Electronic Communications of the EASST
Volume 25 (2010)

Proceedings of the Workshop Visual Formalisms for Patterns at VL/HCC 2009

Pattern Catalogs using the Pattern Language Meta Language

Andreas Wolff, Peter Forbrig

11 pages

Guest Editors: Paolo Bottoni, Esther Guerra, Juan de Lara
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Pattern Catalogs using the Pattern Language Meta Language

Andreas Wolff[1], **Peter Forbrig**[2]

[1] andreas.wolff@uni-rostock.de [2] peter.forbrig@uni-rostock.de
http://wwwswt.informatik.uni-rostock.de
Universität Rostock, Germany
Institut für Informatik

Abstract: This article focuses on the pattern language PLML. Some enhancements and corrections to it are proposed to make use of PLML in pattern catalogs. Additionally, a textual domain specific language as human-readable variant of PLML is proposed. Supporting editors, textual and graphical, which were developed using model-based techniques are presented.

Keywords: HCI pattern, pattern language, model-based editor

1 Introduction

Object-oriented design patterns, as introduced by Gamma et al. [GHJV02], are considered a valuable aid in software development. Patterns were identified in many other domains of computer science. One such domain is human computer interaction (HCI). HCI patterns do exist for many different aspects of the user interface of an application. For example for navigation through an interface, its layout, input modalities or presentation.

A number of pattern catalogs has been compiled by the HCI community. Well known examples are the catalogs of Tidwell [Tid] and Van Welie [vW]. On examining HCI patterns in those catalogs one soon discovers certain problems:

- There is no consistent naming of patterns across those collections. Multiple entries for what is essentially the same pattern are likely.
- Intra-catalog references are rare, cross-catalog references almost non-existant.
- Aspects of a pattern that are detailed in a pattern entry differ. This is not only in naming but also in extent. (e.g. a problem or solution description is omitted)
- The abstraction level differs from pattern to pattern.
- The pattern solution is given very informal, mostly text accompanied by pictures for illustration purposes.

Of course, the findings above are not novel. There are approaches to standardize pattern catalogs. The XML dialect PLML [Fin03] is such an attempt. It was developed to define a common base of how to describe patterns. PLML defines a pattern language and an exchange format. It was constructed to cover generic patterns, not for HCI patterns in particular. Without having a specific domain as boundary, it was not possible to restrict the language elements beyond very

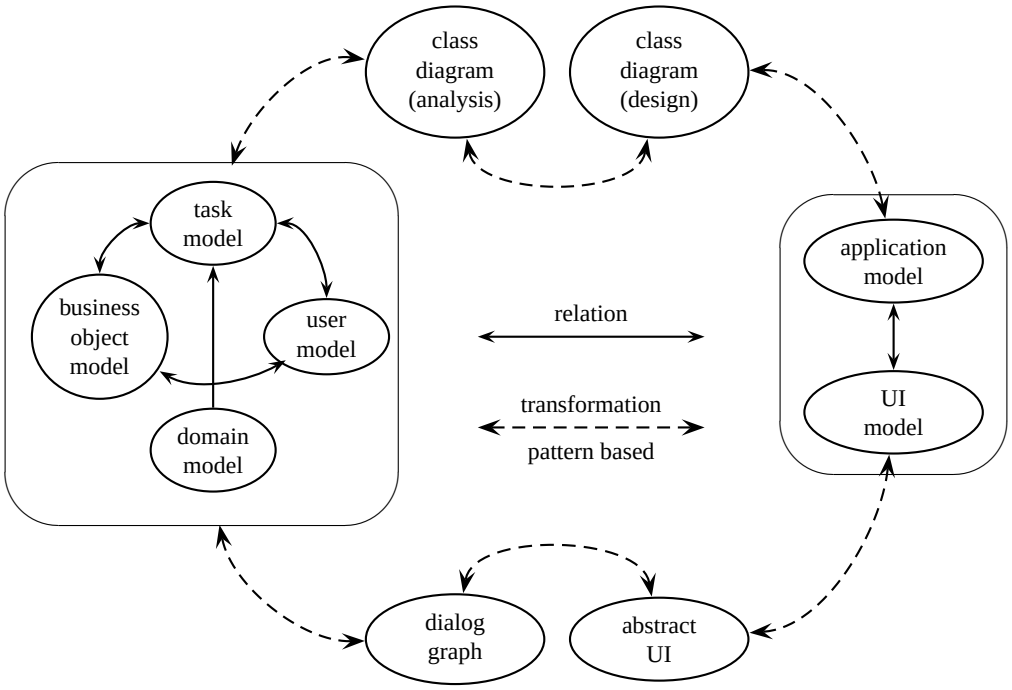

Figure 1: General view on a transformational model-based development process

basic constraints. Therefore PLML itself does not overcome the lack of formalization problem mentioned earlier. Section 2 of this paper presents PLML in some detail and also introduces some enhancements proposed by us.

In our research we develop a pattern-based approach to model-driven user interface engineering. Therein we consider software development as a sequence of transformations of models. Figure 1 shows the source and target models and the in-between transformations of our approach. Further details of this process are described for example in [WFDR05].

More important, than the details of our approach, for the scope of this paper is the ubiquitous use of pattern-based transformations. One consequence of this was the need to specify those transformations. Our goal always was to create a semi-automatic human-controlled overall process, which will be supported by appropriate tools for every transformation step. Therefore we had to store patterns in a machine readable manner.

The current focus of our research is the lower part of Figure 1, the generation of user interfaces (UI). We attempt to represent HCI-patterns for UIs in a suitable way so they can be used within our MDA process. This is done by transforming the pattern idea into a so-called pattern instance component (PIC). Such a PIC is basically an attributed template that may include some programming logic. It is called instance component, since we consider the template to be already an instance of the pattern that is described through this component. We are aware of the fact that, due to their nature, not all known HCI patterns can be treated as or translated into an algorithm or a PIC.

Comparable work has been done for the original Gamma patterns by Arnout [Arn04], who

investigated and, where possible, created usable components of design patterns for the Eiffel programming language.

The language used for our pattern instance components is domain specific to user interfaces, it was outlined in [RWF06]. Nevertheless, it is a XML-based language and therefore it is possible to contain it in or link to it from most existing pattern languages.

As our whole approach is model-based, we developed a pattern language that also is based on models. It is compliant to PLML and able to hold PICs. Also it is possible to store pattern descriptions of patterns whose solution cannot be specified using a PIC. The rest of this paper presents this pattern language. Section 3 shows a textual domain specific language (DSL) as interface to our catalog. In Section 4 a possile graphical interface is discussed and generated as model-based graphical editor.

2 Pattern Language Meta Language

2.1 Overview PLML

The pattern language meta language PLML was developed by a group of stakeholders during a workshop. The idea behind, was to formalize the description of patterns to eventually merge all existing pattern languages or at least to have a general interchange format. It was designed to be able to describe patterns on any abstraction level and of any domain. Beside the pure pattern description a number of identifying meta data became standardized.

```
<!ELEMENT pattern (
    name?, confidence?, alias*, synopsis?, illustration?,
    context?, problem?, forces?, evidence?, solution?, diagram?,
    implementation?, related-patterns?, pattern-link*,
    literature?, management?)>
<!ELEMENT management (
    author?, revision-number?, creation-date?, last-modified?,
    change-log?, credits?)>
<!ATTLIST pattern
  patternID CDATA #REQUIRED
  collection CDATA #REQUIRED
>
<!ATTLIST context mylabel CDATA #IMPLIED>
<!ATTLIST pattern-link
  type CDATA #REQUIRED
  patternID CDATA #REQUIRED
  collection CDATA #REQUIRED
  label CDATA #REQUIRED
>
```

Listing 1: DTD of the PLML standard [PLM]

Listing 1 shows the document type definition of PLML, official version 1.1.2. Every element that is not explained in further detail is of type #PCDATA or ANY. Table 1 has a short description for all pattern related language elements. The meta-data elements of *management* seem to be self

Element	Description
patternID	Collection-unique id of a pattern
name	Name, as short as possible
alias	Alternative names
illustration	Picture that illustrates a particularly good pattern instance
problem	Design situation which the pattern addresses
context	Conditions when application of a pattern is most useful
forces	Forces which are resolved by application of the pattern
solution	Instructions to follow the idea of the pattern
synopsis	Summary of the pattern idea
diagram	Schematic visualization of the pattern, sketched or formal
evidence	Justification that a pattern actually is a pattern by:
example	- known uses
rationale	- principled reasons, axioms, common sense or the like
confidence	Star rating whether the entry is a true pattern (0 to 2 stars)
literature	References to related work
implementation	Code or fragments of code or other technical documentation
related-pattern	Container for connections to other patterns
pattern-link	Connection to other pattern:
type	- kind of connection, either of *is-a, is-contained-by, contains*
patternID	- connection endpoint
collectionID	- connection endpoint in collection named collectionID
label	- descriptive text of connection

Table 1: Summarization of the meanings of PLML's language elements [Fin03]

explanatory.
As mentioned before, the degree of freedom Listing 1 permits was modeled deliberately. However, some aspects that are discussed in the workshop report [Fin03] were not put into the standard. Additionally there is a lack of meta-data storage area if PLML is used to actually build a pattern catalog. Subsection 2.2 outlines our enhancements to PLML and provides a rationale for each amendment.

2.2 Enhancing PLML

Building a pattern catalog using PLML, as defined in Section 2, quickly reveals a first problem. There is no root element to attach the entries to. Also no XML-tags are reserved to describe the catalog itself, i.e. its authors, revision or certain dates. Listing 2 introduces such a catalog root element to meet these basic requirements. Attributes are provided to name a catalog and assign a global identificator to it. The latter feature is needed for cross-catalog pattern-links. Other, catalog describing, meta-data is stored using the already existing element *management*.

```
<!ELEMENT catalog (management, pattern+)>
<!ATTLIST catalog
```

```
id CDATA #REQUIRED
name CDATA #IMPLIED >
```

Listing 2: Introducing catalog meta information into PLML

The PLML explanation in [Fin03] explains that the *confidence*, whether an entry is a true pattern, is expressed using a star rating. It is possible to include this into the document type definition of PLML. Listing 3 defines an attribute *level* for the element *confidence* and limits its value space to one of the three entries. The element *confidence* is redefined to not contain content any more. Through this amendment the notation of confidence levels is fixed and therefore easy to parse.

```
<!ENTITY % confidenceLevels "(0|*|**)">
<!ELEMENT confidence EMPTY>
<!ATTLIST confidence level %confidenceLevels; #REQUIRED>
```

Listing 3: Narrowing the value space of confidence information

The PLML standard often refers to illustrations to describe certain aspects of a pattern. Nevertheless, there was little support to actually integrate pictures or at least resource locators of pictures in a catalog. While it is possible to include pictures as binary data in an XML document, we resorted to annotate URLs of pictures in our pattern entries. Listing 4 defines optional *url* attributes for three PLML elements.

```
<!ATTLIST context url CDATA #IMPLIED>
<!ATTLIST diagram url CDATA #IMPLIED>
<!ATTLIST example url CDATA #IMPLIED>
```

Listing 4: Annotating picture URLs to certain elements

Another notable change to the original standard was to allow *pattern-link* only as subelements of *related-pattern*. Also there were minor changes to the representation of *management* meta-data.

3 PLML as textual DSL

PLML was designed as storage and interchange format, human-readability was not of primary concern. But, when writing and maintaining a catalog it is sometimes inconvenient and error-prone to edit in XML structures. There are different ways to cope with this situation. One solution is to use dedicated XML editors or to develop a specialized editor.
Since we work in a model-based context we decided to create a dedicated PLML editor using model-based technology. We defined a grammar for a textual domain specific language (DSL) resembling PLML. From this grammar we generate a fully featured text editor and a meta-model of PLML. Having a proper meta-model enables us to easily include the patterns of the catalog within our software engineering approach of Figure 1. The meta-model also is a prerequisite to keep our pattern catalog compliant to PLML standard. Using the model it is very straightforward to generate valid PLML from our modified pattern language and thus adhere to the interchangeability concept of PLML.
The grammar of our textual DSL is a xText [EV] grammar. XText is a model-based framework for such DSLs. It is a plugin to the Eclipse rich client platform. Provided a grammar, xText

generates a text editor and an EMF Ecore [emf] meta-model. Through simple customizations this text editor supports:

- Syntax highlighting, keywords or structuring elements are emphasized
- Syntax completion, there is an editing help which offers keyword completion within the text editor
- Error checking, the text is permanently checked against the language grammar and violations are marked within the editor.
- A structured outline view provides an overview and simple navigation through the catalog.

A xText grammar is context-free and its notation is EBNF-like. Nonterminal symbols start with a capital letter, terminal symbols are in quotation marks. Several syntax elements and predefined types were introduced through xText to derive better meta-models.

```
RelatedPatterns:
  "RelatedPattern" "{" (references+=PatternLink)* "}";
PatternLink:
  type=PatternLinkType "id=" targetId=STRING
  (hasCollection?="collection=" collection=ID)?
  (hasLabel?="label=" label=STRING)?;
Enum PatternLinkType:
  isA="is-a"|containedIn="is-contained-by"|contains="contains";
```

Listing 5: Grammar for a pattern-link entry

Listing 5 is an excerpt from the xText grammar of PLML. It shows the specification of pattern relations. A *pattern-link* is specified using the nonterminal *PatternLink*. *PatternLink*s occur within the nonterminal *RelatedPatterns*. *RelatedPatterns* is defined as starting with the terminal (or String) "RelatedPattern" followed by zero-to-many *PatternLink*s included in curly braces within the text. In the meta-model every *PatternLink* will be stored in the references aggregation of the RelatedPatterns type. HasCollection and hasLabel are simple markers, they are interpreted as boolean values to easily check whether a collection or label value was set at all. The enumeration type *PatternLinkType* restricts the value space of the *PatternLink* type to one of the three values, this is very much like in Listing 3 for confidence level stars.
The xText framework parses the grammar into an ANTLR grammar. ANTLR [ant] is an object oriented parser generator. XText uses ANTLR to generate parser and lexer for the text editor.

```
Catalog SampleCatalog
MasterData {
 Author "Andreas Wolff"
 CreationDate 2008-12-31
 Revision 1.0.2
}

Pattern ync "Yes-No-Cancel" {
```

```
 Alias "feedback" collection hciPatterns
 Problem "Decide a binary question or cancel the current operation"
 Evidence {
  Rationale
  "A user should be able to cancel an operation,
  not only decide between two possibly undesired consequences"
  }
 Confidence **
 RelatedPattern {
  contains id="yn"
 }
 MasterData {
  Author "andreas"
  CreationDate 2009-02-15
  Revision 1.0
  }
}
```

Listing 6: Catalog containing one pattern entry using our textual DSL

In Listing 6 a sample catalog can be seen. This catalog contains only a single pattern description, the "Yes-No-Cancel" pattern [Pet] which got the id "ync" assigned. The confidence in "ync" being a pattern is high and it is related to a pattern "yn" within the same catalog. "ync" is also known as the pattern "feedback" in another catalog which has the id "hciPatterns". A problem description and a rationale for this pattern are given textually, no illustration or diagrams are used. Listing 6 is a minimum example to illustrate the idea.

Figure 2 shows the meta-model which was derived from the grammar. Again, to reduce complexity, it is only an excerpt of the complete model. Type *Management* is actually not an empty class. Catalog management data has been left out too. The enumeration *PatternLinkType* which consists of three constant values is also not displayed.
While the pattern catalog itself is a plain-text file, the data it contains is accessed via instances of the meta-model. XText parses the catalog text file and builds such instances from the textual entries. The root class of the model is the type *Pattern*, whose objects have an association to an object of class *PatternDescription*. *PatternDescription* is a container class where most aspects of a pattern are defined by creating aggregrations to their respective types.

4 Graphical representation of PLML

The meta-model of PLML is a useful tool. It can be used to generate a graphical editor or viewer for PLML based catalogs. There is another model-based framework to construct such editors from EMF Ecore models, the Eclipse Graphical Modeling Framework (GMF) [gmf]. GMF provides runtime components and a generator framework to build graphical editors, these editors are generated as Eclipse plugins. A number of auxiliary models are necessary within this process:

- **GMFGraph** is the model to define figures. The appearance of any diagram element is

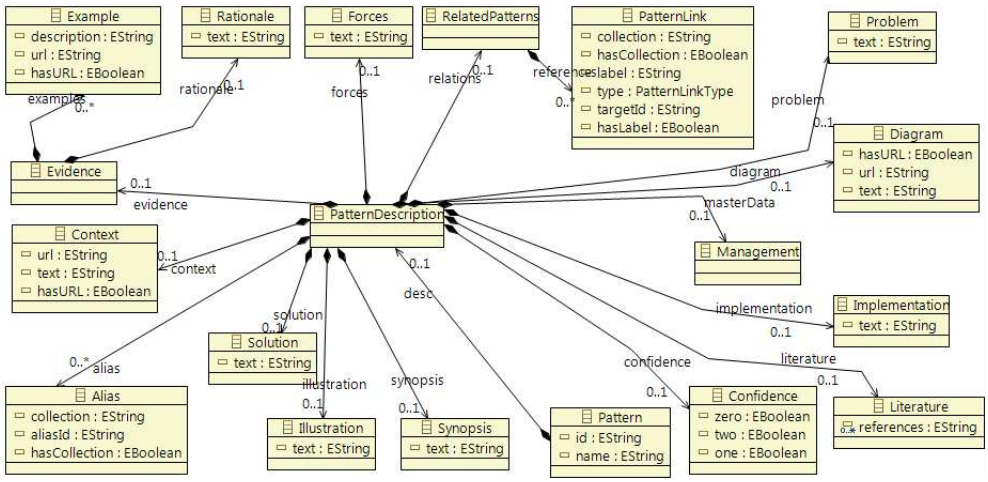

Figure 2: Overview of the meta-model of PLML

described here.

- **GMFTool** describes the various menus, toolbars and the palette of an editor.
- **Ecore** is the underlying meta-model of the application domain.
- **Genmodel** is for generating source code for access and modification of the model instances.
- **GMFMap** merges above four models. The mapping model defines how meta-model elements are mapped on diagram nodes or connections. Other mappings include the connection between menu entries and diagram nodes.
- **GMFGen** is the basis for editor generation. It is derived from the mapping model GMFMap. As it is the very last step before generating the actual editor, a lot of fine tuning can be executed here.

An important decision for every meta-model class and attribute is whether and how to display it in an editor. Basically every such element can be a free-form figure or a connection link or a label or a composed figure or have no figure at all. Property sheets are assigned to every class figure. This way properties that are not displayed as a figure are editable anyway. In GMF-editors we can also choose to display elements as container, i.e. that they can have children within their graphical representation.

For the initial PLML meta-model we map the classes *Pattern*, *PatternDescription* and *Management* onto graphical containers. *Pattern* will be the root container of pattern description and its associated master data. *PatternLink*s will be editable as connections and have labels about their type attached. Attributes of type string are mapped to labels and can be edited in place. They are arrange into their appropriate category, either management or pattern description data. Boolean attributes and the link type of a pattern relation can only be edited using the property

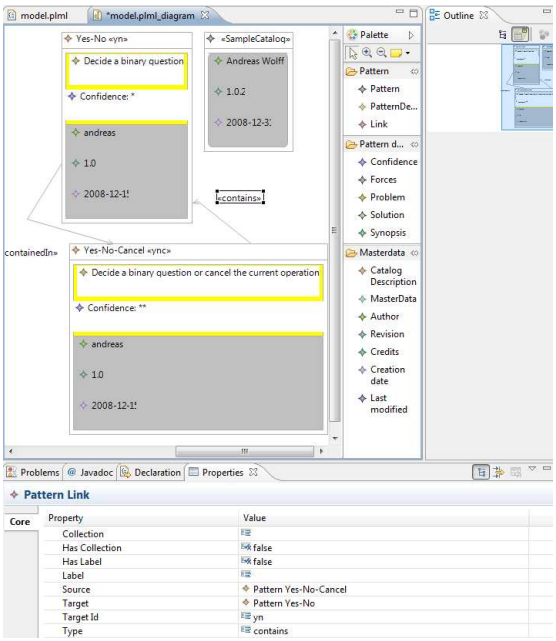

Figure 3: Graphical PLML editor in Eclipse

sheet.

After all mappings are done GMF generates a complete graphical editor as Eclipse plugin. Figure 3 is a screenshot of such an editor. For this specific editor the master data of the catalog itself was mapped to an own graphical element. Two patterns are defined in this catalog and a bi-directional relation exists between them. The lower pattern node (Yes-No-Cancel) is the result of the textual definition in Listing 6. The other pattern node for a Yes-No pattern is defined very similar. Only the containment relation was reversed and the confidence for Yes-No to be a pattern was set to 1-star. In Figure 3 the pattern relation ”contains” between Yes-No-Cancel and Yes-No was selected within the editor window. The property sheet below shows all attributes of this relation. Changing the attribute values here directly effects the content of the editor.
The property sheet of the link reveals two attributes which are not defined in the meta-model of Figure 2. Attributes *source* and *target* were introduced while preparing the GMFMap mapping model, both are of type *Pattern*. They were specified to be derived and transient, i.e. their value is a direct consequence of other attribute values and they therefore need not to be serialized. The value of *target* for example is the pattern object whose id equals the targetID of the *PatternLink*.

Another notable difference is in *Confidence*. The meta-model declares three boolean attributes, this is a direct result from xText's grammar to model transformation. I.e. these attributes are a technical necessity, but somewhat impractical. To visualize the true confidence level a star-like labeling was desired. So again an transient derived attribute *label* was introduced whose value is displayed instead of the booleans.

Of course, the concrete values of *target*, *source* and *label* need to be calculated somewhere. To achieve this we had to leave the model level and actually write some source code. Through

modification of the meta-model edit code, which was generated using the **Genmodel**, getting and setting the value of the derived attributes also modifies the underlying data attributes.

Using GMF for the graphical editors enables us to quickly modify the editors. Since most of the editors source code is generated from the models we can easily try out many kinds of visualizations. This is not only about the layout or form or color of a certain diagram node. Through changes in the mapping model we could pursue a completely different idea of graphical containment. But of course, all such editors would only be different views on the very same data. The catalog data itself will always be serialized using the textual DSL of Section 3.

5 Conclusion

This paper describes the pattern language PLML. It is a XML-based language which can be used to define pattern catalogs. Certain modifications to its original standard were proposed in the paper to repair some issues with applying PLML in a real catalog.
PLML is used as the pattern container in an integrated model-based environment. Therefore the pattern language was backed with model-based editing tools. A textual DSL has been presented that makes it easy to edit the pattern catalog using standard text editors. At the same time this DSL is the meta-model or rather forms the foundation of an EMF Ecore meta-model of the pattern language.
Two separate editors for the DSL where derived from models. The first is an advanced text editor, that supports typical developer features like syntax highlighting and auto-completion. A second editor is a graphical editor that was developed using the eclipse graphical modeling framework. It was shown that such generated editors are highly flexible in terms of the graphical representation of the language elements.

Bibliography

[ant] ANTLR - Another Tool for Language Recognition. http://www.antlr.org, last visited 6th June 2009. University of San Francisco.

[Arn04] K. Arnout. *From Pattern to Components*. PhD thesis, ETH Zurich, 2004.

[emf] Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/?project=emf, last visited 6th June 2009. Eclipse Foundation.

[EV] S. Efftinge, M. Voelter. oAW xText: A framework for textual DSLs. http://eclipsesummit.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium12_xTextFramework.pdf, last visited 6th June 2009.

[Fin03] S. Fincher. Perspectives on HCI patterns: concepts and tools (introducing PLML). In *Workshop at CHI 2003*. Sept. 2003.

[GHJV02] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 24th edition, 2002.

[gmf] Eclipse Graphical Modeling Framework Project. http://www.eclipse.org/modeling/gmf/, last visited 6th June 2009. Eclipse Foundation.

[Pet] R. Petrasch. Model Based User Interface Design: Model Driven Architecture und HCI Patterns. http://pi.informatik.uni-siegen.de/stt/27_3/03_Technische_Beitraege/MDA_HCI_Patterns_Petrasch_Short.pdf, last visited 6th June 2009.

[PLM] DTD of PLML. http://www.hcipatterns.org/tiki-download_file.php?fileId=7, last visited 6th June 2009.

[RWF06] R. Rathsack, A. Wolff, P. Forbrig. Using HCI-Patterns with Model-based Generation of Advanced User-Interfaces. In Pleuss et al. (eds.), *Proceedings of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces*. Genova, Oct. 2006. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-214/

[Tid] J. Tidwell. Pattern library. http://www.designinginterfaces.com, last visited 6th June 2009.

[vW] M. van Welie. Pattern library. http://www.welie.com/patterns/index.php, last visited 6th June 2009.

[WFDR05] A. Wolff, P. Forbrig, A. Dittmar, D. Reichart. Development of Interactive Systems Based on Patterns. In *Workshop on Mapping User Needs into Interaction Design Solutions at Interact*. Rome, Italy, Sept. 2005.