Formal Pattern Specifications to Facilitate Semi-automated User Interface Generation

Jürgen Engel[1,2], Christian Märtin[1], Christian Herdin[1], and Peter Forbrig[2]

[1] Augsburg University of Applied Sciences, Faculty of Computer Science,
An der Hochschule 1, 86161 Augsburg, Germany
{Juergen.Engel,Christian.Maertin}@hs-augsburg.de
[2] University of Rostock, Institute of Computer Science,
Albert-Einstein-Strasse 21, 18059 Rostock, Germany
Peter.Forbrig@uni-rostock.de

Abstract. This paper depicts potentialities of formal HCI pattern specifications with regard to facilitate the semi-automated generation of user interfaces for interactive applications. In a first step existing proven and well accepted techniques in the field of model-based user interface development are highlighted and briefly reviewed. Subsequently it is discussed how we combine model-based and pattern-oriented methods within our user interface modeling and development framework in order to partly enable automated user interface generation. In this context a concrete pattern definition approach is introduced and illustrated with tangible examples from the domain of interactive knowledge sharing applications.

Keywords: HCI patterns, model-based user interface development, pattern-based development, formalized pattern specification, user interface generation.

1 Introduction

There are many valuable pattern collections available for user interface (UI) designers and software developers. However, most patterns lack standardized specification and are therefore hard to retrieve and often impractical to use. Due to this fact the Pattern Language Markup Language (PLML) has been introduced in the year 2003. But PLML in turn shows clear weaknesses when patterns are intended to be used for (semi-)automated UI generation. Therefore, we started from PLML as a basis and made several changes and enhancements to support automatic pattern processing. These efforts predominantly focus on features conveying pattern relationship modeling and provision of means for automated pattern treatment. This paper deals with well-known and widely accepted model-based techniques and how they can be combined with a pattern-based approach where emphasis is on the structured and formal specification of HCI patterns.

2 Related Work

Patterns were originally introduced by Christopher Alexander in 1977 as a means to accomplish reuse when solving problems in architecture and urban planning [1]. Eighteen years later, the pattern concepts were translated to the domains of software architecture and software engineering by the Gang of Four (GoF) [11]. Nowadays patterns are also applied to the fields of HCI [8], user experience (UX) [19], usability engineering [13], task modeling [10], and application security [21].

There exist many widely accepted pattern collections, for instance the ones of Jenifer Tidwell [18], Martijn van Welie [20], or Douglas van Duyne [5]. However, different pattern authors usually describe their patterns in different and inconsistent styles. This can be regarded as a clear shortcoming of patterns, because this makes it difficult or even impossible to search, choose and reference patterns across the various pattern collections. In a workshop held within the context of the CHI 2003 conference the participants aimed at unification of pattern descriptions and guidance for the authors. Hence the Pattern Language Markup Language (PLML) version 1.1 was constituted. According to PLML 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 [8].

In [7] it is concluded that it is possible to map the pattern descriptions contained in the previously mentioned pattern collections into PLML compliant formats, however this cannot be done in a fully automated manner.

Extensions and changes were suggested in PLML version 1.2 [4]. These efforts strived to make PLML more feasible for Management of User Interface Patterns (MUIP). A second development is PLMLx [3]. Additional pattern description elements are introduced, including organization, resulting context, and acknowledgements. Further the <Management information> element is being extended and the <Example> and <Rationale> elements are separated from each other. A third approach is the XPLML framework which can be regarded as a bundle of specifications and tools to formalize HCI patterns. The framework is intended to close the gap between the textual pattern specifications and their application in user interface software. The XPLML framework is implemented on the basis of seven modules: unified HCI pattern form, semantic metadata, semantic relations among patterns, atomic particles of HCI design patterns, requirements engineering in HCI community, survey of HCI design pattern management tools, and specification documentation.

The basic idea to support user interface designers and software developers with a combination of both, model-based techniques and pattern-based methods is realized by the integrated framework for pattern-based modeling and generation of interactive systems (PaMGIS) [6]. PaMGIS is developed by the Automation in Usability Engineering group (AUE) at Augsburg University of Applied Sciences. As illustrated in Figure 1, this framework allows for creation of abstract user interface models (AUI) on the basis of diverse fundamental information about the users, the users' tasks, used devices, and environment. Additionally the AUI designer can make use of patterns stored in a pattern repository. The AUI is iteratively transformed into a semi-abstract

UI model which in turn is used to generate respective user interface source code. The framework has been continuously improved. Patterns are now available in a modified PLML format and seamless pattern hierarchies can be modeled [7].

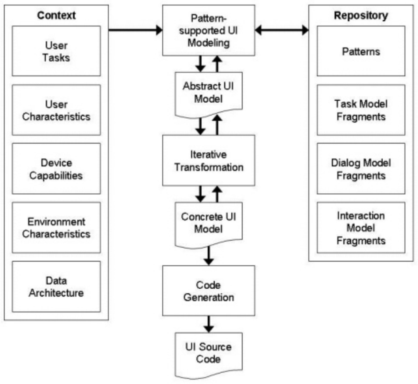


Fig. 1. Functional overview of the PaMGIS framework

Task models play an important role in the area of HCI in general and, in particular, for model-based user interface development. They represent the logical activities of users executed in order to reach their goals [16]. Therefore, knowing the necessary user tasks is fundamental to the design process [15]. A well-known approach for representing task models is ConcurTaskTrees (CTT). CTT provides a graphical syntax and is organized in a strictly hierarchical structure, so that complex tasks can be iteratively decomposed into less complex subtasks until a certain level of granularity is reached. Thus, the logical structure of the task models is represented in a tree-like manner. CTT distinguishes four different task types, i.e. user, interaction, application, and abstract tasks. Temporal relationships between tasks can be expressed by a variety of temporal operators, i.e. hierarchy, enabling, choice, enabling with information passing, concurrent tasks, concurrent communicating tasks, task independence, disabling, and suspend-resume. Additionally tasks can be defined as optional or iterative [15]. ConcurTaskTrees are used for task model specifications within the PaMGIS framework. An illustrated example can be viewed in chapter 3.

Besides task models, dialog models comprise essential information for user interface generation. Dialogs can be directly derived from the related task model [2]. Here, it is assumed that all tasks which are active at a certain point in time are to be visualized within a common dialog. This can be regarded feasible for relatively small task models, but fails for more complex models since related user interfaces tend to be overloaded [9]. This phenomenon can be avoided by explicitly designing navigation specifications on the basis of abstract dialog graphs [17] and assigning individual tasks of the task model to the dialog specification nodes. Using this technique it is possible to define platform-specific navigation models [9]. The nodes of the dialog graph represent dialogs of different types, i.e. single, multiple, modal, and complex. The edges indicate whether dialog transitions are of sequential or concurrent nature [9]. Dialog graphs are used to define platform-dependent dialog models within the PaMGIS framework. Exemplary dialog graphs are provided in chapter 3.

3 Formal Pattern Specifications

The intension of the PaMGIS framework is to combine model- and pattern-based methods and techniques in order to make user interface modeling and realization more easy and practicable even for users with less development skills. Once the relevant models are available the framework takes the job to at least semi-automatically transform the models iteratively and generate UI source code. One of the basic ideas is also to support the construction of the relevant models by means of patterns. In this sense PLML shows some deficiencies notably in terms of pattern relation modeling and provision of details required for automated pattern processing. Indeed PLML provides relevant description elements, i.e. <Pattern-link> and <Implementation>, but the former lacks of detail for appropriate pattern referencing and the latter is completely unstructured yet. Therefore, we started with PLML version 1.1 and made several changes and enhancements which mainly apply to the specification elements <Pattern ID>, <Pattern-link>, and <Implementation> as illustrated in Table 1. Further we introduced a new element named <Embedding-link> which is highlighted in [14]. The entire structure of the resulting PLML variant which we now call PaMGIS Pattern Specification Language (PPSL) is also summarized in [14].

Table 1. Selected pattern specification elements of the PaMGIS Framework

Specification Element	Brief Description
UPID	Unique pattern identifier
CollectionID	Identifier of the respective pattern collection
PatternID	Pattern identifier
Pattern revision	Revision of referenced pattern
InstanceID	Pattern instance identifier
Pattern-link	Relationship to other patterns or pattern instances
LinkID	Unique link identifier
Link-type	Type of link (i.e. PERMANENT or TEMPORARY)
Relationship-type	Type of relation
Pattern identification	UPID of the respective pattern
Label	Name of the pattern link
Implementation	Code or model fragments or details of technical realization
Task model fragment	Specification of pattern-intrinsic tasks and their relationships based on an modified CTT notation
Dialog model fragment	Context-specific definition of dialogs and their relations based on dialog graphs
Interaction model fragment	Abstract specification of the dynamic aspects of the user interface dialogs

Details of the PLML modifications are discussed and illustrated in the following by means of patterns identified during the p.i.t.c.h. project (pattern-based interactive tools for improved communication habits in knowledge transfers) that was conducted by AUE and two medium-sized enterprises and was focused at the knowledge management domain [12]. Within this context prototypical applications for individual platforms were developed.

3.1 Relationships of Patterns

As already elaborated in [7] automated pattern processing demands adequate and accurate pattern referencing. On one hand, this affects the PLML specification element <Pattern ID> which must allow for exact identification of an individual pattern. On the other hand, <Pattern-link> must be capable to address and describe particular pattern relations. Therefore, we have replaced PLML's <Pattern id> element by <UPID> which now is a composite of identifiers of the relevant pattern collection, the pattern itself, the particular pattern revision, and an individual pattern instance. This allows to distinguish individual pattern entities in the case a pattern is applied more than once in a certain context.

In terms of the PLML specification element <Pattern-link> there is a need to distinguish two fundamental types of pattern links. First, there exist kinds of permanent links to other patterns, which can be regarded as "hard-coded" and generally will not change for a long period of time. If a permanent link is considered to be changed this would normally lead to a new revision of the pattern. As soon as a respective pattern is applied, all related patterns referenced by permanent links are also applied automatically. Moreover, there must be a possibility to model temporary pattern links in case a relationship to an individual pattern is required just under certain circumstances or in a specific context. Hence we defined a sub-element of <Pattern-link> as outlined in Table 1, i.e. <Link-type>. A descriptive example is given in [7].

3.2 Support for Automated Pattern Processing

In order to equip HCI patterns with information facilitating automated pattern processing and user interface generation we render the so far unstructured PLML element <Implementation> more precisely. For this reason we store relevant task model, dialog model, and interaction model fragments together with the patterns. Thus, we have defined a sub-element of <Implementation> named <Fragment>. Fragments can be regarded as building blocks which can be used to improve the overall user interface model by applying a pattern in the design process.

Task models used within the PaMGIS framework are expressed in CTT syntax. Therefore, the task model fragment of a particular pattern is defined in CTT XML format. Figure 2 shows an excerpt of the task model of the p.i.t.c.h. pattern *Advanced Search* [7].

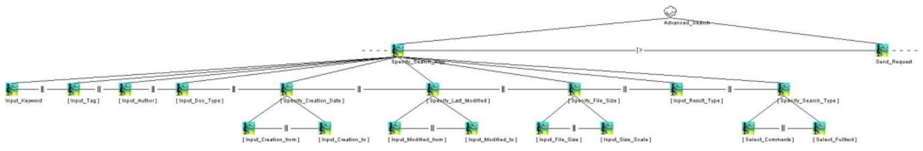


Fig. 2. Excerpt of the task model of the *Advanced Search* pattern

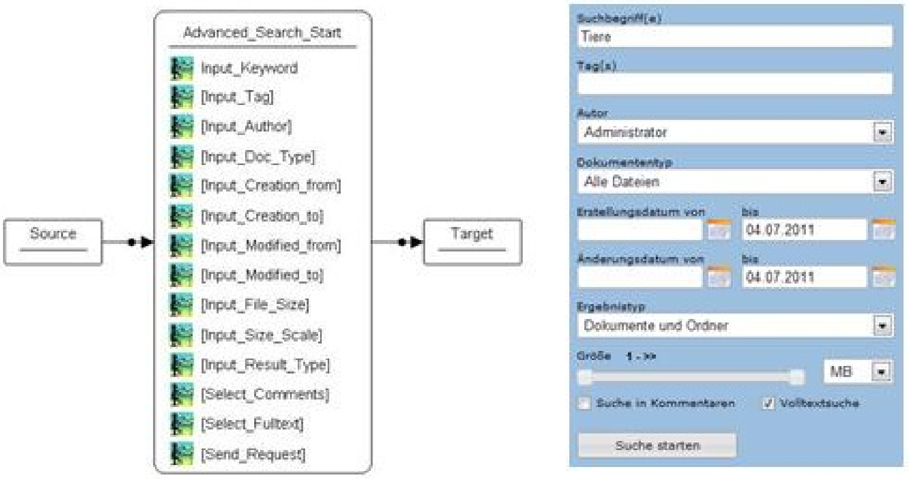
In this example we focus on interaction tasks which directly contribute to the resulting user interface while abstract tasks, user tasks and application tasks are less important within the scope of this paper. We iteratively refine the task model until the leaves of the task tree can be matched to exactly one interaction object. For this purpose we have introduced an additional specification element <IeRef> which establishes a link between the task and a certain interaction element specified within the interaction model fragment (see below). The XML representation of the above task model fragment is sketched in Figure 3.

```
<Fragment Type="TaskModel" Identifier="TMF_0001">
   <Task Identifier="PAS_0001" Category="abstraction" Iterative="false"
         Optional="false" Frequency=" ">
      <Name>Advanced_Search</Name>
      <Parent name=" "/>
      <SiblingLeft name=" " TempOp="Interleaving"/>
      <SiblingRight name=" "/>
      <SubTask>
         …
         <Task Identifier="PAS_0003" Category="interaction"
               Iterative="false" Optional="false" Frequency=" ">
            <Name>Specify_Search_Args</Name>
            <TemporalOperator name="SuspendResume"/>
            <Parent name="Advanced_Search"/>
            <SiblingLeft name="Decide_Search_Args"/>
            <SiblingRight name="Send_Request"/>
            <SubTask>
               <Task Identifier="PAS_0013" Category="interaction"
                     Iterative="false" Optional="false" Frequency=" ">
                  <Name>Input_Keyword</Name>
                  <TemporalOperator name="Interleaving"/>
                  <Parent name="Specify_Search_Args"/>
                  <SiblingLeft name="Decide_Search_args"/>
                  <SiblingRight name="Input_Tags"/>
                  <IeRef>IE_0001</IeRef>
               </Task>
            </Subtask>
         </Task>
         …
      </Subtask>
   <Task>
</Fragment>
```

Fig. 3. XML code snippet of the *Advanced Search* pattern's task model

Here, the subtask *Input_keyword* is linked to an interaction element with ID *IE_0001*. The content of the elements marked in bold have to be calculated and replaced when the pattern is applied respectively the model fragment is integrated into the overall task model. While the <Parent>, <SiblingLeft> and possibly <SiblingRight> elements are to be automatically aligned to the conditions inside the overall task model the data held within the <TempOp> attribute of the <SiblingLeft> element is destined to be moved to the <TemporalOperator> element of the left sibling task and deleted from the task model fragment. Note that dependent on the task types included in the task model fragment adjustments of the type of parent elements might be necessary, i.e. becoming abstract tasks. However, this can be covered automatically, too.

While one pattern usually possesses one particular task model fragment it might include several dialog model fragments. Dialog models represent target platform-specific navigations. In the mentioned example of the *Advanced Search* pattern all subtasks can be assigned to one single dialog on a desktop PC equipped with a large display. The related dialog graph can be viewed on the left and the resulting UI dialog on the right side of Figure 4.



Suchbegriff(e): Tiere
Tag(s):
Autor: Administrator
Dokumententyp: Alle Dateien
Erstellungsdatum von: bis: 04.07.2011
Änderungsdatum von: bis: 04.07.2011
Ergebnistyp: Dokumente und Ordner
Größe 1 - >> MB
☐ Suche in Kommentaren ✓ Volltextsuche
Suche starten

Fig. 4. Possible desktop PC dialog graph (left) and resulting UI (right)

The related XML representation of the PC desktop dialog model fragment is sketched in Figure 5. The assignment of tasks to the dialog is accomplished by means of the <Coverage> specification element.

```
<Fragment Type="DialogModel" Identifier"DMF_0001">
   <DMName>Advanced_Search_Desktop</DMName>
   <Dialog>
      <DID>00010001</DID>
      <DName>Prepare_Advanced_Search<DName>
      <Coverage>
         <Task>
            <TID>PAS_0003</TID
            <TName>Specify_Search_Args</TName>
            <Processing>recursive</Processing>
         </Task>
         <Task>
            <TID>PAS_0004</TID>
            <TName>Send_Request</TName>
            <Processing>exclusive<Processing>
         </Task>
      </Coverage>
   </Dialog>
</Fragment>
```

Fig. 5. XML code snippet of the desktop PC dialog model

The <Processing> element indicates whether solely the mentioned subtask itself (*exclusive*) or all subtasks shall also be included in the dialog specification (*recursive*). In contrast to this simple example the dialog model for mobile phones is more complex because owing to screen size limits the functionality has to be split into several dialogs. The various nodes in the task tree help to compose meaningful groups. Note that not all tasks are incorporated in the mobile dialog model. The respective dialog graph is illustrated on the left of Figure 6. The resulting UI dialog is shown on the right side.

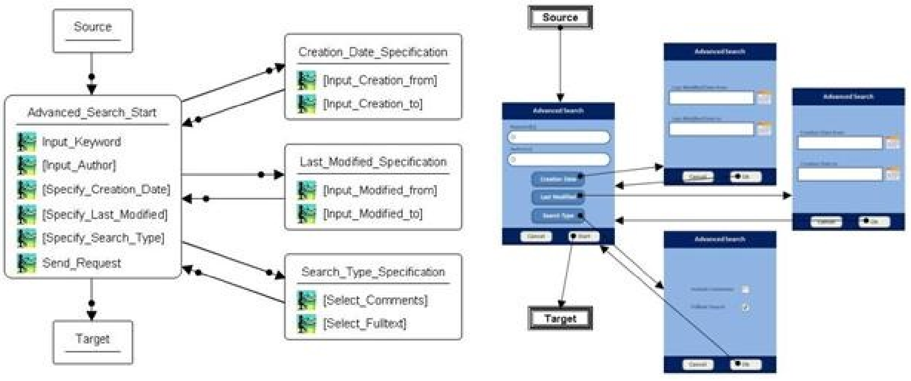


Fig. 6. Possible mobile phone dialog graph (left) and resulting UI (right)

Similar to the desktop version it is necessary to assign the tasks to particular dialogs using the <Coverage> element. But in addition it must be specified when and how a transition to a different dialog shall happen. For this purpose we introduced the <DialogFlow> element which allows for specification of respective successor dialogs and interaction elements triggering the dialog transition. As shown in Figure 7 both, the successor dialog and the interaction element are referenced via appropriate identifiers.

```
<Fragment Type="DialogModel" Identifier"DMF_0002">
   <DMName>Advanced_Search_Mobile</Name>
   <Dialog>
      <DID>00020001</DID>
      <DName>Advanced_Search_Start</DName>
      <Coverage>
         …
      </Coverage>
      <DialogFlow>
         <Successor Type="sequential">
            <DID>00020002</DID>
            <Trigger>
               <IeRef>IE_0101</IeRef>
               <Event>On_Klick</Event>
            </Trigger>
         </Successor>
         …
      </DialogFlow>
   </DialogFlow>
</Fragment>
```

Fig. 7. XML code snippet of the mobile phone dialog model

The three dialogs *Creation_Date_Specification*, *Last_Modified_Specification*, and *Search_Type_Specification* do yet neither directly nor indirectly possess an interaction element that could trigger the transition back to the *Advanced_Search_Start* dialog. This problem is fixed by applying the *OK_Cancel* pattern in each case.

Finally the interaction model fragment contains the abstract specifications of the required interaction elements. The XML definitions of the two previously mentioned interaction elements are illustrated in Figure 8.

```
<Fragment Type="InteractionModel" Identifier"IMF_0001">
   <InteractionElement Identifier="IE_0001" Visible="true"
                       Enabled="true" Optional="false">
      <Name>userInput_Keyword</Name>
      <Type>InputField</Type>
      <DataType>String</DataType>
      <Label>Keyword(s)</Label>
   </InteractionElement>
   <InteractionElement Identifier="IE_0101" Visible="true"
                       Enabled="true" Optional="false">
      <Name>userAction_CreationDate</Name>
      <Type>TransitionActivator</Type>
      <Event>OnKlick</Event>
      <Label>Creation Date</Label>
   </InteractionElement>
</Fragment>
```

Fig. 8. XML code snippet of the interaction model fragment

Some of the elements specified in the interaction model address user inputs and system outputs, e.g. the interaction element *userInput_Keyword* specified in the XML code snipped above. Such elements can be regarded as interface to the underlying business logic of the particular software.

During the model transformation process the defined abstract interaction elements are substantiated until they can be mapped to the particular widget sets appropriate for the present context of use and available on the target platform. For instance, an abstract *TransitionActivator* might become a link in a browser-based application or a button in a Windows-based fat client.

4 Conclusion

In this paper, we have introduced our approach to specify HCI patterns formally in order to support automatic respectively semi-automated pattern processing and user interface generation. We took PLML version 1.1 as basis and reworked the mechanisms for appropriate modeling of relationships between patterns, i.e. the specification elements <Pattern id> and <Pattern-link>. Additionally we have structured the <Implementation> element in order to hold fragments of task, dialog, and interaction models which can be used during the user interface model design process and for UI generation purposes. These enhancements are explained and illustrated by means of examples from experimental applications in the knowledge sharing domain.

In our current research we focus on further improvements of model design and code generation automation.

References

1. Alexander, C., et al.: A pattern language. Oxford University Press (1977)
2. Berti, S., et al.: A transformation-based environment for designing multi-device interactive applications. In: Proceedings of the 9th International Conference on Intelligent User Interfaces, Funchal (January 2004)
3. Bienhaus, D.: PLMLx Doc. (2004), `http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html` (last website call on February 3, 2012)
4. Deng, J., Kemp, E., Todd, E.G. (Hg.): Focusing on a standard pattern form: the development and evaluation of MUIP. In: Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: Design Centered HCI (2006)
5. van Duyne, D., Landay, J., Hong, J.: The Design of Sites, Patterns for Creating Winning Websites, 2nd edn. Prentice Hall International (2006) ISBN 0-13-134555-9
6. Engel, J., Märtin, C.: PaMGIS: A Framework for Pattern-based Modeling and Generation of Interactive Systems. In: Jacko, J.A. (ed.) HCI International 2009, Part I. LNCS, vol. 5610, pp. 826–835. Springer, Heidelberg (2009)
7. Engel, J., Märtin, C., Herdin, C.: Exploiting HCI Pattern Collections for User Interface Generation. In: Proceedings of PATTERNS 2012, the 4th International Conferences of Pervasive Patterns and Applications, Nice, France, pp. 36–44 (2012)
8. Fincher, S., et al.: Perspectives on HCI patterns: concepts and tools. In: CHI 2003 Extended Abstracts on Human Factors in Computing Systems, Ft. Lauderdale, Florida, USA, pp. 1044–1045. ACM (2003)
9. Forbrig, P., Reichart, D.: Spezifikation von "Multiple User Interfaces" mit Dialoggraphen. In: Processdings of INFORMATIK 2007: Informatik Trifft Logistik, Beiträge der 37, Bremen. Jahrestagung der Gesellschaft für Informatik e.V., GI (September 2007)
10. Gaffar, A., et al.: Modeling patterns for task models. In: TAMODIA 2004 Proceedings of the 3rd Annual Conference on Task Models and Diagrams. ACM, New York (2004)
11. Gamma, E., et al.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
12. Kaelber, C., Märtin, C.: From Structural Analysis to Scenarios and Patterns for Knowledge Sharing Applications. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 258–267. Springer, Heidelberg (2011)
13. Marcus, A.: Patterns within Patterns. Interactions 11(2), 28–34 (2004)
14. Märtin, C., Herdin, C., Engel, J.: Patterns and models for automated user interface construction – in search of the missing links. In: Kurosu, M. (ed.) Human-Computer Interaction, Part I, HCII 2013. LNCS, vol. 8004, pp. 401–410. Springer, Heidelberg (2013)
15. Paternò, F.: ConcurTaskTrees: An Engineered Approach to Model-based Design of Interactive Systems. ISTI-C.N.R., Pisa (2001)
16. Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Springer, London (2000)
17. Schlungbaum, E., Elwert, T.: Dialogue Graphs – A Formal and Visual Specification Technique for Dialogue Modelling. Springer (1996)
18. Tidwell, J.: Designing Interfaces. Patterns for Effective Interaction Design, 2nd edn. O'Reilly Media Inc. (2011) ISBN 978-1-449-37970-4
19. Tiedtke, T., Krach, T., Märtin, C.: Multi-Level Patterns for the Planes of User Experience. In: Proc. of HCI International, July 22-27. Theories Models and Processes in HCI, vol. 4. Lawrence Erlbaum, Las Vegas (2005)
20. van Welie, M.: Patterns in Interaction Design, `http://www.welie.com` (last website call on November 25, 2012)
21. Yoder, J., Barcalow, J.: Architectural patterns for enabling application security. In: International Conference on Pattern Language of Programs, PLoP (1997)