Towards Formalized Adaptation Patterns for Adaptive Interactive Systems

Matthias Bezold

[1] University of Ulm, Institute for Information Technology, Ulm, Germany
[2] Elektrobit Automotive Software, Erlangen, Germany
matthias.bezold@uni-ulm.de

Abstract. A design pattern provides a general and proven solution for a recurring problem. Design patterns are an established approach in the domain of software engineering. Collections of such patterns also exist for graphical interfaces and adaptive hypertext. However, a collection of patterns for adaptive interactive systems does not exist. This paper presents such a collection to provide structured knowledge about applying adaptations to interactive systems. In addition, a formalization of these patterns using semantic technologies is presented as well as the application of these formalizations in an adaptation framework.

1 Introduction

Adaptive interactive systems describe user interfaces that change based on the user-system interaction to better reflect the requirements of an individual user. Adaptation has been recognized as a means for improving the usability of user interfaces and studied accordingly [13]. For instance, one possible adaptation highlights frequently used values in a list or another one emphasizes interface elements that might be of increased interest to the user. However, there is no structured work so far that lists and categorizes different kinds of adaptations for adaptive interactive systems. At the same time, tool support and frameworks facilitate a wide-spread use of adaptations in interactive systems.

This paper proposes an approach for adapting multimodal interactive systems, such as automotive dashboard systems, personal navigation devices, or home entertainment systems, which can also be speech-enabled. Two contributions are presented. First, an adaptation architecture employs an abstract definition of adaptation patterns to define adaptations for interactive systems. These definitions can be reused between different systems, but at the same time adjusted to the requirements of specific systems. The approach is based on a semantic description of the interactive system and the adaptations are integrated into a model-based development process. Second, we describe a set of adaptation patterns for interactive adaptive systems. These patterns define successful adaptations that have been used in different systems.

This paper is organized as follows. After a review of related work in Section 2, the application of formalized adaptation patterns in the development of

interactive systems in demonstrated in Section 3. Section 4 introduces a set of adaptation patterns that can be used with the presented framework. After an overview of a prototype implementation and a use case in Section 5, Section 6 concludes this paper.

2 Related Work

This section introduces the concept of patterns, reviews patterns from related work, e.g. interface patterns, and presents approaches for the formalization of design patterns. Design patterns are an established method in software engineering that collects solutions for recurring problems. A design pattern consists of a proven solution and a discussion of a problem it solves. Moreover, the context of the pattern further refines the circumstances under which this pattern is applicable. One problem can be addressed by different patterns and the context determines which pattern is used best.

The most well-known use of design patterns are the software design patterns by Gamma el al. [7]. These patterns are widely adopted and taught in classes. Similarly, Buschmann et al. [4] present a set of patterns for software architecture, which deal with a more high-level view on software design.

Originally, patterns only existed in textual, narrative form. However, research on the formalization of patterns aims to increase their utility. A formalized description is a representation using a well-defined structure and vocabulary, thus providing a standardized and machine-processable representation. Different levels of formalization exist for patterns. The first level is to write down the narrative pattern descriptions in a formalized pattern format, which enforces special markings to label the different sections (e.g. motivation or solution) of a pattern description. This ensures consistency and allows referencing between different pattern collections by providing a machine-readable structure in which the patterns are filled in. The Pattern Language Markup Language (PLML) [6] follows this approach and provides an XML document type definition (DTD) for specifying patterns. PLML is however a very high-level definition that aims at describing pattern collections in a uniform way and the semantics of the patterns are not formalized.

A more formal notation of patterns can serve as a basis for intelligent tool support, for instance by providing support when refactoring existing projects to patterns (e.g. Zannier and Maurer [20]). Other approaches even formalize the semantics of the adaptation patterns. Mikkonen [16] presents an approach for formalizing patterns based on a custom notation for defining objects formally, with the focus being on the temporal behavior of design patterns. Hallstrom and Soundarajan [9] present an approach that enables validation and reasoning with patterns. Another example is the work by Henninger [11], who proposes a meta-model for software patterns based on an Web Ontology Language (OWL) infrastructure for applying the patterns in the software development process. This model conceptually builds on PLML, but extends it considerably by including a more formalized representation of the patterns using description logic

to add further semantic knowledge about the patterns. Henninger presents interface patterns as an example of this approach. Our approach provides tool support for adaptation patterns and includes a semantic description of parts of the patterns, but does not fully formalize the precondition for patterns. Therefore, our approach is located between simple tool support and fully formal pattern systems.

Whereas design patterns are mostly used in the domain of software engineering, patterns were also applied by different researchers to the domain of user interface design. For instance, van Welie and van der Veer [19] and Borchers [3] have compiled extensive pattern collections of reusable interface design knowledge, which can be used by designers and system developers in creating graphical interfaces. Tidwell [18] presents an extensive structured catalog of interface design patterns, which covers a wide range of topics, such as the general structure of a graphical application, form input, and aesthetics. However, these patterns do not cover adaptive user interfaces.

A basic set of abstract adaptation patterns, which are descriptions of proven adaptations, was presented for adaptive hypertext systems by Danculovic et al. [5], introducing Link Personalization, Content Personalization, Structure Personalization, and Remote Personalization, which all are very general. These patterns were extended by Koch and Rossi [14] by adding more detailed patterns such as Adaptive Anchor Selection or Adaptive Sorting of Anchors. However, the adaptation of hypertext is focused on content and the linking of different documents rather than the user interface, as required for adaptive (graphical) interfaces. Moreover, more information can be extracted from the user-system interaction of an interactive system, since the observation by an interactive system is richer than tracking the list of visited pages in hypertext systems. Therefore, adaptive interactive systems have their own adaptation patterns and defining such patterns can aid developers in deciding under which circumstances to apply which adaptations to improve an interactive system.

3 Formalization and Execution of Adaptation Patterns

This section describes how a description logic-based, semantic definition of adaptation patterns is used to enable adaptivity in interactive systems. In order to apply these patterns to an interactive system, we introduce a semantic model of the interactive system.

Whereas the domain of the design patterns by Gamma et al. [7] is source code, the domain of these adaptation patterns are interactive systems. Therefore, the abstract description of interactive systems used in this work corresponds to an abstract description of source code that is required by tool support for design patterns. For this purpose, the interactive system is described by a semantic layer. This layer is based on an ontology defined in the Web Ontology Language (OWL) [17] format and consists of a number of classes and individuals of these classes. For instance, each graphical element or speech output prompt in the interactive system is represented by an instance of a “button” or “prompt” class

and further described by a set of properties, such as color and size in case of the button. The semantic layer is derived automatically from a description of the interactive system, but annotation contributes further information. The advantage of this approach is that all parts of the interactive system, e.g. the interactive system, information about the user, and a description of the adaptations, are represented using the same formalism. In addition, OWL allows for an automatic inference of additional information (cf. Horrocks et al. [12]).

An adaptation pattern is a description of a common problem of interactive systems and a solution of this problem that is based on adapting the interactive system to the behavior of a user. The formalized definition of adaptation patterns in this work consists of two parts: a declarative description that can be reused between different systems and a functional description of the adaptations that defines how an adaptation is performed in a certain system. The formalization facilitates an inclusion in the tool chain, thus enabling a tool-supported development of adaptive interfaces.

3.1 Declarative Description of Adaptation Patterns

The declarative description of patterns consists of three parts: a trigger of the pattern, a selection that determines which part of the interface the adaptation should be applied to, and the name of an abstract adaptation. The declarative description is called *adaptation selector* in the adaptation framework, because it selects an interface element and an adaptation. Adaptation selectors can be reused between different systems, since they only rely on abstracted information in the semantic layer to define the declarative part of the adaptation. For instance, graphical elements are addressed as "graphical button" or "list", which can support a variety of different implementation and flavors of actual interface elements. The declarative description of an adaptation pattern consists of a number of adaptation selectors.

The adaptation trigger is connected to the user modeling component [2], which observes the user and derives information from the user-system interface. For instance, the user modeling component can predict future user actions (e.g. opening a certain sub-menu) or the user's favorite values (e.g. favorite names in an address book). When the user modeling component provides new information, the respective adaptation selectors, which are triggered by this information, are activated.

Interface elements are selected by an adaptation executor through a query that returns all elements from the semantic layer that match the given query. For instance, if the user modeling predicts that a certain action called "A" will be performed next by the user, the selection could load all elements from the semantic layer that trigger action "A". A simplified version of SPARQL [3] is used for the notation for the queries.

[3] SPARQL Protocol and RDF Query Language: http://www.w3.org/TR/rdf-sparql-query/

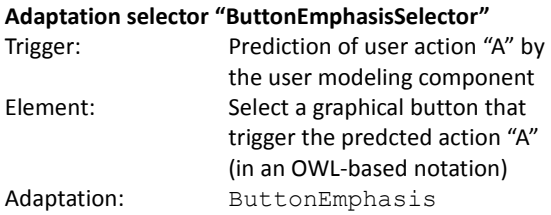

Fig. 1. An adaptation selector selects all graphical buttons that trigger an action that was predicted. In addition, an abstract adaptation ("Component Emphasis") is chosen for the selected elements.

Finally, an abstract adaptation is recommended for the selected elements. For example, the selected button that triggers action "A" could be emphasised to draw the user's attention to it by enabling the "ButtonEmphasis" adaptation, which is an instance of the "Component Emphasis" pattern (to be introduced in Section 4).

Fig. 1 shows an adaptation selector that was used as an example throughout this section: Based on a prediction of a user action "A" by the user modeling component, the adaptation selector picks all graphical buttons from the semantic layer that trigger action "A" and recommends the "Component Emphasis" adaptation for these elements.

3.2 Functional Description of Adaptation Patterns

In order to enable the interactive system to apply an abstract adaptation recommended by an adaptation selector, a functional description of the pattern is required, i.e., instructions on how to execute this pattern on a specific interface element. Adaptation executors can be system-specific and therefore not be reused between different systems in every case. The reason is that the execution of adaptations depends on the specific implementation of the interface element. However, default implementations have been defined that can be used on a wide range of elements by changing only basic properties (such as the position or the size).

The functional definition of patterns specifies the effects of an adaptations by defining which properties of the individuals are changed. Fig.2 shows an adaptation executor of the "Component Emphasis" pattern that changes the color or the size of a graphical button, which are represented by properties of the corresponding individual. One adaptation pattern is not represented by a single adaptation executor, but by a set of adaptation executors for different interface elements, since the same adaptation manifests itself very differently for different elements. For instance, the Emphasis pattern has a different functional description for a graphical button than it has for a graphical list.

Adaptation executor "ButtonEmphasisExecutor"

Adaptation:	`ButtonEmphasis`
Interface element:	`Graphical button`

Property changes:	`x = x - 5`
	`y = y - 5`
	`width = width + 10`
	`height = height + 10`
	`textColor = color.yellow`

Fig. 2. Example of an adaptation executor, which perform the Emphasis adaptation on a button.

3.3 Application of the Adaptation Patterns in an Interactive System

The previous section discusses how adaptations are defined, but not how the adaptive system decides whether to execute a suggested adaptation. This section introduces two approaches for defining which adaptations to execute: selection by the system designer and an automatic procedure in which the adaptation component decides automatically which adaptations to execute.

Specification by the System Designer An integration of formalized adaptation patterns into the model-based development process thus facilitates the development of adaptive interactive systems. At design time, the formalized adaptation patterns provide tool support to the designer of the adaptive system. For this purpose, the system designer can decide when the individual adaptations should be executed by enabling or disabling adaptations for certain parts of the system.

Adaptations can be enabled on three levels: globally, for an interface context, or for an individual element. First, if an adaptation is enabled globally, it is executed whenever it is recommended by an adaptation selector. Second, adaptations can be enabled for interface contexts. An interface context is for example a graphical screen or a speech component, which is a set of speech output prompts and speech input grammars that are enabled together. When an adaptation is enabled for an interface context, it is executed in the respective context, but not in others where the adaptation was not enabled. Third, adaptations can be enabled on a per-element basis.

In addition, conditions can be added these definitions. For instance, an adaptation should only be executed when the user is a beginner, based on information stored in the user model or the semantic layer. Therefore, the system designer can decide in a very flexible way which adaptations should be executed, without the need of defining the adaptations manually.

Automatic Execution by the Adaptation Component In addition to the developer deciding about which adaptation to apply, an adaptation component

can decide automatically at runtime about the execution of adaptations. This decision is based on the semantic representation of both the interactive system and the adaptations. The overall procedure is similar to the specification of adaptations, but decisions taken at design time are instead performed by the adaptation component at runtime.

The adaptation component can automatically decide about the level of adaptivity, based on the proficiency of the user in general or with respect to individual parts of the system. For example, adaptive help can be used for parts of the system that the user has not used extensively, while no adaptations are used in parts that the user knows well.

4 Adaptation Patterns

This section presents a set of adaptation patterns for interactive systems. These adaptation patterns change the user interface of the system, but do not directly depend on the application logic. Certain adaptations are outside of the scope of this paper, because they do not address general problems, but specific algorithms, such as for instance an adaptation of the route generation algorithm of a navigation device by the Adaptive Route Adviser [15]. Instead, this paper deals with general adaptations for user interfaces of interactive systems. Since multimodal adaptive systems are the subject of the adaptations, their applicability to speech-based interfaces is also considered.

The patterns presented in this section are more general in their nature than patterns in other pattern collections, which make them applicable to a broad range of interface components. Unobtrusiveness is a main principle of these adaptations to comply with usability principles such as consistency and learnability. Adaptations that reconstruct the whole interface thus interfere with such usability principles.

Adaptations are executed based on an observation of the user-system interaction. This observation of user behavior, called user modeling, creates a representation of the user that serves as a basis for decisions about adaptations. For this purpose, a user modeling component observes the user (e.g. from log data) and constructs a model using different algorithms, such as the ones presented by Zukerman and Albrecht [21], thus providing information about user actions and preferences. The user modeling phase, which is therefore a crucial part of adaptive interfaces, is not explicitly part of these pattern descriptions. However, the "Adaptation Trigger" section of the patterns describes which observations of the user modeling component trigger the respective adaptation.

Adaptation selectors and a set of generic adaptation executors, as introduced in Section 3, were defined for the patterns presented in this section. In order to make an adaptation fit into a specific system seamlessly, custom adaptation executors can be defined in addition to the existing ones.

4.1 Component Emphasis

Intent Guide the user by emphasizing certain elements of the interface. Limit the changes to the part of the interface that requires emphasis. In doing so, enable users to reuse their acquired knowledge of the interactive system and avoid distracting the user through fundamental changes of the interface.

Motivation During the interaction with an interactive system, a user has a goal and is looking for interface elements that can help in fulfilling it. For instance, the user might look for a graphical button triggering an action. The system provides support by guiding the user to the respective interface elements.

Forces

- The user follows a certain goal when using the system and might spent considerable time looking for interface elements that facilitate reaching this goal.
- Performing major changes to the system can confuse the user and distract from the current task. Subtle guidance instead supports the user.
- Emphasizing wrong elements can impede the user, therefore a sufficiently good user modeling prediction is crucial for this adaptation.
- The adaptive emphasis should be conceived in a way that the user does not confuse it with a regular selection in the user interface.

Solution Make the adaptive system change properties of interface elements in a way that they draw the user's attention. Use assumptions of a user modeling component, such as a prediction of the most likely next action or an action the user has not used yet. Help the user reach the current goal by emphasizing interface elements that are related to the respective assumption of the user modeling component.

Adaptation Trigger The following observations of the user modeling component trigger the Component Emphasis adaptation:

- Prediction of the next user action.
- Actions that the user has not used yet, but which others have used.

Related Patterns The "List Item Emphasis" pattern emphasizes elements in a list and is therefore related to this pattern, which emphasizes arbitrary elements related to triggering actions.

The "Prominent 'done' button" pattern [18] statically emphasizes a button that finishes a task associated with a graphical view, but the emphasis is not performed based on the current user's behavior.

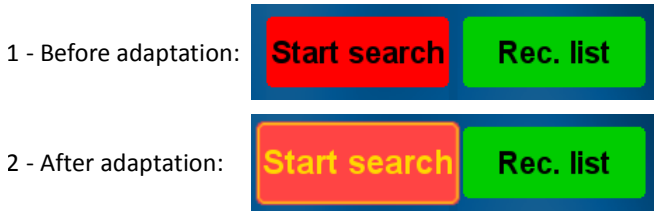

Fig. 3. Emphasis of a button in an interactive TV system. The "Start search" button is emphasized compared to the non-emphasized "Rec. list" button. The reasoning is to provide non-intrusive and subtle hints, in this case by increasing the size of the button and changing the text color.

Example Consider an electronic program guide, where the user specifies filter criteria, such as channel or time, to filter the list of TV shows. After a number of criteria was selected, the user has to press a "Show results" button to see all shows that match the selected criteria. Increasing the size of the button and changing colors (see Fig. 3) emphasizes the button, thus supporting the user in finishing the current task.

The Emphasis pattern is also applicable to voice interfaces. If a user enters a state where the system reads the possible utterances, saying a phrase as the first or the last one draws a user's attention to this phrase.

4.2 List Element Selection

Intent Support the user in selecting similar-looking elements from a list, for instance by highlighting frequently used entries from the list.

Motivation When selecting elements from a list, users often select some elements frequently and others not at all. The selection process can be improved by emphasizing frequently selected elements from the list.

Forces

- Selecting frequently used items in a list should take less time for the user than selecting others.
- If a list is longer than one screen, highlight the interesting items also in the scrollbar to enable the user to quickly scroll to the interesting elements.
- Emphasized list elements should be highlighted in a way that the user does not confuse them with elements the cursor is placed on.
- Emphasizing wrong elements can impede the user, therefore a sufficiently good user modeling prediction is crucial for this adaptation.

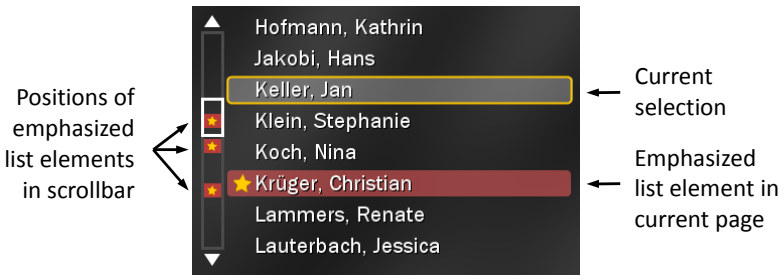

Fig. 4. Three elements are emphasized in a selection list by the List Item Emphasis pattern. The emphasized elements are supposed to be selected more frequently by the user than others.

Solution Emphasize these elements in the list that have been selected more often before than others. In doing so, let the user more quickly see these elements which are of increased interest. For instance, change the text or background color of these elements or add markers to differentiate interesting elements from others.

Adaptation Trigger The following observations of the user modeling component trigger the List Item Emphasis adaptation:

- List entries that have been selected more often than others either by the current user or by other users.
- Elements in a list that the user has not yet selected, but which should be interesting based on the user's previous behvior.

Related Patterns The "Element Emphasis" adaptation pattern also emphasizes interface elements, but these elements are not necessarily similar, as are list elements, and are mostly used for navigating within the system.

The "Annotated scrollbar" pattern [18] recommends adding information to the scrollbar, which is also proposed by this pattern to mark the position of recommended elements in the list. Moreover, the "Adaptive Anchor Annotation" [14] pattern describes how to annotate links in a hypertext system, and this pattern can be considered as a sub-set of the anchor annotation pattern by annotating emphasis.

Example Selecting elements from a list is a very common action when interacting with interactive systems. For instance, selecting a name from the address book is one of the fundamental functions of interactive systems that support phone calls, such as mobile phones or automotive dashboard systems. Since users call a small number of people from their phone book frequently, the selection of these names from the address book can be quickened by highlighting these

names. An example of such a system is given in Fig. 4, which shows an address book that emphasizes the three most frequently selected elements.

Different visualizations of the "List Item Emphasis" pattern are possible and have been examined by research projects. One example are fisheye menus [1] that assign a different font size to different elements; this kind of visualization can be employed for adaptations as well.

4.3 Alternative Elements

Intent Provide a set of configurations for different interface components or screens and select the appropriate configuration to better support the requirements of an individual user.

Motivation Since the demands as well as the skills of users of interactive systems vary, different system configurations can better reflect the needs of an individual user. Instead of providing one configuration that tries to consider all possible users, the adaptation selects the version which is best suited for the needs of the current user. A user modeling component provides information about the proficiency of the user, which it derives for instance from the interaction speed and the number of user errors.

Forces

- Different configurations of interface components or graphical screens better reflect the needs of individual users.
- Automatically generated alternatives can break with existing usability principles.
- Additional time has to be spent developing the different alternatives, but the user can benefit from an improved user-system interaction.

Solution Provide different versions of a certain part or component of the interactive system to the adaptation component, for instance of a graphical screen, a speech output prompt, or a property (e.g. font size). Support users by selecting the appropriate alternative for the respective entity. Use information from the user modeling component at runtime to derive the most suitable configuration for the current user. By providing a set of alternatives to the adaptation component, which were created by the system designer, it is ensured that the interactive system adheres to design principles.

Adaptation Trigger The following observations of the user modeling component trigger the Alternative Elements adaptation:

- Preferences or properties of the user, such as the knowledge level or experience of the user.

Related Patterns The "Alternative views" pattern [18] lets the user decide among alternative views, for instance of a web page. However, the most appropriate view is not selected automatically.

Example The Alternative Elements pattern can be employed at different levels. For instance, when a user has to enter different values in an input screen, such as selecting the destination in a navigation device or selecting criteria in an interactive TV program guide, a simple version of the screen is provided to novice users and a more powerful version to advanced users. On a lower level, a larger font size improves the readability for visually impaired users.

On the other hand, a speech interface can provide different levels of speech output prompts. Novice users receive extended prompts when they enter a new part of the system. These prompts explain the most important functions to them. Intermediate users only require shorter prompts, which list the commands, but do not necessarily explain them. Finally, expert users, who could be annoyed by long and repetitive speech output, only hear a short prompt explaining the current state of the system and receive more explanation on request. A system that employ this kind of adaptation is for example presented by Hassel and Hagen [10].

4.4 Adaptive Help Presentation

Intent Present adaptive help for the current situation of the user.

Motivation Help in interactive system is often static or only considers the currently active screen, but different people are likely to have different problems in different contexts. Providing help to the user is more valuable if it covers the current task of the user. By not only taking into account the current context, i.e., the graphical screen or speech state, but also the user's interaction history, help is more specific and can thus support a user more precisely in the current task.

Forces

- Help tailored to the current task of the user is more valuable than static help.
- Static help can be too advanced for beginners and at the same time too superficial for expert users.
- Providing help can be assistive for beginners, but annoying for expert users.

Solution Provide specific help for the current situation of the user. Observe the user-system interaction to determine the situation and the context of the user. Present the help either on a separate area of the screen, or use an icon (or an acoustical "earcon") to indicate the availability of help. Give the user an option

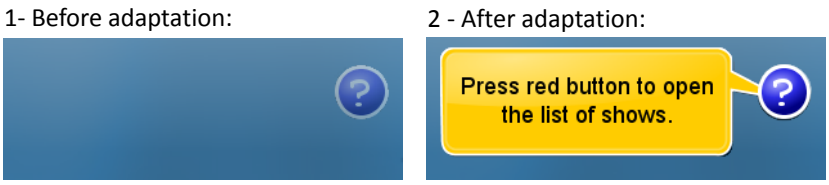

Fig. 5. Adaptive help supports the user by showing a text message that fits the current situation of the user. If a user is sufficiently proficient in working with the system, help messages are no longer shown.

to open this help once it is available. But ensure at the same time that the user is not distracted by the provided help. Therefore, avoid messages that fully engage the user's attention, as for instance modal help messages. Provide an acoustic signal instead of a graphical hint for speech interfaces or visually impaired users.

Adaptation Trigger The following observations of the user modeling component trigger the Adaptive Help Presentation adaptation:

- Prediction of the next user action.
- Detection of user problems.
- Preferences or properties of the user, such as the knowledge level or experience of the user.

Related Patterns The "Multi-level help" pattern [18] suggests using different help techniques. Adaptive help is one kind of help that provides information adjusted to the current situation of the user.

Example In an interactive TV system, the user can browse the TV program in an electronic program guide and for this purpose specify different filter criteria, such as channel or time. Help is presented to the user by fading in a yellow message box on the top of the screen. When the user enters the selection screen for the first time, the help explains how to select filter criteria. After some criteria were selected, the help text on the screen tells the user to open the result screen next. Fig. 5 gives an example of the adaptive help feature in a digital TV system.

4.5 Shortcut Area

Intent Present shortcuts for executing actions or selecting values to the user on a separate part of the interface. In doing so, accelerate the execution of frequent actions or sequences of actions and selection of the user's favorite values.

Motivation Users often select the same values repeatedly, for instance when selecting elements from a list, such as a list of fonts, or by executing the same actions over and over again. The interaction is simplified by presenting these items to the user as shortcuts. By employing a special area for the shortcuts, the decision whether to use shortcuts is left to the user.

Forces

- Finding frequently used elements and executing actions repeatedly can be very time-consuming for the user. Shortcuts can therefore simplify the user-system interaction.
- Shortcuts that automatically pop up on top of the interface interfere with the user interface and distract the user. A separate area that is always visible instead allows the user to decide whether or not to use shortcuts and limits the distraction of the user.

Solution Employ a separate area of the screen – called shortcut area – to present shortcuts to the user, thus avoiding a distraction of the user. Make this list either part of one interface element (e.g. of a list) or make it a separate part of the whole screen for presenting global shortcuts. In doing so, enable the user to find frequently used elements more quickly by selecting them from a distinct area of the screen. Use the output of a user modeling component to create the list of shortcuts.

Adaptation Trigger The following observations of the user modeling component trigger the Shortcut Area adaptation:

- Prediction of the next user action or a sequence of user actions.
- Prediction of a user preference, such as a TV channel.

Related Patterns The "Streamlined repetition" pattern [18] suggests considering repeated operations when creating an interface. The Shortcut Area provides a solution for this recommendation. The "Action panel" pattern [18] presents a list of available actions to the user, which is similar to an Shortcut Area that contains user actions. If the Shortcut Area presents a sequence instead of single items, the adaptation is similar to the "Autocomplete" pattern [18], since the adaptation anticipates user behavior.

Example In a selection list, a separate area on the top of the list presents the most frequently selected entries of the list to the user. By selecting them, the user does not have to scroll through the whole list. One example of such a list is the font selection list in Microsoft Word (2000 and later), which shows the most recently used fonts in a separate area on the top of the list.

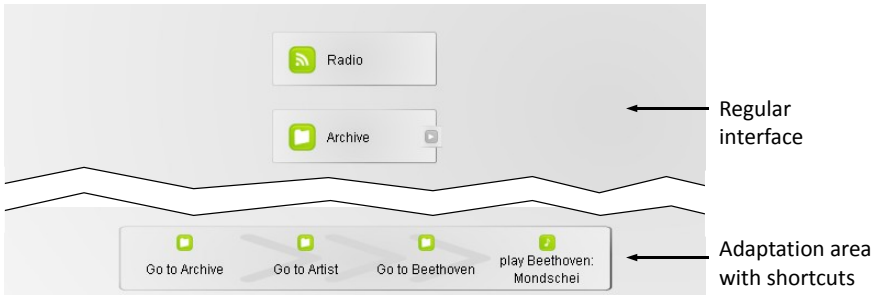

Fig. 6. A separate adaptation area presents a list of buttons based on a prediction of the user's next actions. These actions are executed by pressing the respective buttons.

A different application of the Shortcut Area pattern is to provide navigation shortcuts. A user modeling component recognizes user actions and predicts a sequence of possible next actions, with each action being represented by a button in an adaptation area. If the user presses one of these buttons, the action associated with the button and all actions before the pressed button are executed, thus reducing the number of required interactions. An example of an interface that provides navigation shortcuts to the user is shown in Fig. 6: In addition to the regular interface (shown on top), the interactive system presents a list of likely next actions to the user on the bottom of the screen. If the user presses one of these buttons, the interactive system automatically executes the respective actions.

5 Implementation and Sample Use Case

This section gives an overview of an implementation of the adaptation approach presented in Section 3, which includes a definition of the adaptation patterns introduced in Section 4. In addition, a sample use case further illustrates the use of this approach.

5.1 Implementation

In order to investigate the approach presented in this paper, an adaptation framework was developed within a model-based development environment called EB GUIDE Studio [8]. The tool comprises a simulation component that executes a specified application. A semantic layer was added that uses an OWL-based ontology to describe the system, but also the user-system interaction. The semantic layer is implemented using the Jena framework[4] and is available both at design time and at runtime.

The framework comprises an adaptation component, which works as discussed in Section 3. Moreover, it includes a set of adaptation selectors for the

[4] Jena Semantic Web Framework: http://jena.sourceforge.net/

adaptations presented in Sect. 4. Since these selectors are defined on the abstract level of the semantic layer, they can be reused between different interactive systems. In addition, the framework includes a set of default adaptation executors that only change properties common to all graphical elements. In doing so, these can also be used for different systems, but a better integration into the specific style of a certain system can be achieved by defining custom executors for a specific system.

The prototype implementation includes a user modeling component that models user behavior from low-level events, such as key presses by the user and system reactions to these inputs. Based on these low-level events, information such as the most likely next interaction step or a user preference, such as a favorite TV channel or font, are computed and forwarded to the adaptation component and makes these user modeling events available as triggers.

To show the feasibility of this approach, adaptations were implemented in two different interactive systems. First, an adaptive version of a interactive TV prototype application was created. Among the implemented adaptations are Adaptive Help Presentation, which guides novice user through the system, and Component Emphasis, which emphasizes buttons the user is most likely to use next. The latter adaptation is discussed in detail as a use case in Section 5.2. Second, an adaptive version of a system resembling an automotive dashboard system was created. It includes the Alternative Elements adaptation, which in this system selects between a beginner and an expert version of a route guidance entry screen, and the List Item Emphasis adaptation, which highlights frequently selected names in an address book.

5.2 Sample Use Case

This section presents a use case of applying an adaptation pattern to an interactive system. In addition to the application of the adaptation pattern, the preparation of the user modeling component is also discussed. As an example, the Component Emphasis adaptation is applied to a TV system, which includes an electronic program guide (EPG). After selecting a set of filter values, such as channel, time, or genre, the user can open the results screen by pressing a button labeled “Show Results”. Since novice users could not be aware that the “Show Results” button has to be pressed, emphasizing it can help the user in browsing the EPG.

The user modeling component has to be set up in a way that it observes the user’s behavior and predicts user actions. In this framework, an algorithm based on Markov chains is used for action prediction and an adaptation trigger is emitted when such a prediction was made.

Most of the semantic layer is generated automatically from the model-based description in the development tool. However, some manual annotation was still required. In this case, the information that the “Show Results” button triggers the “OpenResultList” action is annotated to the button and loaded into the semantic layer.

Finally, a custom adaptation executor could be defined for this interactive system. The behavior of the default generic adaptation executor, which simply increases the size of the interface element, might not fit in well with the design of the system, and therefore, a custom adaptation executor, which increases the size through an animation and make the background color slightly lighter, was defined.

The execution of the adaptation then works as follows. First, the user modeling component emits an event that a user action was predicted and thus triggers "ButtonEmphasisSelector" selector, which is part of the framework and selects a button that triggers the predicted user action. This selector then recommends the abstract "Component Emphasis" adaptation for execution and the adaptation component tries to find an appropriate executor. Two executors are available: the default executor, which is part of the framework, and the custom executor defined for this system. In this case, the adaptation component gives priority to the custom executor and executes the custom adaptation executor accordingly.

6 Conclusion

In this paper, we presented a definition of adaptation patterns that can be integrated into the model-based development process. A set of general adaptation patterns for adaptive interactive system was defined. A sample implementation of the presented framework was presented to show the feasibility of this approach. The adaptation patterns defined in this paper were integrated into the framework and the adaptations were applied to two sample systems. A detailed use case further illustrated the approach.

Acknowledgements

The author is indebted to Paul Adamczyk, the shepherd of this paper for EuroPLoP 2009, for his time, comments, and observations, which allowed the paper to evolve during the shepherding process. Many thanks also for the insightful and encouraging comments from the discussion group at EuroPLoP 2009.

References

1. B. Bederson. Fisheye Menus. In *ACM Conference on User Interface Software and Technology (UIST)*, pages 217–226. ACM Press, Suracuse, NJ, USA.
2. M. Bezold. User Modeling from Basic Events in Interactive Systems for Intelligent Environments. In *International Conference on Intelligent Environments (IE)*, pages 319–326, 2009.
3. J. Borchers. *A Pattern Approach to Interaction Design.* Wiley, Chichester, UK, 2001.

4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented Software Architecture*. Wiley, Chichester, UK, 1996.
5. J. Danculovic, G. Rossi, D. Schwabe, and L. Miaton. Patterns for Personalized Web Applications. In *European Conference on Pattern Languages of Programs (EuroPLoP) 2001*, pages 423–436, 2001.
6. S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. J. Molina. Perspectives on HCI patterns: Concepts and Tools. In *Extended Abstracts on Human Factors in Computing Systems (CHI)*, pages 1044–1045. ACM, 2003.
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Upper Saddle River, NJ, USA, 1995.
8. S. Goronzy, R. Mochales, and N. Beringer. Developing Speech Dialogs for Multimodal HMIs Using Finite State Machines. In *9th International Conference on Spoken Language Processing (Interspeech), CD-ROM*, 2006.
9. J. O. Hallstrom and N. Soundarajan. Formalizing Design Patterns: A Comprehensive Contract for Composite. In *Proceedings of the 7th FSE Workshop on the Specification and Verification of Component-Based Systems*, pages 77–82, 2008.
10. L. Hassel and E. Hagen. Adaptation of an automotive dialogue system to users expertise and evaluation of the system. *Computers and the Humanities*, 40(1):67–85, 2006.
11. S. Henninger and P. Ashokkumar. An Ontology-Based Metamodel for Software Patterns. In K. Zhang, G. Spanoudakis, and G. Visaggio, editors, *Conference on Software Engineering & Knowledge Engineering (SEKE) 2006*, pages 327–330, 2006.
12. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, World Wide Web Consortium, May 2004.
13. A. Jameson. *Human-computer Interaction Handbook*, chapter Adaptive Interfaces and Agents, pages 305–330. Erlbaum, Mahwah, NJ, USA, first edition, 2003.
14. N. Koch and G. Rossi. Patterns for Adaptive Web Applications. In *European Conference on Pattern Languages of Programs (EuroPLoP) 2002*, pages 179–194. Universitätsverlag Konstanz, 2002.
15. P. Langley. User Modeling in Adaptive Interfaces. In *Conference on User Modeling (UM) 1999*, pages 357–370. Springer-Verlag, New York, NY, USA, 1999.
16. T. Mikkonen. Formalizing Design Patterns. In *Proceedings of the 20th international conference on Software engineering (ICSE)*, pages 115–124, Washington, DC, USA, 1998. IEEE Computer Society.
17. M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. Technical report, W3C, 2004.
18. J. Tidwell. *Designing Interfaces*. O'Reilly Media, Sebastopol, CA, USA, first edition, 2005.
19. M. van Welie and G. C. van der Veer. Pattern Languages in Interaction Design: Structure and Organization. In *Interact 2003*. IOS Press, Amsterdam, The Netherlands, 2003.
20. C. Zannier and F. Maurer. Tool Support for Complex Refactoring to Design Patterns. In M. Marchesi and G. Succi, editors, *Extreme Programming and Agile Processes in Software Engineering (XP)*, volume 2675 of *Lecture Notes in Computer Science*, pages 123–130. Springer, Heidelberg, Germany, 2003.
21. I. Zukerman and D. W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):5–18, 2001.