Furnishing HCI Patterns to Support Modeling and Generation of Interactive User Interfaces

Jürgen Engel, Christian Herdin, Christian Märtin

Augsburg University of Applied Sciences
Faculty of Computer Science

{Juergen.Engel, Christian.Herdin, Christian.Maertin}@hs-augsburg.de

Abstract

The construction of interactive software typically requires the skills of software developers and HCI specialists who cooperate intensively with platform and marketing experts in order to arrive at solutions with the required high levels of software quality, usability, and user experience. The combination of model-based user interface development practices with pattern-based approaches that specify HCI- and software-patterns in a formalized way and respect emerging standards, can facilitate and automate the software process, reduce the development costs, and lead to solutions that can easily be adapted to varying contexts and target devices. This paper highlights the capabilities of the PaMGIS framework to facilitate the construction of abstract user interface models. It is discussed, how pattern descriptions that capture important parts of the design knowledge should be organized in order to be automatically processed during the modeling process.

1 Introduction

Interactive software has become an indispensable ingredient of modern human life. Independent from time and location, people are used to interact with products built around interactive software components, such as web applications, telecommunication devices, car navigation systems, smart home appliances, or other electronic equipment. Nowadays software products running on a variety of devices with individual intrinsic potentialities and limitations demand high usability as well as consistent and appealing user experience. Suppliers encounter tough competition and must look after cost effec-

tiveness as well as time-to-market, which is crucial for their business. It is nearly impossible to meet all requirements simultaneously when exercising traditional software engineering and development processes. A promising way out of this dilemma is the application of a model-driven approach that allows for describing the particular aspects of the intended user interface by means of distinct models at different abstraction levels. Model-driven techniques are well accepted by the software engineering community and various model-based user interface development environments (MB-UIDE) have been introduced.

We have combined model-based user interface development practices with pattern-based approaches that specify HCI-patterns in a formalized way and respect emerging standards. Result is the framework for Pattern-based Modeling and Generation of Interactive Systems (PaMGIS) (cf. Engel & Märtin 2009) which is a MB-UIDE designed according to the CAMELEON Reference Framework (cf. Calvary et al. 2002). It helps to facilitate and automate parts of the software development process and leads to solutions that can easily be adapted to varying contexts and target devices.

2 Related Work

Model-based user interface development environments (MB-UIDE) introduce models to the development process of interactive applications. Various established MB-UIDE and model-driven approaches that can be used for facilitating the development process for interactive systems are discussed in da Silva (2001), Engel et al. (2014), and Märtin et al. (2013). The models used by these approaches can be task- or object-oriented and contain functional domain and data requirements at different abstraction levels for the interactive system under development. Models used by MBUID environments typically are also used for mapping and linking the functional requirements of the business logic to the different abstract and concrete representations of the user interface to guarantee user interface quality, usability, and good user experience for the user of the final interactive application. The role of the various models used in MB-UID environments varies with respect to the modeling purpose. Typically more than one model is exploited during the development process to construct the desired solution interactively or (semi-) automatically.

HCI patterns are a means to document design decisions based on established design solutions or best practice work. Patterns represent a relation between a certain design problem and a solution in a given context. The use of patterns has a number of advantages (cf. Seffah 2010). Patterns are simple and easily readable for designers, developers and researchers, and they are useful for the collaboration between the involved people. Furthermore, patterns are based on established knowledge and capture fundamental principles for good design. In order to ensure a certain standard, patterns are summarized in so-called pattern catalogs (cf. Alexander 1977). A catalog of related patterns that belongs to a common domain is called a pattern language (cf. Seffah 2010) or pattern collection. For allowing for a clear definition and comparison of patterns, so-called schemes were introduced by Alexander (1977). They were grouped into sections of textual and graphical descriptions. However, no standardized description of patterns evolved from these schemes, as numerous pattern catalogs were created based on a different understanding of attributes. To solve this problem, a standardized pattern language was proposed, based on XML: the Pattern Language Markup Language (PLML). PLML unifies and standardizes the schemes of different authors with the help of XML tags. Each XML tag represents a part of the scheme. Version v1.1 of PLML specifies that the documentation of a certain pattern should consist of the following elements: a pattern identifier, name, alias, illustration, descriptions of the respective problem, context and solution, forces, synopsis, diagram, evidence, confidence, literature, implementation, related patterns, pattern links, and management information (cf. Fincher 2003).

3 Combined Model- and Pattern-based Development Approach

The PaMGIS framework combines a model-driven approach and pattern-based development techniques. As depicted in Figure 1, it allows for creation of abstract user interface models (AUI) on the basis of diverse fundamental information about the users, the users' tasks, dialog structure, data architecture, used devices, and environment. An AUI designer can make use of patterns stored in a pattern repository. The AUI is iteratively transformed into a concrete user interface (CUI) model which in turn is used to generate respective user interface source code for the intended system platform.

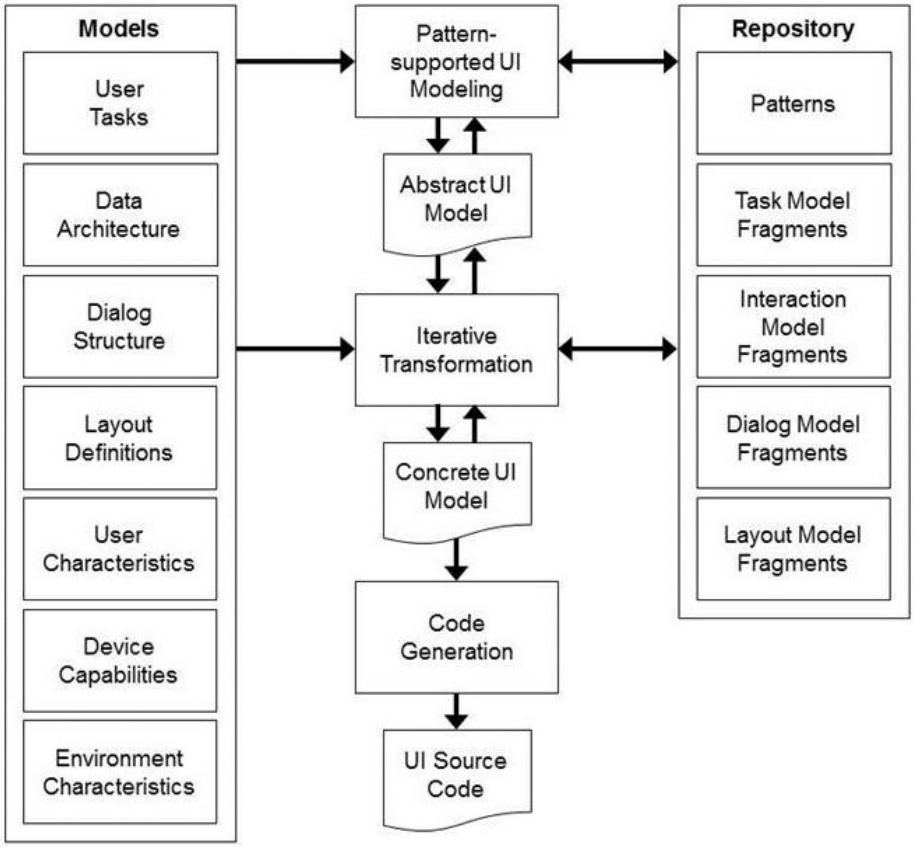

Figure 1 Functional overview of the PaMGIS framework

3.1 PaMGIS Software Development Life-Cycle

The focus of the PaMGIS framework lies on the pattern- and model based construction of the interactive parts, i.e. the user interface, of the software system under development. However, for each non-trivial application the business logic or content parts of the system have to be considered as well. Even if automation for the content parts cannot be driven as far as for the user interface parts, both aspects of the target software system can profit from software patterns during the iterative software life-cycle. In Märtin et al. (2010) and Kaelber & Märtin (2011) it was shown for a media-intensive application from the domain of knowledge based information systems, how both, patterns related to the user interface, and domain patterns for the busi-

ness logic and functionality, can be applied in the overall software life-cycle through all phases from very abstract models to concrete program code.

As current software engineering practice shows, there is more than one accepted software development life-cycle for the overall software system, depending, e.g., on the size of the system under development, the target domain, or the experience background of the software developers. In order to integrate the PaMGIS framework into most accepted software processes from waterfall, spiral model, to agile processes (e.g. Scrum), and various object-oriented methods, we have established so-called embedding-links into our pattern representations that allow for the mapping of abstract, semi-abstract and concrete user interface patterns to parts of the UML object-models that represent the business data and functionality behind the user interface (cf. Märtin et al. 2013). In addition, the PaMGIS framework can easily be coupled with standard development environments like Eclipse to access both, user interface and domain models from the same developer interface.

3.2 PaMGIS Pattern Representation

In order to appropriately support the modeling and UI generation process the patterns are described according to a particular markup language named PaMGIS Pattern Specification Language (PPSL). It is a further development of PLML and remedies some of PLML's inherent weaknesses, notably in terms of pattern relation modeling and provision of details required for automated pattern processing (cf. Engel et al. 2012).

Amongst others, compared to PLML major extensions have been made to the <Implementation> description element, which plays a key role regarding the capability of patterns to facilitate the construction of AUI models. It is now structured to hold fragments of PaMGIS's basic abstract models, including task model, dialog model, and interaction model fragments. Task model fragments are specifications of the pattern-intrinsic user tasks and their relationships based on a modified ConcurTaskTrees (CTT) notation (cf. Paternò 2001). Dialog model fragments contain context-specific definitions of dialogs and their relations based on dialog graphs (cf. Forbrig & Reichart 2007). Finally, the interaction model fragments are kinds of AUI data models and provide abstract specifications of interactive elements and dynamic aspects of the user interface dialogs. These model fragments are intended to be automatically integrated into the overall PaMGIS abstract models as soon as the AUI designer applies them. They can be regarded as building blocks for

the PaMGIS models which speed up the model design process, feature reuse of design work, and positively contribute to high usability and acceptable user experience of the final user interface. In the following more details of how to specify and store the model fragments are provided by means of a tangible example.

4 Example

We illustrate the practical realization of the pattern description element <Implementation> by means of the *Poll* pattern which is published within the pattern collection of Martijn van Welie. The intention of this pattern is to prompt users' opinions regarding a particular topic of the current web site. The original pattern is specified as shown in Table 1 (cf. van Welie 2014).

Table 1: Excerpt of the specification of the "Poll" pattern according to van Welie (2014)

Element	Description
Problem	Users want to state their opinion about a certain statement that is relevant to the site's content.
Solution	List the statements as exclusive options and present the results directly after voting.
Use when	You are designing a site where interaction with the users is desired. Typically this will be a news site or community site where visitors are to be encouraged to share their opinions and improve interactivity.
How	The poll consists of two steps. First the list of options is presented, usually using radio buttons, together with a 'vote' button. After clicking the vote button, the results are displayed. The results include both a percentage and an absolute number.
Why	A poll is a very simple and direct page element that invites users to interact with the site. Users can even do it anonymously so there is no barrier at all to participate. Polls are often linked to content on the site such as articles or products, and the results of a poll can be linked to a discussion in a forum.

We usually employ the pattern in a more generalized way and use it to query users' attitudes towards any matter of fact, not solely towards the web site at hand. Further we do not yet decide the concrete appearance (i.e. radio buttons) of the related user interface element but define an abstract user interface (AUI) element which will be replaced later by a concrete one when

iteratively transforming the AUI to CUI. Finally we slightly change the implied course of action and enable access to the result values not until the vote has been accomplished. At the end of the day the UI must display a question to the user, offer a set of possible answers to that question from which the user can choose exactly one, and provide three different interactor elements to confirm the choice, to request the poll results, and to finally abandon the poll procedure.

In order to populate the <Implementation> description element of the *Poll* pattern according to the PPSL specification it is necessary to analyze the pattern and to specify its inherent portions of the task, dialog, and data models.

4.1 Task Model Fragment

As illustrated in Figure 2, the root element of the task model is an abstract task named “participate in poll” which consists of the three subtasks “vote”, “retrieve results”, and “abandon”. Here, the “retrieve results” task is defined to be optional and therefore is not necessarily executed by the user. The “vote” subtask consists, in turn, of subordinated machine tasks for displaying the question and the possible answers, user tasks for reading the system output and picking the best fitting answer, interaction tasks to select that answer and submit the choice, and an additional machine task to send the data from the user interface to the instance representing the business logic of the system. The “retrieve results” task incorporates an interaction task to request the poll results and three machine tasks to send out that request, to receive the related result data, and to display the values on the screen.

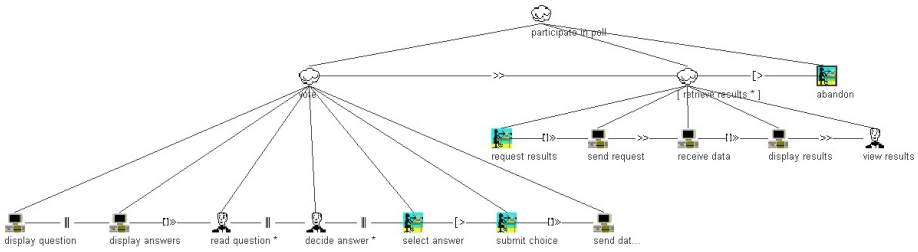

Figure 2 Entire task model fragment of the *Poll* pattern

Amongst others, the user tasks indicate the required cognitive workload of the user and provide information to evaluate whether he/she has been pro-

vided with all data necessary to correctly complete the next process step. Apart from that, they do not considerably contribute to the UI specification and are therefore not considered within the further generation process. Hence, we remove them from the task model fragment. After that, some of the remaining subordinated tasks possess temporal dependencies which are indicated by the temporal operator *enabling with information passing* ([]>>). In this case both tasks deal with the selfsame data element respectively the selfsame set of data. Therefore, in these cases we can eliminate the respective machine tasks without losing significant information required for the user interface generation process. In the given example, these machine tasks are *display answers*, *send data*, *send request*, and *receive data*. The remaining task model fragment is depicted in Figure 3.

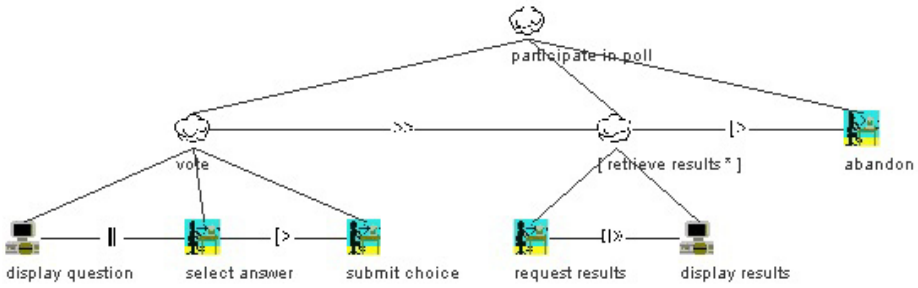

Figure 3 Task model fragment of the *Poll* pattern further reduced by redundant machine tasks

The XML representation of this task model fragment is shown in Figure 4. The notation is a variant of the XML-compliant CTT notation. It basically specifies all involved task elements by defining their related properties, such as unique identifier, name and type of the task, whether the task is optional and/or iterative, the root element of the current sub-tree (called *Parent*), and, if applicable, the predecessor and successor task elements (called *SiblingLeft* and *SiblingRight*) together with their temporal dependencies. Further all task/sub-task relations are expressed within the model. Details of the top element of the task model fragment accentuated in bold (lines 5 to 8) have to be adjusted as soon as the task model fragment is integrated into the overall task model of the intended user interface.

```
<Fragment Type="TaskModel" Identifier="TMF_0001">
  <Task Identifier="TSK_0101" Category="abstraction" Iterative="false"
        Optional="false" Frequency=" ">
    <Name>participate in poll</Name>
    <TemporalOperator name=""/>
    <Parent name=""/>
    <SiblingLeft name="" TempOp="Interleaving"/>
    <SiblingRight name=""/>
    <SubTask>
      <Task Identifier=TSK_0102" " Category="abstraction" Iterative="false"
            Optional="false" Frequency=" ">
        <Name>vote</Name>
        ...
        <IeRef>UIE_0011</IeRef>
      </Task>
      ...
    </SubTask>
  </Task>
</Fragment>
```

Figure 4 Excerpt of the XML representation of the *Poll* pattern's task model fragment

The <TemporalOperator>, <Parent>, <SiblingLeft>, and <SiblingRight> elements of the root task can be automatically aligned to the circumstances of the overall task model. The <TempOp> attribute of the <SiblingLeft> element is designated to be placed in the <TemporalOperator> element of the current left sibling task and subsequently deleted from the task model fragment. After the integration of the fragment into the overall task model it might become necessary to adjust the types of the respective parent tasks, i.e. to make them abstract tasks. However, also this type conversion can be performed automatically. In addition, each task representing a leaf within the task tree is equipped with exactly one interaction object that is specified by means of the <IeRef> tag. It implements a respective link between the task and the corresponding interaction element specified within the interaction model fragment (see Section 4.3). Each pattern usually possesses one particular task model fragment.

4.2 Dialog Model Fragment

Since dialog models can be regarded as platform-specific navigations patterns, they might incorporate several different dialog model fragments. In the context of PaMGIS dialog models are specified by means of dialog graphs (cf. Forbrig & Reichart 2007). Here, dialogs are compositions of relevant tasks and therefore implicitly comprise the interaction elements which are associated with these tasks. The flow between the various dialogs is specified

by arrows, whereupon the arrowheads indicate the direction. Figure 5 shows on its left side a possible dialog graph regarding the application of the *Poll* pattern on a desktop computer with a large display. All tasks involved in the poll (please refer to Figure 3) are combined into one single dialog.

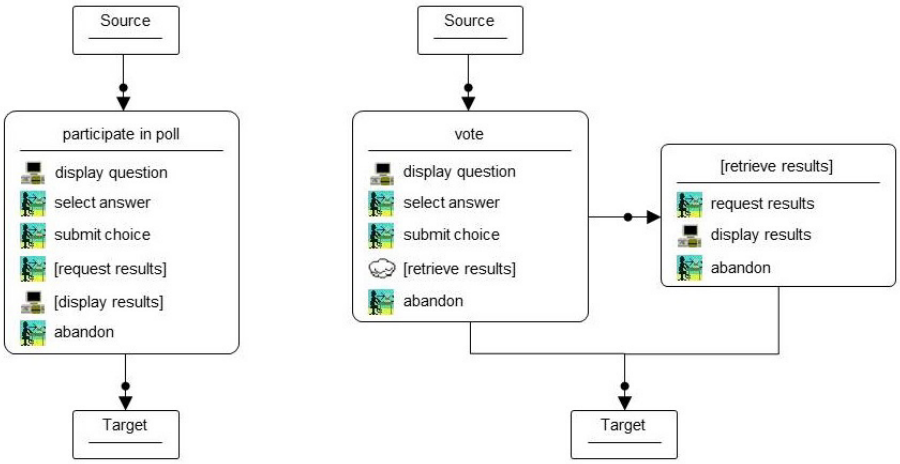

Figure 5 Dialog model fragments of the *Poll* for desktop computers (on the left) and for mobile devices with small displays (on the right)

The right part of Figure 5 illustrates a possible dialog model for a mobile device with limited screen space, such as a smart phone. Due to the display size limits it is not possible or reasonable to display all interaction elements simultaneously. Therefore, we define two different dialogs with transitions between them. The question which tasks to include in which dialog can be answered when having a look at the task model structure. Very often, it is a promising approach to combine the task elements included in the selfsame sub-tree because they are closely related to each other. All of them have to be completed in order to achieve the superordinate goal related to the top element of the task sub-tree. In the present case we decide to implement two dialogs, one for the sub-tree starting with the abstract task *vote* and likewise a second one for the *retrieve results* sub-tree. Since it makes no sense to treat the remaining single interaction task *abandon* in a separate dialog, we assign it to the *vote* dialog. Thus, the user may vote, optionally have a look at the results, and finally abandon the poll process. In case of the user likes to view the results he/she requires an interaction element which allows to navigate to the second dialog. For that purpose, the optional abstract task *retrieve results*

has been added to the *vote* dialog. Figure 6 shows the XML compliant model fragment representation of the dialog for mobile devices as specified above.

```
<Fragment Type="DialogModel"
Identifier"DMF_0002">
  <DmfName>poll mobile</DmfName>
  <Purpose>mobile phone</Purpose>
  <Dialogs>
    <Dialog Identifier="DLG_1002">
      <DName>vote<DName>
      <Coverage>
        <Task>
          <TaskID>TSK_0102</TaskID>
          <TName>vote</TName>
          <Processing>recursive</Processing>
        </Task>
        <Task> ... </Task>
      </Coverage>
    </Dialog>
    <Dialog Identifier="DLG_1003">
      <DName>results<DName>
      ...
    </Dialog>
  </Dialogs>
  <DialogFlow>
    <Dlg>
      <DlgID>DLG_1002</DlgID>
      <Predecessors>
                                        <Predecessor Type="">
                                          <DID></DID>
                                          <Trigger></Trigger>
                                        </Predecessor>
                                      </Predecessors>
                                      <Successors>
                                        <Successor
                                        Type="sequential">
                                          <DID>DLG_1003</DID>
                                          <Trigger>
<IeRef>UIE_17</IeRef>
                                            <Event></Event>
                                          </Trigger>
                                        </Successor>
                                        <Successor Type="">
                                          <DID></DID>
                                          <Trigger>
<IeRef>UIE_16</IeRef>
                                            <Event></Event>
                                          </Trigger>
                                        </Successor>
                                      </Successors>
                                    </Dlg>
                                    ...
                                  </DialogFlow>
                                </Fragment>
```

Figure 6 Excerpt of the XML representation of the *Poll* pattern's dialog model fragment for mobile devices

Within its upper part every involved dialog is described by specifying a unique identifier, a name, and a list of all task elements covered by the dialog. The latter are represented as links to the task model fragment. By means of the <Processing> element it is indicated whether solely the mentioned subtask itself (exclusive) or also all subtasks shall also be included in the dialog specification (recursive). In the following part the flow between these dialogs is configured. This can be done by defining the predecessors and successors of each dialog and specifying the triggers which cause the transitions between the related dialogs. When integrating the dialog model fragment into the overall dialog model of the intended user interface the parts marked in bold related to predecessor and successor have to be adjusted. Some information will be added at a later point in time, i.e. the <Event> element will be specified not before the transformation of AUI to CUI elements.

Since the added abstract task is not a leaf in the task tree, it does not possess a reference to an interaction element within the interaction model fragment. Therefore, we have to introduce one within the dialog model (see <IeRef> to interaction element with identifier *UIE_0017* inside the definition of the <Trigger> element in the bold marked lines 48 to 51).

After having viewed the poll results we do not like to make users navigate back to the *vote* dialog before they are able to finalize the poll process. For that reason we added the interactive task *abandon* also to the *result* dialog.

4.3 Interaction Model Fragment

As already explained above, the interaction model fragment plays the role of a data model and provides the specifications of the required abstract user interface (AUI) elements which are referenced inside the task model and dialog model fragments. The AUI meta model of PaMGIS is very similar to the one described in Paternò et al. (2009). It specifies AUI elements as summarized in Table 2.

Table 2: Overview of PaMGIS AUI elements

AUI Element	Description
Activator	activate a user interface element or call a function
Navigator	navigate to a different dialog
Output	display objects of diverse data types for read-only purposes
Editor	manipulable objects of diverse data types
singleChoice	select one of several options
multiChoice	select none, one or more of several options

The XML representation of the interaction model fragment of the *Poll* pattern is illustrated in Figure 7.

```
<Fragment Type="InteractionModel" Identifier"IMF_0001">
  <InteractionElements>
    ...
    <InteractionElement Identifier="IE_0016" Visible="true" Enabled="true"
      <Name>userAction_Abandon</Name>
      <Type>Navigator</Type>
      <Label>Abandon</Label>
    </InteractionElement>
    <InteractionElement Identifier="IE_0017" Visible="true" Enabled="true"
      <Name>userAction_Retrieve_Results</Name>
      <Type>Navigator</Type>
      <Label>Retrieve results</Label>
    </InteractionElement>
  </InteractionElements>
</Fragment>
```

Figure 7
Excerpt of the XML representation of the *Poll* pattern's interaction model fragment

5 Discussion

In this paper, we have outlined our approach to specify HCI patterns formally and exploit them in order to facilitate user interface modeling and subsequent support semi-automated pattern processing. Starting with PLML version 1.1 we developed the markup language PPSL which allows for incorporating all required information into the pattern specifications. Amongst other features, we textured the so far unstructured <Implementation> description element in a way that task model, dialog model, and interaction model fragments can be stored within the patterns. These model fragments can be automatically integrated into the overall PaMGIS abstract models as soon as a pattern is applied. They serve as building blocks for the constitutive models and accelerate the model design process, feature reuse of design work, and effectively contribute to high usability and acceptable user experience of the final user interface.

References

Alexander, C. et al. (1977): *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

Calvary, G. et al. (2002): The CAMELEON Reference Framework, Document D1.1 of the CAMELEON R&D Project IST-2000-30104.

da Silva, Paulo Pinheiro (2001): User Interface Declarative Models and Development Environments: A Survey. In: *DSV-IS'00 Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems*. S. 207–226.

Engel, J. & Märtin, C. (2009): PaMGIS: A Framework for Pattern-based Modeling and Generation of Interactive Systems. In: *Proceedings of HCI International '09*. San Diego, USA, 826–835.

Engel, J. et al. (2012): Pattern-based Modeling and Development of Interactive Information Systems. In: Frotschnig, A. & Raffaseder, H. (Eds.): *Forum Medientechnik – Next Generation, New Ideas : Beiträge der Tagung 2012 an der Fachhochschule St. Pölten*. Glückstadt: Verlag Werner Hülsbusch, S. 155–167.

Engel, J. et al. (2014): Evaluation of Model-based User Interface Development Approaches. In: *Human-Computer Interaction, Theories, Methods, and Tools, HCII 2014* (LNCS 8510). S. 295–307.

Fincher, S. et al. (2003): Perspectives on HCI patterns: concepts and tools. In: *CHI'03 Extended Abstracts on Human Factors in Computing Systems*. New York: ACM, S. 1044–1045.

Forbrig, P. & Reichart, D. (2007): Spezifikation von „Multiple User Interfaces" mit Dialoggraphen. In: *Processdings of INFORMATIK 2007: Informatik trifft Logistik.* Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Bremen. S. 449–453.

Kaelber, C. & Märtin, C. (2011): From Structural Analysis to Scenarios and Patterns for Knowledge Sharing Applications. In: *Human-Computer Interaction, Design and Development Approaches, HCII 2011* (LNCS 6761). Heidelberg: Springer, S. 258–267.

Märtin, C. et al. (2010): Using HCI-Patterns for Modeling and Design of Knowledge Sharing Systems. In: P. Forbrig & H. Günther (Eds.): *Perspectives in Business Informatics Research, BIR 2010* (LNBIP 64). Heidelberg: Springer, S. 1–13.

Märtin, C. et al. (2013): Patterns and Models for Automated User Interface Construction – In Search of the Missing Links. In: M. Kurosu (Ed.): *Human-Computer Interaction, Part I, HCII 2013* (LNCS 8004). Heidelberg: Springer, S. 401–410.

Paternò, F. (2001): *ConcurTaskTrees: An Engineered Approach to Model-based Design of Interactive Systems*. Pisa: ISTI-C.N.R.

Paternò, F. et al. (2009): Model-based Design of Multi-device Interactive Applications Based on Web Services. In: *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I.* Berlin/Heidelberg: Springer, S. 892–905.

Seffah, A. (2010): The evolution of design patterns in HCI: from pattern langauges to pattern-oriented design. In: *Proceedings of the 1st Interational Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS'10)*. ACM. S. 4–9.

van Welie, M. (2014). Patterns in Interaction Design: http://www.welie.com <Sept. 2nd, 2014>.