only enabled in an animation run if operations *replacePeg* and *setState* are enabled in the components *tm* and *sm*. Additionally, higher-order processes focus on and control only some operations of their components. The control of operation *getState* remains, for example, in component *sm*. This is also to be seen in the bottom left part of Fig. 8 (encircled 2) where *getState* is enabled in node *sm* of the component tree. In other words, the behavior of a higher-order process P is defined by a set of sequences of operations which are either defined by P itself or which are defined by components but not controlled by P.

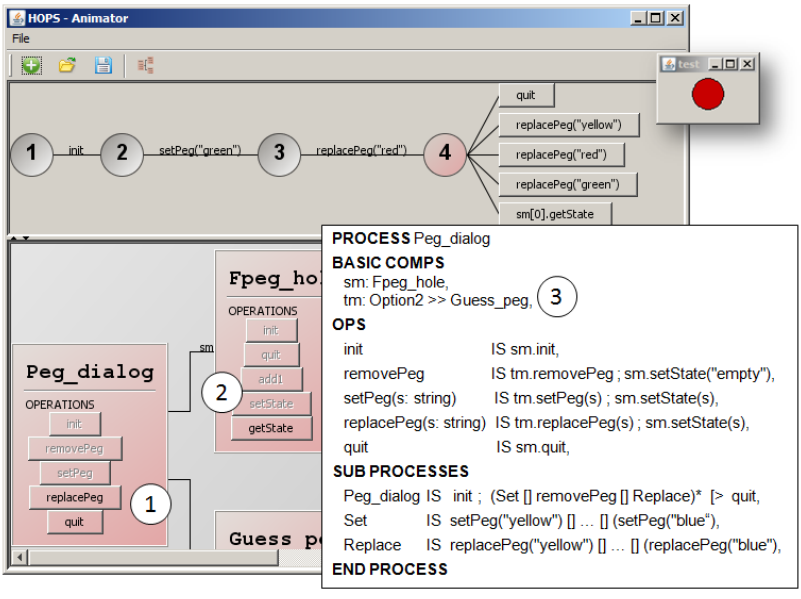

Fig. 8. Specification of higher-order process *Peg_dialog* (bottom right) and screenshot of the HOPS-animator after animating sequence ⟨init, setPeg("green"), replacePeg("red")⟩

5.2 Selective Modeling Techniques Supported by HOPS

Model coupling, model-guided prototyping and deliberated underspecification are selective modeling techniques that support resource-based design ideas, the application of analytic and empirical means, the co-evolution of design representations and the exploration of design options. Higher-order processes implement model coupling. Mappings between HOPS processes and Java classes as explained in the previous section can be used for model-guided prototyping. Deliberated underspecification is facilitated by partly redundant descriptions with distributed control in HOPS. The ideas are illustrated by considering aspects the example problem at different levels of granularity: the design of an interactive peg hole, of an interactive peg row, and of the whole mastermind board.

Peg Holes – Exploration of Design Options by Model Coupling. Higher-order process *Peg_dialog* in Fig. 8 couples a task model describing how to place a peg and a model of an interactive peg hole with few constraints on its behavior. *Peg_dialog* integrates both views but describes less constrained behavior than the task model component *tm* itself. Hence, *tm* fully controls the use of the peg hole (component *sm*): a peg can be set, and then it can be replaced arbitrary often. However, two other variants of a task model were given in the definition of process *Guess_peg* (subprocesses *Option1*, *Option3*). Fig. 9 illustrates effects on the behavior of *Peg_dialog* if the behavior of component *tm* is switched to these options. In this way, different alternatives for refining the behavior of interactive peg holes can be explored.

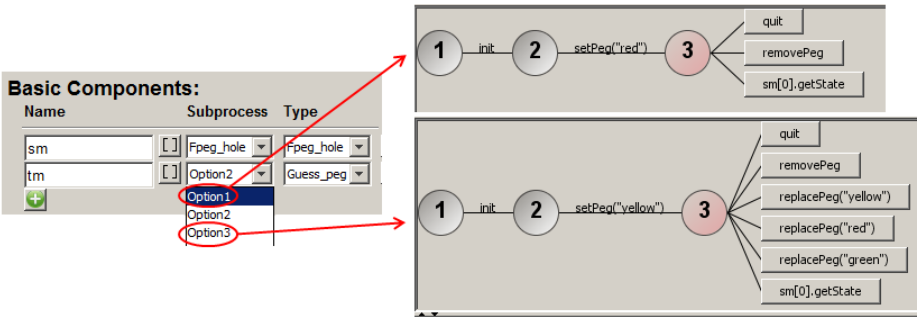

Fig. 9. Different options of the task model are coupled with the underspecified model of an interactive peg hole to explore possible refinements of its behavior in animation runs

A selected integration of different, but overlapping views on the design space supports resource-based thinking rather than task-based thinking because task models do not play the dominant role anymore. The focus of attention is reflected in the root process of the actual process composition. Its type is not restricted to task models. In a further step it is shown how prototypes and models about roles, artifacts and tasks can be used to explore different resource allocations.

Peg Rows - Model-Guided Prototyping. Model-guided prototyping in HOPS is a technique where Java implementations and HOPS models are loosely mapped to each other. Models describe only those aspects that are the actual focus of analysis. During animation they partially control the prototypes. This provides an analytical means for designers while, at the same time, prototypes can be tested empirically to a certain extent. In the example in Fig. 10, a prototypical implementation of rows of peg holes is given. In this case, the implementation of peg holes was refined according to *Option3* in Fig. 9. Users can replace pegs but they can also remove pegs from holes. This is very important for the experience of the game. The basic HOPS process does not model single peg movements anymore but is focused on enabling/disabling of a row and on setting/getting the whole code. Otherwise, users can interact directly with the prototype as indicated in the bottom of the figure.

Mastermind Board - Exploration of Resource Allocations by Deliberated Underspecifications. A co-evolution of representations is assumed to better ground design ideas in existing practices. It can drive the production of systems specifications or prototypes but it can also "slow down" the design process by re-considering current activities and describing them in more depth. It was shown in Sect. 3 that task migratability can only be achieved in systems if designers acquire a deep understanding about possible usage situations and how they could be resourced by artifacts. Task-based design approaches may easily result in artifacts that unnecessarily constrain the users' activities. For example, an interactive mastermind board may be too restrictive if the peg holes of free rows cannot be accessed or if the computer immediately starts to mark the codebreakers' guesses after they placed the last peg. In [6], (Dynamic) Task allocation is considered as task transformation and as

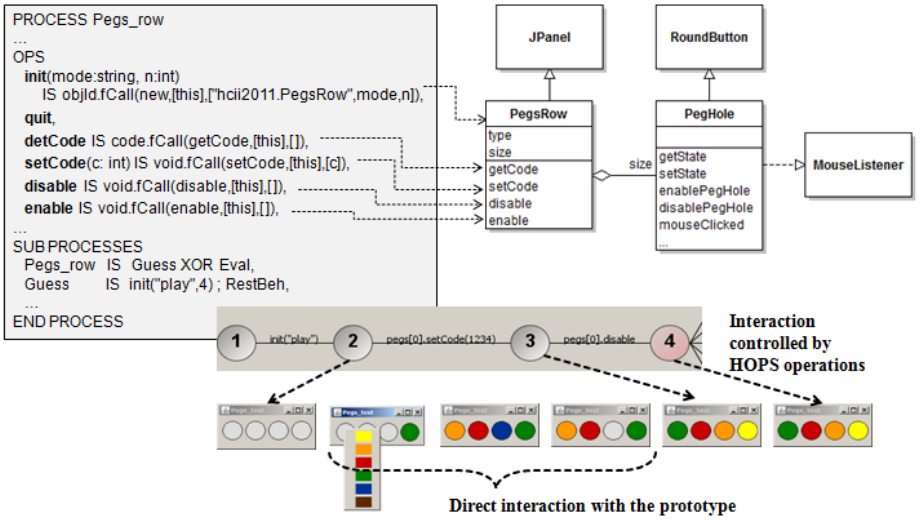

Fig. 10. Top: mapping between HOPS operations and the implementation for rows of peg holes (mastermind), bottom: test of the prototype, partially controlled by the HOPS process

resource allocation problem. It is important to understand existing transfer and control mechanisms in the interplay between different actors and (internal and external) artifacts in order to adapt them.

Fig. 11 illustrates a coupling of HOPS models that can be used to explore the design space of a mastermind board. The components describe assumptions about the actors in a mastermind game and about resources that are supplied by the interactive prototype. The computer acts as codebreaker and a human as codemaker in the specified situation (HOPS components *cb* and *cm*). The model of the activity *Break_code* is depicted in the right top corner. Take note that this model describes the codemaker signaling that the marking is finished. Although people give also such signals when they play together (e.g. by leaning back in their chair) this was not considered in the task description of Fig. 4.

An essential idea of selective modeling is that designers deliberately underspecify representations which are in the actual focus. They are coupled with complementary design representations to develop assumptions about possible usage situations and to reflect design options. In Fig. 11, the models of actors and of the code-breaking activity help to put constraints on the prototype of an interactive mastermind board and to test variants of its refined behavior in model-guided animation runs. In the interaction scenario at the bottom of the figure, the computer made a guess and the peg row was disabled. Then, the user marked the guess (indicated by HOPS operation *setPegs*) and signaled the end of this process (operation *finish*). This caused a disabling of this marking space. The prototype has to generate a new guess now.

Other models could assume e.g. a shared evaluation of guesses. There are two human players but the computer could check whether the markings are correct. This could be useful in situations where the codebreaker is a beginner and the codemaker

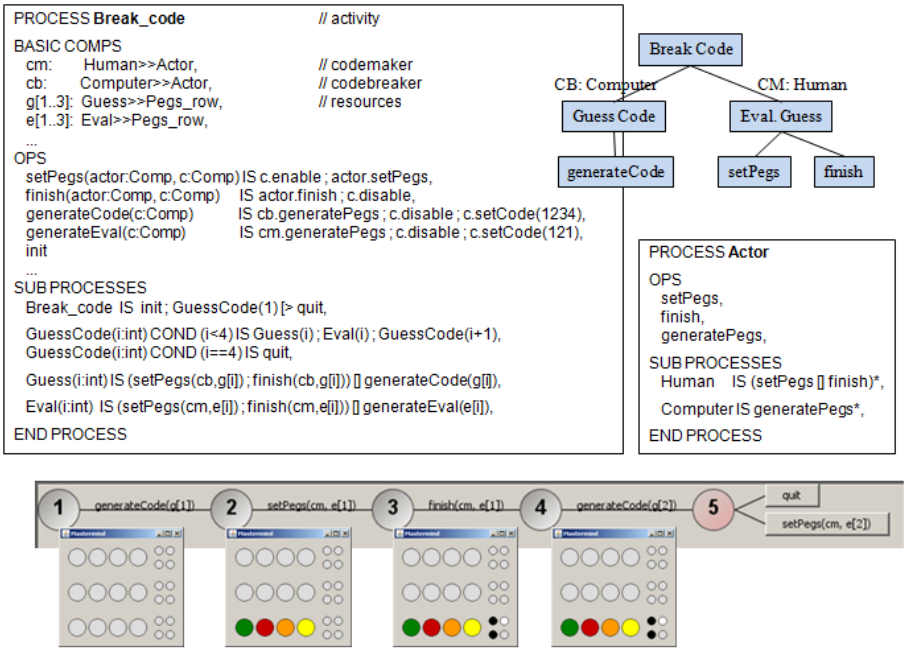

Fig. 11. The underspecified prototype of an interactive mastermind board (mainly described by components g[1..3], e[1..3], see Fig.10) is examined and explored for situations with the computer as codebreaker and the user as codemaker

is more concentrated on giving explanations than on marking. The co-evolution of models helps to develop a more fine-grained understanding of resource allocations and informs design decisions.

6 Summary

"Design are hypotheses about how artefacts shape cognition and collaboration." (D. Woods)

Task migratability is a usability principle that encourages designers to develop a deep understanding about possible usage situations of an interactive artifact. Refined flexible task allocation is rarely achievable through pure top-down decomposition of tasks. Resource-based conceptions of human activity seem to be more appropriate than task-based ones to face complex design problems.

Selective modeling has been proposed as a technique to adapt some of the assumptions and practices in model-based design. Designers are encouraged to co-evolve different representations which describe the problem space from different viewpoints. They are partially coupled. Sometimes they are deliberately under-specified to allow for reflection and exploration by other models. The interactive artifact under design "emerges" from these descriptions. Selective modeling is a model-guided but not a model-driven process. The designers need to be aware of the actual design situation.

Design reflects what we consider as important in the current world but also what we don't see or too late. The co-evolution of different design representations is often recommended in UCD in order to ground system design in current work practices. However, an effective coupling is still not easy to achieve. The general approach of selective modeling has been demonstrated in the paper at the example of the HOPS formalism and prototypes in Java. The examples given in this paper demonstrate the practicality of the method. Further work aims to improve mapping mechanisms between models, to improve tool support and to explore a broader range of examples.

Acknowledgments. We would like to thank Stefan Piehler, Toralf Hübner, and Mathias Kühn for their contributions to the HOPS tool. Our thanks also go to the reviewers of this paper for their useful comments.

References

1. Dix, A., Finlay, J.E., Abowd, G.D., Beale, B.: Human-Computer Interaction, 3rd edn. Prentice-Hall, Englewood Cliffs (2003)
2. Benyon, D., Turner, P., Turner, S.: Designing interactive systems: people, activities, contexts, technologies. Addison-Wesley, Reading (2005)
3. Dittmar, A., Forbrig, P.: Task-based design revisited. In: Proc. of ACM SIGCHI Conf. on Engineering Interactive Computing Systems (EICS 2009), pp. 111–116 (2009)
4. Dittmar, A., Harrison, M.D.: Representations for an iterative resource-based design approach. In: Proc. of ACM SIGCHI Conf. on Engineering Interactive Computing Systems (EICS 2010), pp. 135–144 (2010)
5. Diaper, D.: Understanding Task Analysis for Human-Computer Interaction. In: Diaper, D., Stanton, N. (eds.) The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates, Inc., Mahwah (2004)
6. Dearden, A., Harrison, M., Wright, P.: Allocation of function: scenarios, context and the economics of effort. Int. Journal of Human-Computer Studies 52(2), 289–318 (2000)
7. Degani, A.: Taming HAL: Designing Interfaces Beyond 2001. St Martin's Press Inc., New York (2004)
8. Carroll, J.: HCI Models, Theories, and Frameworks: Toward a multidisciplinary science. Morgan Kaufman Publishers, San Francisco (2003)
9. Johnson, P.: Human computer interaction: psychology, task analysis, and software engineering. McGraw-Hill Book Company, New York (1992)
10. Annett, J.: Hierarchical Task Analysis. In: Diaper, D., Stanton, N. (eds.) The Handbook of Task Analysis for Human-Computer Interaction, Lawrence Erlbaum Associates, Inc., Mahwah (2004)
11. John, B.E., Kieras, D.E.: The GOMS family of user interface analysis techniques: Comparison and contrast. ACM Trans. on Computer-Human Interaction 3, 320–351 (1996)
12. Dourish, P.: Where The Action Is: The Foundations of Embodied Interaction. MIT Press, Cambridge (2001)
13. Suchman, L.: Plans and Situated Actions: The Problem of Human Machine Interaction. Cambridge University Press, Cambridge (1987)
14. Hollan, J., Hutchins, E., Kirsh, D.: Distributed cognition: toward a new foundation for human-computer interaction research. ACM Trans. Comput.-Hum. Interact. 7(2), 174–196 (2000)

15. Kaptelinin, V., Nardi, B.A.: Acting with technology: activity theory and interaction design. MIT Press, Cambridge (2006)
16. Wright, P.C., Fields, R.E., Harrison, M.D.: Analyzing human-computer interaction as distributed cognition: the resources model. Human Computer Interaction 15(1), 1–42 (2000)
17. Paterno, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, Heidelberg (2000)
18. 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(3), 289–308 (2003)
19. Rosson, M.B., Carroll, J.M.: Usability Engineering – Scenario-Based Development of Human-Computer Interaction. Morgan Kaufmann Publishers, San Francisco (2002)
20. Beyer, H., Holtzblatt, K.: Contextual Design – Defining Customer-Centered Systems. Morgan Kaufmann Publishers, San Francisco (1998)
21. Greene, S.L., Jones, L., Matchen, P., Thomas, J.C.: Iterative development in the field. IBM Syst. J. 42(4), 594–612 (2003)
22. Gould, J.D., Lewis, C.: Designing for usability: key principles and what users think. Communications of the ACM 28(3), 300–311 (1985)
23. Fischer, G., Lemke, A., McCall, R., Morch, A.: Making Argumentation Serve Design. In: Moran, T., Carroll, J. (eds.) Design Rationale: Concepts, Techniques and Use, pp. 267–293. Lawrence Erlbaum Associates, Inc., Mahwah (1996)
24. Barboni, E., Ladry, J.-F., Navarre, D., Palanque, P., Winckler, M.: Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models. In: Proc. ACM SIGCHI Conf. on Engineering Interactive Computing Systems (EICS 2010), pp. 143–152 (2010)
25. Dittmar, A., Hübner, T., Forbrig, P.: HOPS: A Prototypical Specification Tool for Interactive Systems. In: Graham, T.C.N. (ed.) DSV-IS 2008. LNCS, vol. 5136, pp. 58–71. Springer, Heidelberg (2008)