Practical Aspects of Pattern-Supported Model-Driven User Interface Generation

Jürgen Engel[1,2(✉)], Christian Märtin[1], and Peter Forbrig[2]

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

Abstract. Today, highly interactive software has become a crucial ingredient of the daily life. Rapidly evolving technologies and simultaneously increasing user demands make human-computer interaction (HCI) more and more a major criterion for the judgement on success or non-success of software and electronic devices. Contemporary users expect that software products run on a variety of heterogeneous devices with a consistent look and feel, invariable high usability, and a high degree of appealing user experience. In addition, people tend to be impatient and like to have their applications contemporaneously available on their different devices. In order to meet these requirements we have combined a model-driven user interface development process with pattern-based methods in order to realize the best possible reuse when constructing models which subsequently serve as basis for at least semi-automatic user interface generation. This hybrid development approach is the fundament of the PaMGIS framework which is explicated in theory and primarily in practice within the following sections.

Keywords: Model-based user interface development · Pattern-based development · User interface modeling · User interface generation · HCI patterns

1 Introduction

Prior to the design of the current version of the combined model-driven and pattern-based PaMGIS framework a total of 18 model-based user interface development environments (MBUIDE) have been analyzed and evaluated. A résumé of the related results is provided in [5]. In addition, eight user interface description languages (UIDL) have been examined. An overview of the achievements is recapitulated in [4]. The findings of these assessments served as groundwork for the model-driven part of the PaMGIS framework which was designed in the style of the CAMELEON Reference Framework (CRF) [1]. In particular, the ontological domain and context-of-use models are used as proposed by the CRF. However, we decided to split the CRF platform model into a device model and a UI toolkit model. While the former

M. Kurosu (Ed.): HCI 2017, Part I, LNCS 10271, pp. 397–414, 2017.
DOI: 10.1007/978-3-319-58071-5_30

comprises all relevant characteristics of the respective end-user device, the latter holds information about the user interface (UI) elements that are available on the respective underlying software development platform. This avoids redundancies especially in cases where the same software basis supports significantly different devices, e.g. Android on smartphones and tablet computers. A dialog model comprises the dynamic aspects of the future user interface. On the basis of the domain and dialog models an abstract user interface model is derived which is subsequently transformed into a concrete and lastly into a final user interface model. The various user interface model transformations are affected by the information which is deposited within the context of use model. An overview of the PaMGIS models is given inside the dotted box in the upper part of Fig. 1.

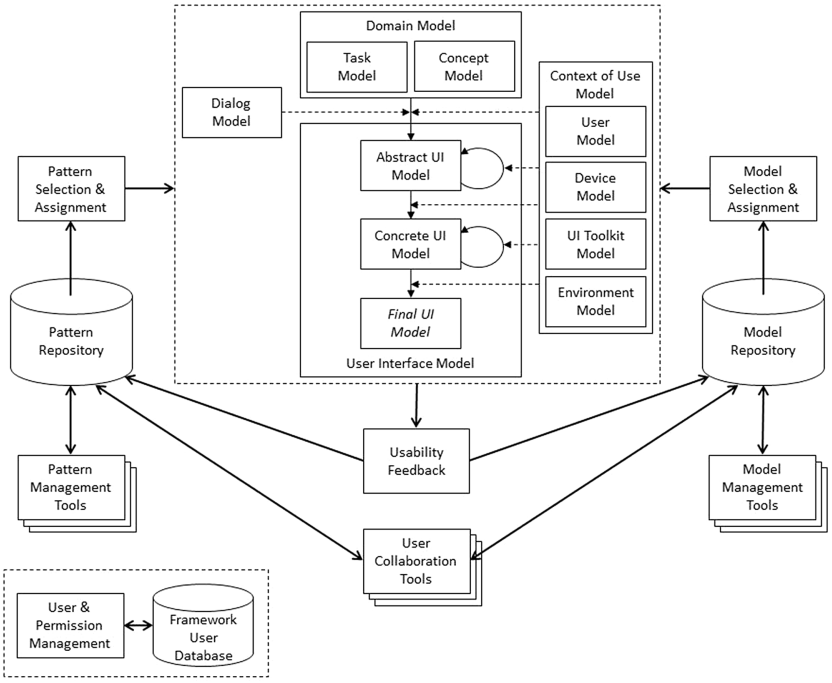

Fig. 1. Logical overview of the PaMGIS framework architecture

In order to alleviate the sometimes complex construction of the models required for user interface generation, the model-driven approach is supported by pattern-based development techniques. For this purpose, we have developed the XML-compliant PaMGIS Pattern Specification Language (PPSL) which was designed to meet two major goals. On the one hand, it shall be possible to easily transform already existing patterns from popular and well-accepted pattern collections, such as *Designing Interfaces* of Jenifer Tidwell [11], *Patterns in Interaction Design* of Martijn van Welie [12], and *Design of Sites* of Douglas van Duyne [2] into the PPSL formalism. On the other

hand, the patterns shall be augmented with any kind of information required for supporting the model-driven user interface generation with PaMGIS.

Regarding the former target, we explored six existing pattern description languages, including PLML 1.1 [8]. Results concerning this matter can be retraced in [6, 7]. Further, five pattern collections were analyzed. An excerpt of the related conclusions is outlined in [6]. We already have demonstrated in [5] that patterns from [2, 11, 12] can be mapped to PLML 1.1 at least in a semi-automatic manner. In [7] it is demonstrated that patterns being specified by means of PLML 1.1 can be translated to PPSL without loss of information.

The basic idea in terms of the latter goal is to equip patterns with pre-assembled model fragments which can be used as building blocks for the construction of the domain and UI models. In addition, certain patterns could also provide valuable input for the various model transformation steps. The complete specification of PPSL is presented in [7].

In preparation of the specification of the tool set required for pattern authoring and management a total of fourteen pattern tools have been analyzed and evaluated. The related results are documented in [3]. The tools for both the pattern-based and model-driven part of PaMGIS are indicated in Fig. 1.

2 The Hybrid Development Process of the PaMGIS Framework

The hybrid development process follows a model-driven methodology which is extended by pattern-based techniques, i.e. patterns are equipped with task, concept, and dialog model fragments which serve as building blocks during the construction of the related models. As soon as a pattern is selected and applied, the fragments are integrated into the overall models. An overview of the development process is provided in Fig. 2. It encompasses nine major steps. Gray-shaded arrows indicate that the associated efforts can generally be supported by the use of patterns.

The domain model constitutes the starting point of the user interface modeling and generation process. Hence, it is to be specified in the very first step. It comprises of two sub-models, i.e. a task and a concept model. The formalism for describing task models is based on the *ConcurTaskTrees* notation [10] which we have extended according to our needs. Subsequently or in parallel, a concept-of-use model is either to be defined or selected from the PaMGIS model repository. This model includes four sub-models for the intended end user device, the software development toolkit, user, and environment. Then, the dynamic aspects of the user interface are determined in a dialog model. The dialogs and their sequence are specified by means of dialog graphs [9]. The next step is to derive the abstract user interface (AUI) from the domain and dialog models. In consideration of the UI toolkit model, the abstract UI elements are to be transformed into concrete ones and context-specific adaptations are carried out, if necessary. Finally, the CUI is to be translated into the final user interface, i.e. coded in the target programming language. Optionally, the resulting source code must be compiled before the user interface can be executed.

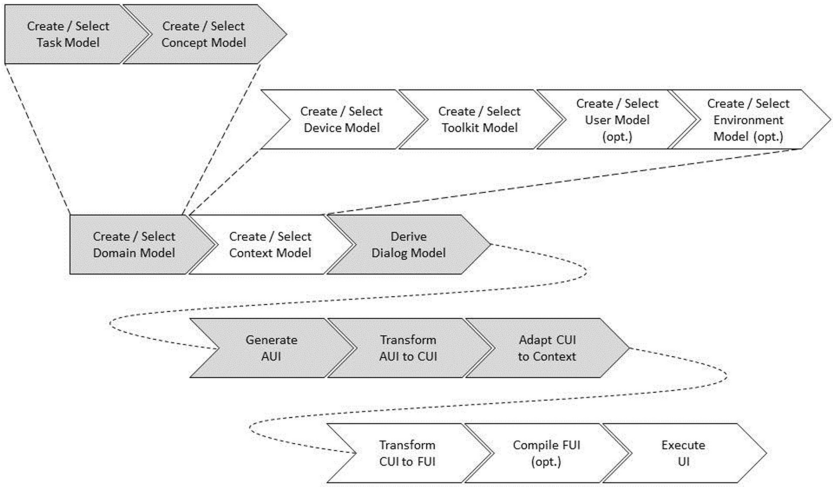

Fig. 2. Development process of the hybrid model-driven and pattern-based approach

In the following sections we describe practical details of (1) how to develop and formalize the various model fragments as integral part of a pattern and (2) how these fragments can be used during the hybrid UI development process.

3 Practical Application of the Hybrid Development Approach

The practical work with the PaMGIS framework is demonstrated using the example of the *Poll* pattern which is part of the pattern collection *Patterns in Interaction Design* [12]. The intention of this pattern is to prompt users' opinions in terms of a specific aspect of the current web site. The original pattern specification is outlined in Table 1.

Before we translated the pattern into the PPSL format we generalized it in a way that it can be used to query users' attitudes towards any matter of fact, not solely toward a particular topic of a web site. Ex ante, we do not decide the concrete appearance of user interface elements, i.e. radio buttons, but employ an abstract user interface (AUI) element which will be replaced by a concrete widget when iteratively transforming the entire AUI model into a concrete user interface (CUI) model. Additionally, we decided to change the implied course of action and enable access to the overall result values not until the vote has been accomplished.

In summary, the UI must display a question to the user and offer a set of possible answers to this question from which the user can choose the one that fits best in his view. Further, three different interaction elements must be provided to confirm the choice, to request the overall results, and to finally quit the poll procedure.

In order to benefit from the pattern within the hybrid development approach it must be analyzed in detail and inherent task, concept, and dialog model portions have to be extracted, formally described, and accommodated in the <Deployment> <PaMGIS> <ModelFragments> section of PPSL.

Table 1. Excerpt of the Poll pattern specification according to [12]

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 radiobuttons, 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

3.1 Task Model Fragment

Within this subsection it is shown how the task model fragment (TMF) of the *Poll* pattern is implemented by means of PPSL.

As illustrated in Fig. 3, the root element of the task model is an abstract task named *poll* which consists of the three subtasks *vote*, *retrieve results*, and *terminate*. 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 application 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 confirm that answer and submit the choice, and an additional application 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 application tasks to

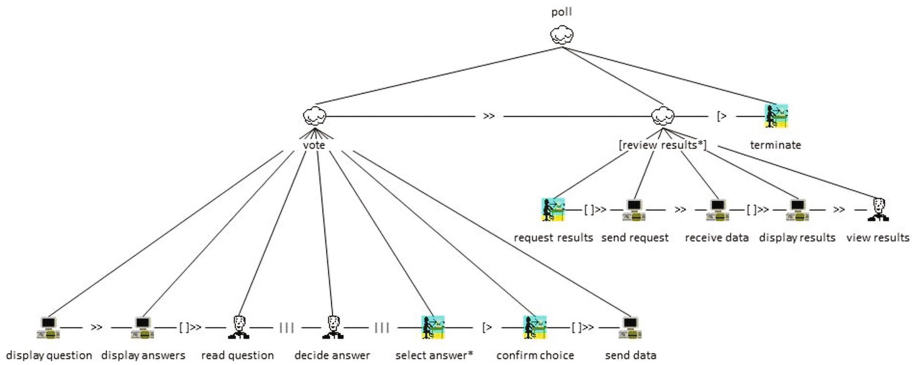

Fig. 3. Entire task model fragment of the *Poll* pattern

send out that request, to receive the related result data, and to display the values on the screen. Finally, the purpose of the *terminate* interaction task is to abandon the poll procedure.

In the following steps this fundamental task model fragment is reduced in order to incorporate solely the information that is relevant for user interface generation. Amongst others, the user tasks indicate the required cognitive workload of the user and provide information to evaluate whether he/she has been provided 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 required for our purposes as illustrated in Fig. 4.

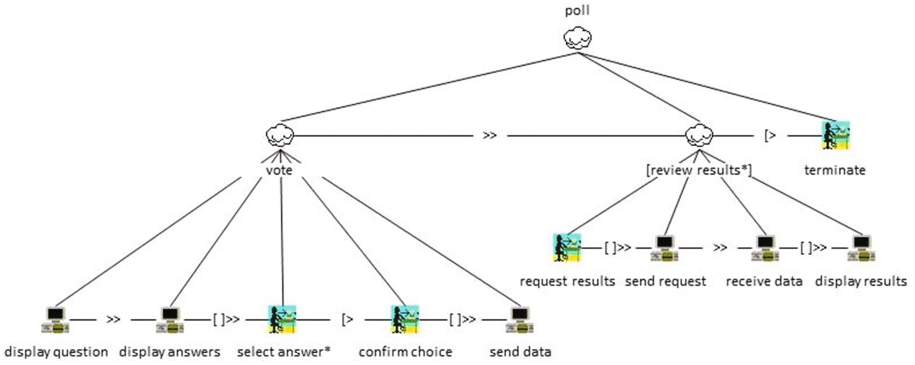

Fig. 4. Task model fragment of the *Poll* pattern reduced by user tasks

Some of the remaining subordinated tasks possess temporal dependencies which are indicated by the temporal operator *enabling with information passing* ([]≫). In this case both adjacent tasks deal with the selfsame data element respectively the selfsame set of data. Therefore, in these cases we can eliminate the respective application tasks without losing significant information required for the user interface generation process. In the given example, these application tasks are *display answers*, *send data*, *send request*, and *receive data*. The remaining TMF is depicted in Fig. 5.

In order to increase the flexibility of the patterns, we introduced a new task type. Besides the standard abstract, user, application, and interaction types of CTT, PaMGIS supports dummy tasks. Please note that such tasks may only appear in task model fragments, but not in task models. Dummy tasks act as templates for variable parts of a TMF and are to be replaced before the pattern is used. If a user selects a pattern by means of the *Pattern Selection & Assignment* function and tries to apply it, the tool will automatically open a dialog and ask whether and how the dummy tasks should be adopted. Here, the predetermined texture of the dummy tasks serves as construction plan, i.e. the resulting task type, the position inside the TMF and the temporal dependencies to sibling tasks. The TMF of the *Poll* pattern including related amendments is shown in Fig. 6.

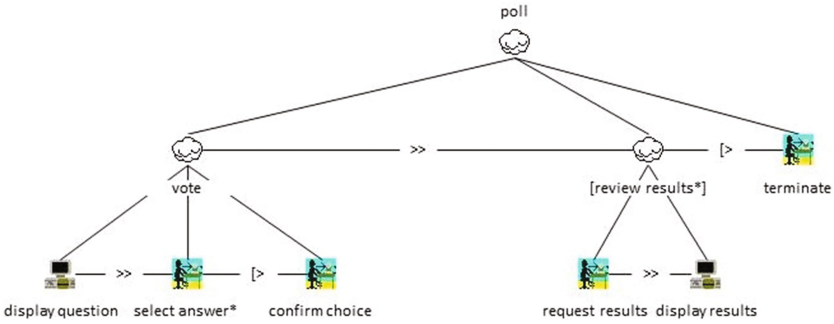

Fig. 5. Task model fragment of the *Poll* pattern further reduced by redundant application tasks

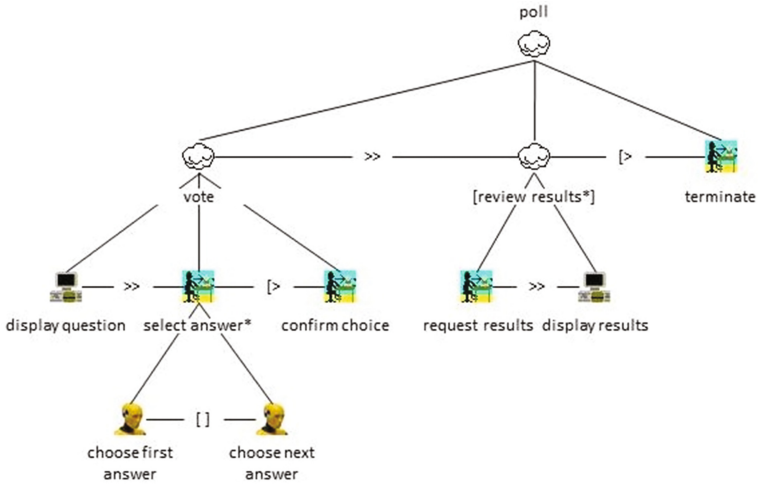

Fig. 6. Task model fragment of the *Poll* pattern supplemented by dummy tasks

The PaMGIS user may change the name of the task interactively within the mentioned dialog and can decide on how often the dummy mechanism is to be repeated. Please note that the concept model fragment of the pattern also contains related dummy concepts which have to be adapted where necessary, too.

In our example the *choose first answer* dummy task is renamed to *choose "excellent"* and *choose next answer* to *choose "fair"*. Further the right dummy task is applied a second time and renamed to *choose "poor"*. The resulting task model fragment is illustrated in Fig. 7. If the user decides not to use the pre-designed dummy tasks, the tool will remove them automatically.

The task model fragment is coded in PPSL format and stored with the pattern. Basically, the same syntax rules apply as for task models. The beginning of this XML compliant representation is provided in Fig. 8. Please note that line numbers have been included for better orientation.

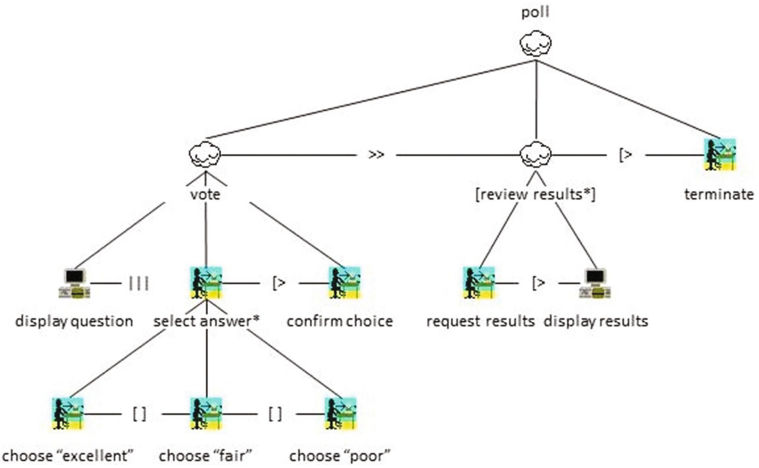

Fig. 7. Resulting task model fragment of the *Poll* pattern after dummy task resolving

When the pattern is applied the task model fragment has to be integrated into the overall task model the user is working on. This can be achieved by simply interlinking the root task of the TMF with the desired sibling tasks and the new parent task inside the task model. The specification of the TMF root element starts at line 10 in Fig. 8.

Once the user has specified the position inside the overall task model where the pattern shall reside this can be performed automatically. In a first step, the entire content of the <TMF_Content> element is copied into the overall task model specification. Subsequently, the identifier and name of the new parent task have to be set accordingly in the <Parent> section of the TMF root element (see lines 25 and 26). If applicable, this must also be done in analogous manner for the left and right siblings (refer to line 29 and 30 resp. 33 and 34). Further, the identifier and name of the pattern's root task (see lines 11 and 12) must be copied into the <RightSibling> section of the preceding and the <LeftSibling> section of the successive task. The temporal relation between the TMF root element and its right sibling is defined straight forward by means of the <TemporalOperator> element (see line 37). A proposed value for the temporal relation between the left sibling and the TMF root element is included in the <TMF_ProposedTempOp> element (see line 08). Here, the user can decide whether to accept or override this suggestion. As a last step, this value must be registered within the <TemporalOperator> element of the left sibling. An exemplary illustration of the resulting task model can be viewed in Fig. 9. Here, the names of the tasks which are not shipped with the TMF are marked with in bold.

3.2 Concept Model Fragment

This subsection shows how the concept model fragment (CMF) of the *Poll* pattern is constructed and formalized using PPSL.

```
01: <ModelFragment>
02: <MDFR_Type>Task</MDFR_Type>
03:<MDFR_FragmentID>"__TMF_0001_01"</MDFR_FragmentID>
04: <MDFR_Label>"Poll"</MDFR_Label>
05: <MDFR_Diagram>"see Fig. 7"</MDFR_Diagram>
06: <MDFR_Fragment>
07: <TMF_IncludesDummy>True</IncludesDummy>
08: <TMF_ProposedTempOp>Interleaving</TMF_ProposedTempOp>
09: <TMF_Content>
10: <Subtask>
11: <TaskID>"__TSK_0001_01_0001"</TaskID>
12: <TaskName>"poll"</TaskName>
13: <TaskDescription>""</TaskDescription>
14: <TaskType>Abstraction</TaskType>
15:     <TaskOrigin>
16:<TaskOriginPatternID>"PPT_0100_0001"</TaskOriginPatternID>
17: <TaskOriginPatternName>"Poll"</TaskOriginPatternName>
18:<TaskOriginPatternVersion>"0001"</TaskOriginPatternVersion>
19:<TaskOriginPatternRevision>"0000"</TaskOriginPatternRevision>
20:</TaskOrigin>
21: <Optional>False</Optional>
22: <Iterative>False</Iterative>
23: <Position>
24: <Parent>
25: <ParentID>""</ParentID>
26: <ParentName>""</ParentName>
27: </Parent>
28: <SiblingLeft>
29: <SiblingLeftID>""</SiblingLeftID>
30: <SiblingLeftName>""</SiblingLeftName>
31: </SiblingLeft>
32: <SiblingRight>
33: <SiblingRightID>""<SiblingRightID>
34: <SiblingRightName>""</SiblingRightName>
35: </SiblingRight>
36: </Position>
37: <TemporalOperator>Interleaving</TemporalOperator>
38: <UIConcepts></UIConcepts>
39: <Subtasks>
40: <Subtask>
41: <TaskID>"__TSK_0001_01_0002"</TaskID>
426: <TaskName>"vote"</TaskName>
……
```

Fig. 8. Excerpt of the XML representation of the TMF of the *Poll* pattern

A concept model comprises high-level specifications of all data elements and interaction objects elements that are relevant for the user interface. Hence, all concepts required by the *Poll* pattern have to be identified. In addition, we have to define the abstract type of the concepts. The most common types are listed in Table 2.

When having a look at the TMF of the *Poll* pattern eight required concepts can be identified.

As illustrated in Fig. 10 these are: (1) a <DataOutput> element for the *display question* task, (2) <SingleChoice> for *select answer*, (3) <ChoiceItem> for the first dummy, (4) another <ChoiceItem> for the second dummy, (5) <Activator> for *confirm choice*, (6) <Activator> for *request results*, (7) <DataOutput> for *display results*, and (8) a <Navigator> element for the *terminate* task in order to abandon the poll procedure.

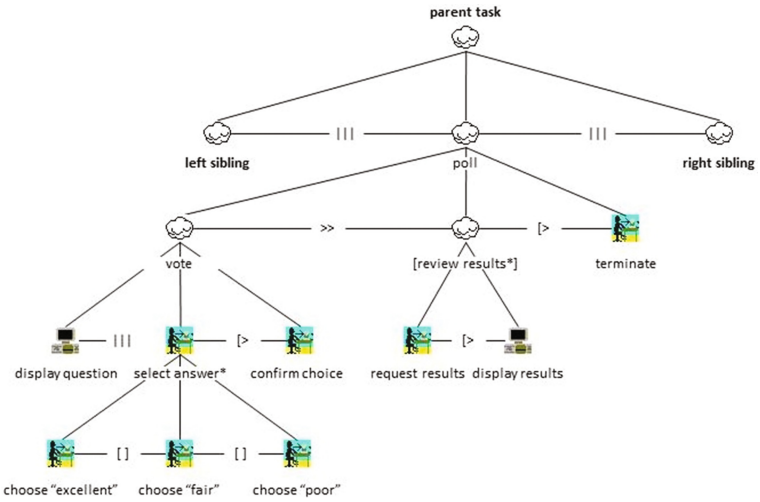

Fig. 9. Task model fragment of the *Poll* pattern integrated in the overall task model

Table 2. Most common concept types

Type	Purpose
DataOutput	Making information perceptible for the user, e.g. displaying it on screen or replaying an audio stream
DataInput	Request input from the user
Activator	Activate a user interface element or call a function
Navigator	Navigate to a different window, screen or dialog
SingleChoice	Select exactly one of several options
MultiChoice	Select none, one, or more of several options
ChoiceItem	A single item that can be chosen by SingleChoice or MultiChoice
UserFeedback	Providing feedback to the user, e.g. about the current system state
Alarm	Providing urgent feedback to the user, e.g. in case of system error or emergency
Cluster	Construct consisting of several components of any type

All required concepts are listed within the concept model fragment of the pattern as presented in Fig. 11. Each concept specification includes links to the tasks that use the concept (see lines 28 and 28 in Fig. 11).

Vice versa, the task specifications contain references to the concepts which are required for their completion (please refer to lines 31 and 32 in Fig. 12).

When the pattern is applied, the user is prompted by the PaMGIS *Pattern Selection & Assignment* function whether the concepts being listed within the CMF should be adopted. If this is the case, the concepts of the pattern will simply be copied into the list of the overall concept model.

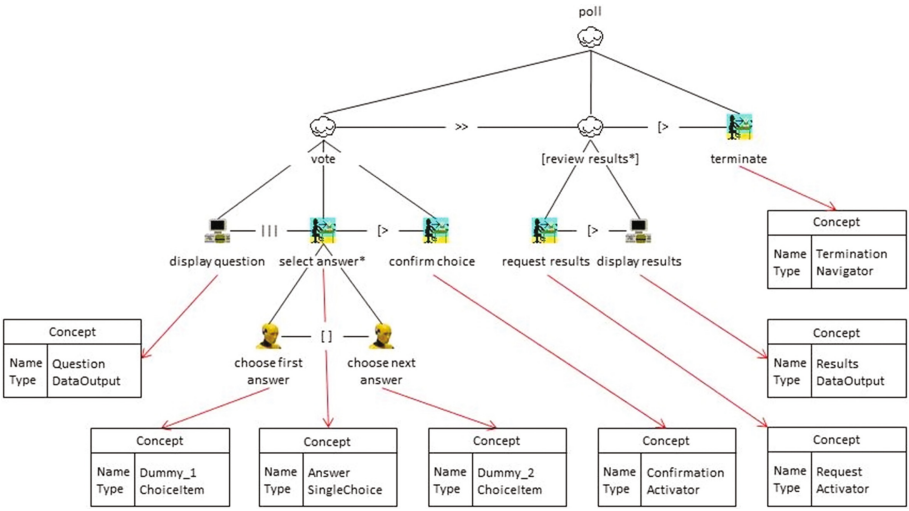

Fig. 10. Deducing required concept from the task model fragment of the *Poll* pattern

3.3 Dialog Model Fragments

While patterns can possess at most one TMF and one CMF they may comprise multiple dialog model fragments (DMF). The task model serves as basis for the specification of dialog models. In consideration of the intended context of use, e.g. the capabilities of the target end user device, it is divided into different sets of closely related tasks, i.e. the dialogs. In addition, the transitions between these dialogs have to be defined. Valuable indications for this work are the hierarchical structure of the task model, i.e. the various sub-trees and the temporal relationships between the tasks.

In the context of PaMGIS dialog models and DMF are specified by means of dialog graphs [9]. Dialogs are represented as boxes containing the name of the dialog as well as the different task that are covered by the dialog. The flow between the various dialogs is specified by arrows, whereupon the arrowheads indicate the direction.

Figure 13 illustrates a possible dialog graph regarding the application of the *Poll* pattern for desktop computers with a large screen. All tasks involved in the poll (see Fig. 7) are combined into one single dialog.

The PPSL representation of this DMF is shown in Fig. 14. The actual dialog specification starts at line 10. The <Position> section (see lines 16 to 39) incorporates information regarding potential predecessor and successor dialogs. The related description elements have to be specified when the pattern is applied. Especially the concepts that trigger the transitions forth and back between the dialogs must be specified, where applicable. This work cannot be accomplished in a fully automated way. Hence, the user must decide how to integrate the DMF into the overall dialog model and which triggers shall be used. However, the PaMGIS *Pattern Selection & Assignment* function offers support regarding this procedure. The tasks being included within the dialog are specified in the <DLG_Tasks> section (see lines 40 to 56). Not necessarily every single task has to be listed here. With the <DLG_TaskProcessing>

```
01: <ModelFragment>
02:  <MDFR_Type>Concept</MDFR_Type>
03:  <MDFR_FragmentID>"__CMF_0001_01"</MDFR_FragmentID>
04:  <MDFR_Label>"Concepts for poll pattern"</MDFR_Label>
05:  <MDFR_Fragment>
06:   <CMF_Content>
07:    <Concept>
08:     <CCPT_ConceptID>"__CPT_0001_01_0001"</CCPT_ConceptID>
09:     <CCPT_ConceptName>"Question"</CCPT_ConceptName>
10:     <CCPT_Description>"Question to be displayed"</CCPT_Description>
11:     <CCPT_Label>""</CCPT_Label>
12:     <CCPT_Perceptible>True</CCPT_Perceptible>
13:     <CCPT_Enabled>True</CCPT_Enabled>
14:     <CCPT_Required>False</CCPT_Required>
15:     <CCPT_ConceptType>DataOutput</CCPT_ConceptType>
16:     <CCPT_DataType>"String"</CCPT_DataType>
17:     <CCPT_ConceptOrigin>
18:      <CCPT_OriginPatternID>"PPT_0100_0001"</CCPT_OriginPatternID>
19:      <CCPT_OriginPatternName>"Poll"</CCPT_OriginPatternName>
20:      <CCPT_OriginPatternVersion>"0001"</CCPT_OriginPatternVersion>
21:      <CCPT_OriginPatternRevision>"0000"</CCPT_OriginPatternRevision>
22:     </CCPT_ConceptOrigin>
23:     <CCPT_Preconditions></CCPT_Preconditions>
24:     <CCPT_Postconditions><CCPT_Postconditions>
25:     <CCPT_DataLink></CCPT_DataLink>
26:     <CCPT_TaskLinks>
27:      <CCPT_TaskLink>
28:       <CCPT_TaskID>"__TSK_0001_01_0005"</CCPT_TaskID>
29:       <CCPT_TaskName>"display question"<CCPT_TaskName>
30:      </CCPT_TaskLink>
31:     </CCPT_TaskLinks>
32:    </Concept>
33:    <Concept>
34:     <CCPT_ConceptID>"__CPT_0001_01_0002"</CCPT_ConceptID>
35:     <CCPT_ConceptName>"Answer"</CCPT_ConceptName>
……
```

Fig. 11. Excerpt of the XML representation of the CMF of the *Poll* pattern

element (refer to lines 44, 49, and 54) it can be indicated whether solely the related sub-task itself (*Exclusive*) or also all defined subtasks shall also be included in the dialog specification (*Recursive*).

A second scenario for another DMF for the *Poll* pattern is the implementation on mobile devices with limited screen space, such as a cell phones or smart phones. Due to the display size limits it is not possible or reasonable to display all interaction elements simultaneously. Therefore, we opt for the definition of two different dialogs with a transition between them as illustrated in Fig. 15.

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 decided to assign the sub-tree starting at the *vote* task to the first (named *Vote*) and likewise the sub-tree beginning at the *review results* task to the second dialog (named *Results*). Since it makes no sense to treat the remaining single interaction task *terminate* separately,

```
01: <Subtask>
02:  <TaskID>"__TSK_0001_01_0005"</TaskID>
03:  <TaskName>"display question"</TaskName>
04:  <TaskDescription>""</TaskDescription>
05:  <TaskType>Application</TaskType>
06:  <TaskOrigin>
07:   <TaskOriginPatternID>"PPT_0100_0001"</TaskOriginPatternID>
08:   <TaskOriginPatternName>"Poll"</TaskOriginPatternName>
09:   <TaskOriginPatternVersion>"0001"</TaskOriginPatternVersion>
10:   <TaskOriginPatternRevision>"0000"</TaskOriginPatternRevision>
11:  </TaskOrigin>
12:  <Optional>False</Optional>
13:  <Iterative>False</Iterative>
14:  <Position>
    ......
27:  </Position>
28:  <TemporalOperator>Interleaving</TemporalOperator>
29:  <UIConcepts>
30:   <UIConcept>
31:    <UIConceptID>"__CPT_0001_01_0001"</UIConceptID>
32:    <UIConceptName>"Question"</UIConceptName>
33:   </UIConcept
34:  </UIConcepts>
35: </Subtask>
......
```

Fig. 12. Excerpt of the XML representation of the *display question* subtask

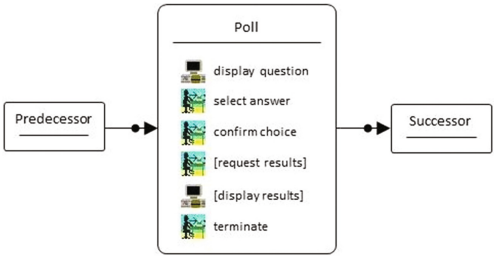

Fig. 13. Dialog model fragment of the *Poll* pattern for desktop computers (large screen)

we added it to the *Results* dialog. The transition from the *Vote* to the *Results* dialog is triggered as soon as the *confirm choice* task is completed. The integration of the DMF into the overall dialog model as soon as the pattern is applied is carried out under the same conditions as described before.

Due to lack of space we abstained from including the PPSL specification of this second dialog model fragment into the document at hand. But its derivation is straight forward and similar to the previous example (see Fig. 14).

```
01: <ModelFragment>
02:<MDFR_Type>Dialog</MDFR_Type>
03:<MDFR_FragmentID>"__DMF_0100_01"</MDFR_FragmentID>
04:<MDFR_Label>"Poll"</MDFR_Label>
05:<MDFR_Purpose>"DMF for large screens"</MDFR_Purpose>
06:<MDFR_Diagram>"see Fig. 13"</MDFR_Diagram>
07:<MDFR_Fragment>
08:<DMF_ContextModelReferences></DMF_ContextModelReferences>
09:<DMF_Content>
10:<Dialog>
11:<DialogID>"__DLG_0100_01_01"</DialogID>
12:<DialogName>"Poll"</DialogName>
13:<DialogDescription>"Poll for large screens"</DialogDescription>
14:<DialogLabel>"Poll"</DialogLabel>
15:<DialogType>Modal</DialogType>
16:<Position>
17:<Predecessors>
18:<Predecessor>
19:<PRED_DialogID>""</PRED_DialogID>
20:<PRED_DialogName>""</PRED_DialogName>
21:<PRED_TransitionType>Sequential</SUCC_TransitionType>
22:<PRED_Trigger>
23:<PRED_TaskID>""</SUCC_TaskID>
24:<PRED_TaskName>""<SUCC_TaskName>
25:<PRED_Trigger>
26:</Predecessor>
27:<Predecessors>
28:<Successors>
29:       <Successor>
30: <SUCC_DialogID>""</SUCC_DialogID>
31:<SUCC_DialogName>""</SUCC_DialogName>
32:<SUCC_TransitionType>Sequential</SUCC_TransitionType>
33:<SUCC_Trigger>
34:<SUCC_TaskID>"__TSK_0001_01_0004"</SUCC_TaskID>
35:<SUCC_TaskName>"terminate"<SUCC_TaskName>
36:<SUCC_Trigger>
37:       </Successor>
38:</Successors>
39:</Position>
40:<DLG_Tasks>
41:<DLG_Task>
42:<DLG_TaskID>"__TSK_0100_01_0002"</DLG_TaskID>
43:<DLG_TaskName>"vote"</DLG_TaskName>
44:<DLG_TaskProcessing>Recursive</DLG_TaskProcessing>
45:<DLG_Task>
46:<DLG_Task>
47:<DLG_TaskID>"__TSK_0100_01_0003"</DLG_TaskID>
48:<DLG_TaskName>"review results"</DLG_TaskName>
49:<DLG_TaskProcessing>Recursive</DLG_TaskProcessing>
50:<DLG_Task>
51:<DLG_Task>
52:<DLG_TaskID>"__TSK_0100_01_0004"</DLG_TaskID>
53: <DLG_TaskName>"terminate"</DLG_TaskName>
54:<DLG_TaskProcessing>Exclusive</DLG_TaskProcessing>
55:</DLF_Task>
56:</DLG_Tasks>
57:</Dialog>
58:</DMF_Content>
59:</MDFR_Fragment>
60:</ModelFragment>
```

Fig. 14. XML representation of DMF of the *Poll* pattern for desktop computers (large screen)

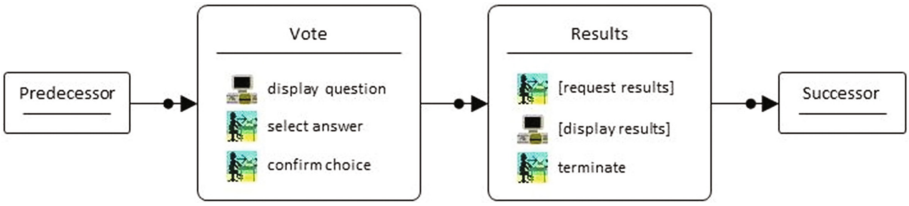

Fig. 15. Dialog model fragment of the *Poll* pattern for mobile devices (small screen)

3.4 Derivation of the User Interface

In an initial step, the structure of the abstract user interface (AUI) must be built up. Here, the dialog model delivers fundamental input. The tasks which are combined within the dialogs include links to the particular concepts being required to complete these tasks. As illustrated in Fig. 16 for the DMF of the *Poll* pattern for mobile devices, these concepts are mapped to corresponding AUI elements. The dialogs themselves are treated as <Cluster> elements. In the example at hand, the *display question* task is linked to a concept of the type *DataOutput*. It is intended to hold a string variable specifying the question for the poll. The *select answer* task and its sub-tasks provide the information regarding the selection of the best fitting answer. Hence, it is a matter of a <SingleChoice> AUI element with related <ChoiceItem> elements. The *confirm choice* and *request results* tasks contribute <Activator> elements while *display results* relates to a <DataOutput> element for displaying an image and *terminate* to a <Navigator> element allowing to abandon the poll process and effect the transition to the subsequent dialog.

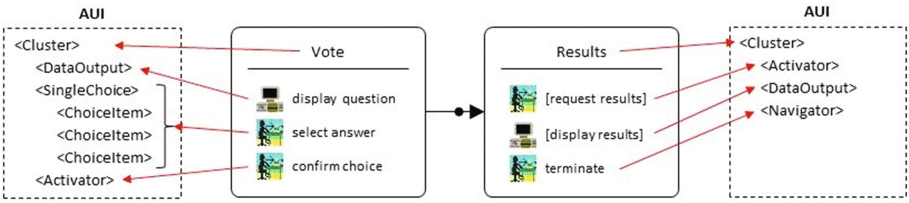

Fig. 16. Deriving the AUI from the DMF of the *Poll* pattern for mobile devices (small screen)

The XML representation of the related AUI excerpt is sketched in Fig. 17. Here, the *Vote* dialog is covered by the first <Cluster> element (see lines 01 to 30), the *Results* dialog by the second one (see lines 31 to 46).

In the next process step, the AUI specification is to be translated into a concrete user interface (CUI) description. In the example we want to arrive at a final user interface (FUI) coded in Hypertext Markup Language (HTML). In this regard, the <Cluster> AUI elements are mapped to <Form> CUI elements as illustrated in Fig. 18. The <DataOutput> element for the poll question is represented as a <TextField> and the <SingleChoice> element together with its associated <ChoiceItem> elements is

```
01: <Cluster>
02:  <ClusterID=>"__GRP_01"</ClusterID>
03:  <ClusterName>"Vote"</ClusterName>
04:<DataOutput>
05:<ConceptID>"__CPT_0001_01_0001"</ConceptID>
06:<ConceptName>"Question"</ConceptName>
07:</DataOutput>
08:<SingleChoice>
09:   <ConceptID>"__CPT_0001_01_0002"</ConceptID>
10:   <ConceptName>"Answer"</ConceptName>
11:   <ChoiceItems>
12:    <ChoiceItem>
13:     <ConceptID>"__CPT_0001_01_0007"</ConceptID>
14:     <ConceptName>"Answer1"</ConceptName>
15:    </ChoiceItem>
16:<ChoiceItem>
17:<ConceptID>"__CPT_0001_01_0008"</ConceptID>
18:<ConceptName>"Answer2"</ConceptName>
19:</ChoiceItem>
20:<ChoiceItem>
21:<ConceptID>"__CPT_0001_01_0009"</ConceptID>
22:<ConceptName>"Answer3"</ConceptName>
23:</ChoiceItem>
24:</ChoiceItems>
25:</SingleChoice>
26:</Activator>
27:<ConceptID>"__CPT_0001_01_0003"</ConceptID>
28:<ConceptName>"Confirmation"</ConceptName>
29:  </Activator>
30: </Cluster>
31:<Cluster>
32:<ClusterID>"__GRP_02"</ClusterID>
33:<ClusterName>"Results"</ClusterName>
34:<Activator>
35:<ConceptID>"__CPT_0001_01_0004"</ConceptID>
36:<ConceptName>"Request"</ConceptName>
37:  </Activator>
38:<DataOutput>
39:<ConceptID>"__CPT_0001_01_0005"</ConceptID>
40:<ConceptName>"Results"</ConceptName>
41:</DataOutput>
42:<Navigator>
43:<ConceptID>"__CPT_0001_01_0006"</ConceptID>
44:<ConceptName>"Termination"</ConceptName>
45:</Navigator>
46:</Cluster>
```

Fig. 17. XML representation the AUI of *Poll* pattern for mobile devices (small screen)

translated into <Radiobuttons> on the CUI level. The <Activator> and <Navigator> elements are transformed into <Button> elements and the remaining <DataOutput> element into a <Picture> CUI element.

Finally, the CUI specification must be transformed into the final user interface, i.e. HTML source code. The <Form> elements are both replaced by <form target="_blank"> constructs that each incorporate the required UI elements of the original *Vote* resp. the *Results* dialogs. The <TextField> element is realized by a <p> HTML tag while the three radio buttons appear as <input type="radio"> and the <Button> elements as <button type="submit"> tags. Finally, the <Picture> CUI element is translated into a <img> tag. Figure 19 shows screenshots of the resulting HTML FUI. The red arrow indicates the transition from the *Vote* to the *Results* dialog when the confirm button is pressed.

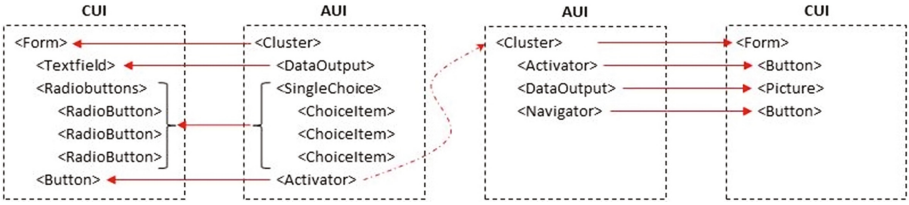

Fig. 18. Deriving the CUI from the AUI of the *Poll* pattern for mobile devices (small screen)

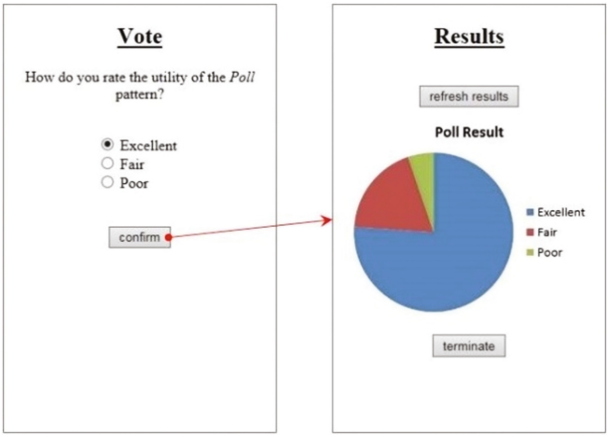

Fig. 19. Realization of the dialog model for mobile devices (small screen) (Color figure online)

4 Conclusion

The PaMGIS framework combines a model-driven user interface development approach and pattern-based development techniques. It allows for the creation of abstract user interface models (AUI) on the basis of fundamental information stored in task, concept, and dialog models. In consideration of context-of-use characteristics held in user, device, UI toolkit, and environment models, the AUI is iteratively transformed into a concrete UI representation which, in turn, is used to generate respective user interface source code for the intended context.

In the current paper, we emphasized on practical aspects of the work with the PaMGIS framework. Using the example of the *Poll* pattern which we have adopted from Martijn van Welie's pattern collection *Patterns in Interaction Design* [12] and translated into PPSL, we have demonstrated how task, concept, and dialog model fragments can be derived from the pattern description. In addition, we delivered insight into the formal description of these model fragments and showed how they can be used as building blocks during the construction of the respective constitutive PaMGIS models when a pattern is applied.

We believe that the hybrid UI development process can accelerate the model-driven design process, feature the reuse of already existing design know-how, and effectively contribute to high usability and appealing user experience of the resulting user interfaces. Amongst others, our plans for future research include various studies to substantiate these presumptions and render related facts more precisely.

References

1. Calvary, G., et al.: The CAMELEON Reference Framework. Document D1.1 of the CAMELEON R&D Project IST-2000-30104 (2002)
2. van Duyne, D., et al.: The Design of Sites: Patterns for Creating Winning Websites, 2nd edn. Prentice Hall International (2006). ISBN 0-13-134555-9
3. Engel, J., Herdin, C., Märtin, C.: A review of HCI pattern tools. In: Proceedings of IHCI 2015, Las Palmas de Gran Canaria, Spain, 22–24 July 2015, pp. 51–58. IADIS Press (2015)
4. Engel, J., Herdin, C., Märtin, C.: A review of user interface description languages. In: Proceedings of the 6. Forum Medientechnik, St. Pölten, Austria – Next Generation, New Ideas, vwh, pp. 183–198 (2014)
5. Engel, J., Herdin, C., Märtin, C.: Evaluation of model-based user interface development approaches. In: Kurosu, M. (ed.) HCI 2014. LNCS, vol. 8510, pp. 295–307. Springer, Cham (2014). doi:10.1007/978-3-319-07233-3_28
6. Engel, J., Herdin, C., Märtin, C.: Exploiting HCI pattern collections for user interface generation. In: Proceedings of PATTERNS 2012, Nice, France, IARIA 2012, pp. 34–44 (2012)
7. Engel, J., Märtin, C., Forbrig, P.: A unified pattern specification formalism to support user interface generation. In: Kurosu, M. (ed.) HCI 2016. LNCS, vol. 9731, pp. 445–456. Springer, Cham (2016). doi:10.1007/978-3-319-39510-4_41
8. Fincher, S., et al.: Perspectives on HCI patterns: concepts and tools (introducing PLML). In: CHI 2003 Workshop Report (2003)
9. Forbrig, P., Reichart, D.: 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 (2007)
10. Paternò, F.: ConcurTaskTrees: an engineered approach to model-based design of interactive systems, ISTI-C.N.R., Pisa (2001)
11. Tidwell, J.: Designing Interfaces: Patterns for Effective Interaction Design, 2nd edn. O'Reilly Media Inc. (2011). ISBN 978-1-449-37970-4
12. van Welie, M.: Patterns in interaction design. http://www.welie.com. Accessed 6 Jan 2016