Adapting User Interface Models by Transformations Based on UI Patterns

Mathias Kühn(✉) and Peter Forbrig

University of Rostock, Albert-Einstein-Str. 22, 18051 Rostock, Germany
{mathias.kuehn,peter.forbrig}@uni-rostock.de

Abstract. Models used for software designs are artifacts of today's development culture. Generators and interpreters for models reduce the implementation effort and open a broader range of applications. This also is true for user interface models in any context of use (The Context of Use: http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/#the-context-of-use.). UI models that are designed independently of end users together with varying platforms in alternating environments can be used in many contexts. Of course, derived transformations can be complex and are not as simple as needed. Applying reusable solutions to model-based user interface specifications implies transformations that could be performed automatically and adapt user interfaces to specific contexts of use. UIs can benefit from proven structures that are commonly used in cross-domain software. Applying patterns to model-based UI specifications is the focus of the paper. An example shows how UIs can be adapted by transformations based on patterns that are part of relevant specifications.

Keywords: Model-based user interfaces · UI patterns · Context-specific transformations

1 Introduction

Designing software for a broad range of applications is a challenging task. Different contexts of use force to adapt implemented designs accordingly. These adaptations can be based on transformations that could be performed at runtime by interpreters that are used in specific contexts. Design specifications based on models often are used to reduce the implementation effort. This also is true for models that describe the end users interface to the implemented functions. User interfaces (UIs) enable to access software by any user on any platform in any environment [12]. Designs that allow transforming model-based UI specifications to every context of use would further reduce the effort. In order to achieve this goal, UI specifications also need to be adapted.

Transformations for design adaptations can be based on general reusable solutions for commonly recurring problems. Design patterns [5] have an impact on object-oriented designs that also can be structures for interactive systems. Of course, UI designs can benefit from structures that are proven in cross-domain software [15]. Nevertheless, UI model transformations need to be specified for adaptations. Extending UI specifications with notations for pattern applications can reduce the effort for specifying corresponding transformations. Additionally, transformations at runtime depend on specific contexts that adapt solutions to users, platforms, and environments.

The paper is structured as follows: the next section considers model-based UIs together with the CAMELEON Reference Framework [2] that presents an approach for transformations of UIs to any context of use. Additionally, patterns for interactive systems and user interfaces are considered that can be used more explicitly with model-based languages. The following section considers an example that illustrates the idea as implementation of the model-based language UsiXML [10]. Also a pattern-based extension for this language is proposed for the reason of applicability. Section 4 highlights the approach in detail. Benefits and drawbacks are considered that focus on potentials and limitations of the proposed approach. The last section summarizes the paper and gives an outlook on tools that could be improved by the approach.

2 Related Work

Model-based UIs are essential artifacts for operating various interactive devices together with different surfaces. UIs can be specified with model-based languages that are independent of specific contexts. Additionally, UI patterns can implicitly be part of these specifications. Work in the area of model-based UIs together with patterns mostly considers design-time specifications [4, 6, 8, 14]. Pattern specification together with runtime interpretation of corresponding UI models [9, 16] need to be considered more extensively. Transformations based on patterns can be performed within given contexts that adapt UIs accordingly.

Patterns for object-oriented designs [5] were introduced in 1994 and describe structures that have been proven in various software systems from then until now. Applying these patterns to specific contexts helps to get solutions for problems on structures, behaviors, and creations of object-oriented implementations. Patterns for interaction [17, 18] were identified for designs that either focuses the development (based on evolutions of UIs) or the user (based on changes of requirements) of interactive systems. Of course, these patterns target the design of user interfaces directly.

One of the problems that pattern application can solve relates to the navigation within UIs. UI patterns like stepwise, hub-and-spoke, and pyramid (see Fig. 1) can be applied that describe different ways of navigating in the system. Other UI patterns can for instance specify layouts, complex data visualizations, and beautifications.

Navigational patterns allow specifying dialog structures that can later be generated to source code automatically. Applying patterns allows adapting generated UIs to different contexts of use.

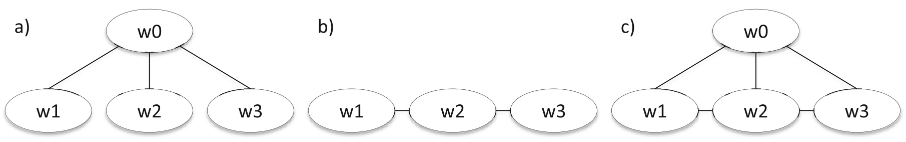

Fig. 1. Visualization of UI pattern applications for (a) hub & spoke, (b) stepwise, and (c) pyramid represented by navigation structures for windows w0, w1, w2, and w3

User interfaces can be specified with model-based languages like MARIA [13] and UsiXML [10, 11] that allow specifying UIs on different levels of abstraction. Both languages allow specifications of models for tasks and abstract & concrete UIs that are grounded on the CAMELEON Reference Framework (CRF). The CRF [2] targets the transformation of UI models from as well as into different contexts. UsiXML additionally allows model-based specifications of contexts, mappings, and transformations. Unfortunately, both languages do not allow specifying UIs together with pattern applications. However, patterns need to be applied within specifications that are based on those languages for adapting UIs by corresponding transformations.

Approaches like Role-Based User Interface Simplification (RBUIS) [1] and Multi-Adaptive Migratory User Interfaces (MAMUI) [20] also are grounded on the CRF and consider adaptations of UIs. In RBUIS UIs are adapted by minimizing features-sets and optimizing layouts for increasing usability. In MAMUI UIs are adapted by an Adaptation Manager that adapts models for task features, navigations, and layouts for improving user experience. However, approaches like those do not consider UI patterns for deriving transformations that could adapt UIs.

Another problem in the area of pattern specification is that formal languages are missing. The Pattern Language Markup Language (PLML[1]) was introduced in 2003 and is used in approaches like [19], but it lacks in formality what makes it problematic for deriving transformations. In contrast to this, DelTa [3] is a visual language that allows describing patterns more formally. However, in our approach we decided to extend UsiXML for applying patterns explicitly as part of specifications. This allows deriving transformations that affect reifications and abstractions of the CRF. These transformations can for instance be made with ATL [7] which is based on QVT[2] and allows describing model-to-model transformations that are needed in the context of CRF.

3 Questionnaire Survey Application

The following section considers an example that illustrates the idea of the proposed approach. Both instances focus the specification of patterns together with UI specifications that later are used for context-dependent transformations. Additionally, screenshots give an impression of resulting instances. An extension of the model-based language UsiXML is used for the reason of applicability.

[1] PLML: http://www.cs.kent.ac.uk/~saf/patterns/CHI2003WorkshopReport.doc.
[2] QVT: http://www.omg.org/spec/QVT/.

3.1 Simple Example

Let's consider a small example that illustrates the idea of the proposed approach. The model-based language UsiXML is used for specifying structures of user interfaces. The language additionally is extended for specifying patterns that are interpreted within specific contexts later. Patterns can be specified by enveloping targeted component specifications with a special XML element. This element is used for applying patterns by performing pattern-related transformations. If necessary, roles within the patterns can be introduced as XML attributes as well.

```
<uimodel>
    <cuimodel>
        <pattern name="stepwise">
            <window name="w0"/>
            <window name="w1"/>
            <window name="w2"/>
        </pattern>
    </cuimodel>
</uimodel>
```

Fig. 2. UI pattern stepwise applied to extended UI model specification

Figure 2 shows an application of the navigational UI pattern stepwise within a concrete user interface model specification that contains three windows. The pattern is applied to components that directly are enveloped by this XML element what are the three windows. These windows can later be extended by elements that are related to the specified pattern. For instance, they could be extended by trigger components (e.g. buttons) that allow users to navigate between them.

Additionally, the dialog structure can also be created for reified windows that refer the behavior of corresponding UIs. However, this could be implemented in different ways for users that interact via speech gestures. Such pattern-related transformations can be performed within given contexts at runtime and for any context at design time. Patterns also can be replaced by others what would lead to other transformations. This has to be done by designers that specify corresponding model-based UIs. Later tools automatically take changes into account.

3.2 Extended Example

Let's consider another more complex example. Someone is conducting a customer survey and is planning to use questionnaires for gathering data. Such questionnaires can be specified as interactive forms that are used within data collection applications later. Additionally, each question of the questionnaire is specified as a single form that contains the question (e.g. label) as well answers (e.g. radio button, check box), for instance with choice questions. Participants are asked about personal information (gender, age group, etc.) and about information on a certain product (satisfaction, etc.).

Figure 3 shows an abbreviated specification of UIs that later are used as interactive forms. The forms contain four questions of the questionnaire. Some of the widgets that can be used together with CUI model specifications are labels and radio buttons. These are parts the following instance.

```
<uimodel>
    <cuimodel>
        <pattern name="stepwise" direction="forward">
            <window name="w0">
                <label>What is your gender?</label>
                <!-- choice -->
            </window>
            <window name="w1">
                <label>How old are you?</label>
                <!-- choice -->
            </window>
            <pattern name="stepwise">
                <window name="w2">
                    <label>How often do you use the product?</label>
                    <!-- choice -->
                </window>
                <window name="w3">
                    <label>How well do the product meet your needs?</label>
                    <!-- choice -->
                </window>
                <!-- more questions -->
            </pattern>
        </pattern>
    </cuimodel>
</uimodel>
```

Fig. 3. Extended UI model specification with UI patterns combined

Figure 3 shows a part of the extended UI model specification for interactive forms of a questionnaire. Two patterns are combined into each other that provide a certain navigation structure within the final user interface. At first, participants will answer questions on personal information. According to the UI specification, they are not allowed to go back when they have finished entering individual information. This is specified by the attribute direction of the pattern element. The value forward is used for parameters of the UI pattern stepwise.

After entering personal information, participants are asked about information on the product itself. They are allowed to go back when they have finished entering any information. This is specified by the enveloping pattern element. No value is specified for the direction attribute what will be interpreted as unspecified and allows to navigate forward as well as backward by default. It is easy to change the navigation structure just by exchanging applied patterns. For instance, exchanging the topmost UI pattern stepwise with pyramid would lead to adding links to each form that refer to an extra form. This extra form is generated automatically and holds links to the specified forms as well.

Regarding the UsiXML extension, UI patterns in general should be specified by XML elements (tags) that directly refer to patterns by their attribute. Of course, patterns also can be applied for beautifications and other purposes. However, the referred user interface elements are enveloped by the XML element for patterns. The corresponding attribute refers to the semantics of the corresponding pattern instance.

The specification of Fig. 3 can be used for generating UIs to different context of use. Patterns are implemented to the reified windows of the final UI specification. Additionally, the pattern-specific dialog structure is implemented for navigating users of FUI instances. Following this, specifications of these transformations do not need to be designed. Figure 4 show examples of some generated UIs.

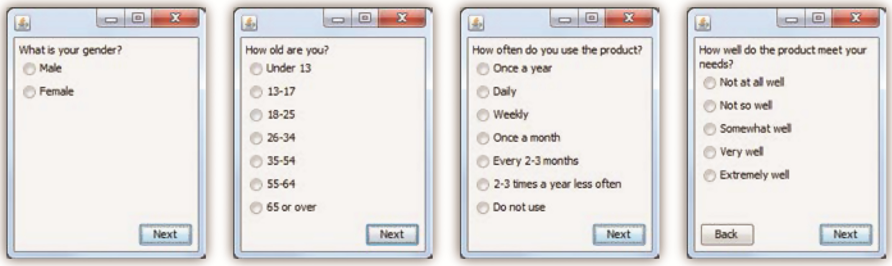

Fig. 4. Screenshots of some generated UIs

Figure 4 shows some screenshots of the generated UIs for the specification in Fig. 3. The windows are adapted to users with Desktop PCs that can use buttons for navigating as it is intended by the patterns. The buttons as well as the dialog structure are generated automatically. Adaptations for users that use vocal interfaces would imply speech gestures instead of buttons together with an equal dialog model. However, UI patterns are applied within given contexts by transforming UI model specifications to final UI instances that depend on pattern-specific dialog models.

4 Approach

The following section discusses the proposed approach in a more abstract way using the discussed example of a transformation. A visualization illustrates modifications that are made while performing adaptations. It also is described how the CRF is applied to achieve mentioned transformations. The approach as well as the corresponding tool support is discussed afterwards. All visualizations are relating to model-based languages in general. Following this, UsiXML is one candidate for applications (compare Fig. 3 with 5a and Fig. 4 with 5b).

FUI model instances are not covered by model-based languages for UIs. Code generators or interpreters can be used for transforming CUI to FUI model instances. According to this, windows of CUI model specifications can for instance be transformed to JFrame implementations in Java programming language. However, an example of a reification transformation is presented in the following figure.

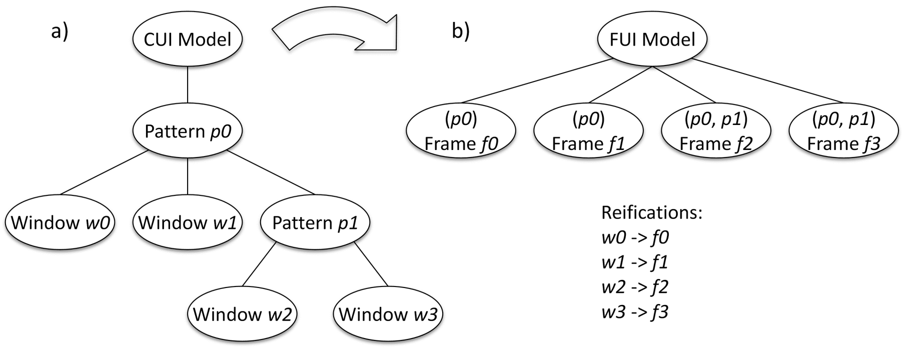

Fig. 5. Visualization of proposed transformations

Figure 5 shows a visualization of transforming an (a) concrete into a (b) final UI model specification. The CUI model specification considers four windows (w0, w1, w2, w3) that are referenced by different patterns (p0, p1). These patterns are part of the CUI model specification and are used for generating pattern-specific structures within the final UI. When performing context-dependent transformations of the CUI model specification, the final UI model can for instance be extended by triggers that allow navigating through windows as specified by the corresponding pattern.

Transformations that result in FUI model specifications are based on CRF. The CRF considers needed reifications for gaining UI instances on specific platforms. These reifications also target applied patterns that need to be implemented accordingly. The following figure shows the relation between patterns and transformations that is implicitly be shown in the figure above.

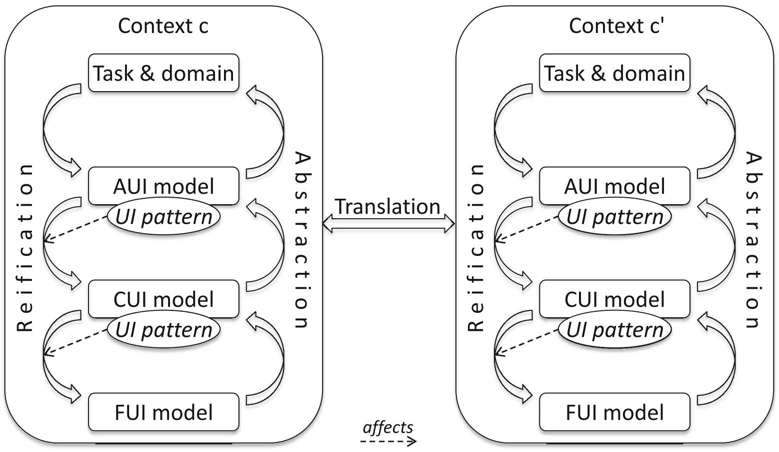

Fig. 6. CAMELEON Reference Framework adapted to the approach

Figure 6 shows the CRF adapted to the proposed approach containing models that can be transformed into each other. The main transformation types are abstraction (concrete to abstract models), reification (abstract to concrete models) and translation (models from one context to another on the same level of abstraction). Abstractions and reifications are considered for a single context only. This is needed to adapt UIs for instance by interpreters at runtime. UIs in general are considered to be on different levels of abstraction (abstract, concrete, final UI).

The level of task & domain would target patterns for user activities and object-oriented structures that rather implicitly is focus of the paper. However, UI patterns can be part of AUI/CUI model specifications and explicitly have an impact on the reification transformations to the underlying model levels. These transformations depend on contexts of use and adapt UI models to runtime use. On abstraction, UI patterns used in CUI models can be moved to AUI models. Applied UI patterns are substituted by their context-dependent implementations on reification.

Any specific UI pattern can be applied either in AUI or in CUI models. Additionally, reified models can be extended with other UI patterns that are more context-specific and meet the end-users needs more. The resulting transformations are based on applied UI patterns that adapt models accordingly. An example of such a transformation is given in Fig. 8 that implements a rule for transforming stepwise pattern instances from AUI to CUI model specifications for graphical UIs.

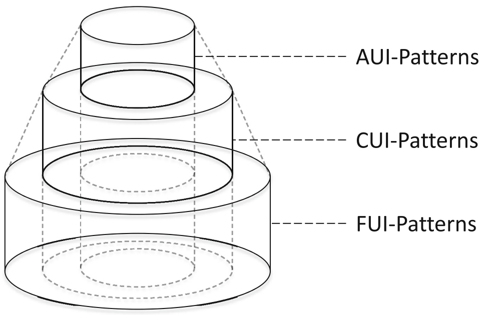

Fig. 7. Visualization of relations between UI pattern applications

Figure 7 shows a visualization of relations between UI patterns that are applied to different UI model specifications. UI patterns that are applied to AUI models remain in CUI models and can be extended with more context-specific patterns. Additionally, UI patterns that are applied to CUI models remain in FUI models and also can be extended with other patterns. However, most patterns can be applied to FUI model specifications that is relating to the CRF. Instances of individual pattern-based solutions depend on any context that they are applied to what is the effect of pattern applications in general.

The approach proposed in the paper considers explicit specifications of UI pattern applications. These specifications allow more context-specific transformations and adapt UIs accordingly. Tools can support designers in specifying pattern applications. According to UsiXML, this can simply be made with XML-Editors that also can be

Text-Editors. Tools that interpret specifications at runtime need to transform instances when contexts of use change. Data that need to be collected for this can be gained by sensors that have to be available for specific platforms.

Transformations can be complex and need to be specified for performing them. Applying patterns explicitly can help to reduce the effort for specifying these transformations. Figure 8 gives an example of such a transformation in ATL. OCL[3] is used for specifying transformations of model instances.

Patterns and components are of type element. Elements can have child nodes that are of type element as well. Components can for instance be transformed to windows of CUI model specifications. However, applications of UI pattern stepwise for AUI model specifications are transformed to CUI model instances. Pattern applications are implemented to sequences of windows. Sequence is an OCL type for collections of ordered elements. These elements can be patterns or components of AUI model instances, respectively.

```
rule StepwiseToCuiModel extends PatternToCuiModel {
    from
        stepwise : IN!stepwise
    to
        windows  : Sequence(OUT!window)
    do {
        for (e in stepwise.elements) {
            if (e.oclIsKindOf(IN!component)) {
                windows <- windows.append(thisModule.ComponentToWindow(e, stepwise.direction));
            }
            if (e.oclIsKindOf(IN!pattern)) {
                windows <- windows.union(thisModule.PatternToCuiModel(e));
            }
        }
    }
}
```

Fig. 8. Example of a transformation specification in ATL

Figure 8 shows an ATL rule for transforming applied stepwise UI patterns from AUI to CUI model specifications within the context of graphical UIs. This transformation rule represents a way of performing model-to-model transformations that are based on patterns. Tools for runtime interpretation of CUI/AUI model specifications need to perform more context-depended transformations like this. Context model specifications can only be gained within specific contexts of use that those tools have to interpret.

The example above demonstrates two kinds of transformations for implementing UI patterns within one rule. One step is to transform instances from abstract to concrete models that are parts of reifications. The other step is to apply patterns to any transformed instance. Someone can imagine that both steps can be performed in any order, but the final results need to be equal. Tools can apply patterns to present instances first and reify them afterwards or they perform reifications first and apply patterns to reified instances then. However, both steps are made with one rule in the example above.

Transforming abstract specifications together with applied patterns to their reified instances is a general assumption for generating adapted UIs within the approach.

[3] OCL: http://www.omg.org/spec/OCL/.

Patterns are implemented within concrete specifications. Of course, other patterns can be applied again to reified models, but patterns of abstract specifications are already implemented within concrete instances. Another idea for pattern applications can be to implement UI patterns only on transforming CUI models to FUI models. This can prevent reapplications of patterns and make implementations more comprehensible.

In the end, pattern applications could be commented within source codes. However, replacing applied patterns by more context-specific patterns would also be easier if their specifications explicitly remain in reified instances. Pattern implementations would only be part of code generators then.

5 Conclusion and Future Work

The paper introduces an approach for specifying UI patterns together with UI models on different levels of abstraction. These pattern-related specifications target the transformation of corresponding models that can be used for adapting UIs to specific contexts of use. UI patterns together with UI specifications reduce the effort for specifying transformations that are needed for adapting UIs. Additionally, adaptations can increase usability and improve user experience of resulting generated UIs.

Specifications and transformations can be made with different languages and different tools that allow adapting UIs. The proposed approach suggests a UsiXML extension for specifying UI models together with UI patterns. It is assumed that patterns can be specified on different levels of abstraction and can be extended with more context-specific patterns in reified models. Comparing to [4, 6], the approach does not consider pattern replacements for adapting UIs to specific contexts. Instead of this, it is assumed that transformations implement patterns to their reified instances. An example of a transformation rule is given in ATL (see Fig. 8) and refers to needed tool support.

Further investigations need to be done on giving adequate tool support for applying UI patterns into UI model specifications. An option can be that designers are aware of any pattern that could be applied into specifications. This implies that they also need to be aware of abstraction levels for pattern applications. Another option can be that designers are supported with a common UI design tool that allows applying patterns to UIs for instance with wizards. However, made specifications need to be interpreted on any platform later. This leads to generating UIs which are adapted to a given context.

There might be the problem that patterns could not be adapted to a given context. For instance, UI patterns that are related to graphical UIs cannot be implemented to vocal UIs. This can be a problem if designers specify patterns that only can be applied to graphical UIs. According to this, replacing patterns can be a solution for making UIs more context-dependent. However, tool support needs to be fine-grained relating to this problem what makes it more difficult for designers again.

References

1. Akiki, P.A., Bandara, A.K., Yu, Y.: Engineering adaptive model-driven user interfaces. IEEE Trans. Softw. Eng. **42**(12), 1118–1147 (2016)
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**, 289–308 (2003)
3. Ergin, H., Syriani, E., Gray, J.: Design pattern oriented development of model transformations. Comput. Lang. Syst. Struct. **46**, 106–139 (2016)
4. Forbrig, P., Saurin, M.: Supporting the HCI aspect of agile software development by tool support for UI-pattern transformations. In: Bogdan, C., Gulliksen, J., Sauer, S., Forbrig, P., Winckler, M., Johnson, C., Palanque, P., Bernhaupt, R., Kis, F. (eds.) HCSE/HESSD -2016. LNCS, vol. 9856, pp. 17–29. Springer, Cham (2016). doi:10.1007/978-3-319-44902-9_2
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Prentice Hall, Upper Saddle River (1994)
6. Javahery, H., Seffah, A., Engelberg, D., Sinnig, D.: Migrating user interfaces across platforms using HCI patterns. In: Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces, pp. 241–259. Wiley (2004)
7. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In: Proceedings of OOPSLA, pp. 719–720 (2006)
8. Kühn, M.: Applying Patterns when generating code: a model-based design approach. In: Proceedings of MIDI (2015)
9. Kühn, M., Forbrig, P.: Applying UI patterns for modeling dialogs. In: Proceedings of 2nd PAME/VOLT@MODELS Workshop, pp. 13–17 (2016)
10. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: USIXML: a language supporting multi-path development of user interfaces. In: Bastide, R., Palanque, P., Roth, J. (eds.) DSV-IS 2004. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005). doi:10.1007/11431879_12
11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D.: USIXML: a user interface description language for context-sensitive user interfaces. In: Proceedings of AVI Workshop, pp. 55–62 (2004)
12. Paterno, F., Santoro, C.: One model, many interfaces. In: Kolski, C., Vanderdonckt, J. (eds.) Proceedings of CADUI, pp. 143–154. Springer, Dordrecht (2002)
13. Paterno, F., Santoro, C., Spano, L.D.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput. Hum. Interact. **16**(4), 1–30 (2009)
14. Seffah, A., Gaffar, A.: Model-based user interface engineering with design patterns. J. Syst. Softw. **80**(8), 1408–1422 (2007)
15. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., Seffah, A.: Patterns in model-based engineering. In: Proceedings of CADUI, pp. 197–210 (2004)
16. Taleb, M., Seffah, A., Abran, A.: A UsiXML proposal for a pattern-oriented and model-driven architecture for interactive systems. In: Proceedings of ADVCOMP, pp. 24–29 (2013)
17. Tidwell, J.: Designing Interfaces, 2nd edn. O'Reilly Media, Sebastopol (2010)
18. Van Welie, M.: Interaction Design Pattern Library. http://www.welie.com/patterns/
19. Vanderdonckt, J., Simarro, F.M.: Generative pattern-based design of user interfaces. In: Proceedings of PEICS, pp. 12–19 (2010)
20. Yigitbas, E., Sauer, S., Engels, G.: A model-based framework for multi-adaptive migratory user interfaces. In: Kurosu, M. (ed.) HCI 2015. LNCS, vol. 9170, pp. 563–572. Springer, Cham (2015). doi:10.1007/978-3-319-20916-6_52