Louvain School
of Management

Working paper 08/13
Generative Pattern-Based Design of User Interfaces

Francisco Montero Simarro, Jean Vanderdonckt

Academic Year 2007- 2008

Louvain School of Management

Generative Pattern-Based Design of User Interfaces

Francisco Montero Simarro[1], Jean Vanderdonckt[2]
[1]*Laboratory on User Interaction & Software Engineering (LoUISE)*
University of Castilla-La Mancha, 02071 Albacete (Spain)
fmontero@info-ab.uclm.es – http://www.info-ab.uclm.es/personal/fmontero
[2]*Université catholique de Louvain, Louvain School of Management*
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
jean.vanderdonckt@uclouvain.be – www.isys.ucl.ac.be/bchi/members/jva

Abstract

This paper introduces a knowledge-based method for developing user interfaces based on generative patterns instead of descriptive patterns. The knowledge base contains generative patterns from which portions of previously designed user interfaces could be identified and re-applied to a new design case study by generating code from its functional specifications. The method introduced in this paper is relying on models that are typically involved in user interface development such as task, domain, abstract user interface, concrete user interface, final user interface, context model, and mappings between them. In this way, any type of model can virtually be the source of a pattern and can be described, searched, matched, retrieved, and assembled together so as to create a new user interface. The method is supported by IDEALXML, a software that can be used for pattern-based design of user interface based on the USIXML user interface description language. In order to support pattern-based design, the USIXML language has been expanded with concepts addressing problems raised by pattern description and matching related to models pertaining to user interface development. This extension could also be considered as an extension of PLML (Pattern Language Markup Language) that has been introduced to uniformly represent user interface patterns.

1. Introduction

Since a more than two decades, design patterns [1,3,13,30] have received much attention in various domains of the human activity, including software engineering [8], software development [5], and User Interface (UI) design [15,33] with the conviction that parts or whole of any UI that has been designed for a past interactive application may be reused later in another, perhaps similar, application. In addition, design patterns are also frequently expressed as a comprehensive way to communicate pairs of (problem, solution) in a manner that remains largely applicable, and more general than usability guidelines [34]. In particular, usability guidelines were criticized for loosing the context where they are applicable or for not providing this context explicitly for applying the guidelines [3].

The CHI'2003 workshop on UI Patterns [12] observed that many different, probably inconsistent, sources of UI design patterns exist today [10,11,29,33], thus raising the need for a common pattern language to express UI design patterns. This resulted into the Pattern Language Markup Language (PLML) [12] specification. The main goal of PLML was to bring some structure and consistency to the many forms that have been used by pattern authors. PLML became more widely applied as several pattern collections have been translated into this format, thus facilitating comparison, reuse, and linking between various collections. But PLML is a natural language-based way for writing patterns. Since any form of natural language always suffers from intrinsic problems like ambiguity, inconsistency, PLML did not escape from these problems that prevent this language from being directly used as a pattern format for a pattern-based UI design process [15] that is effectively and efficiently supported by software.

Lack of expressivity: several PLML tags express various aspects of a pattern that were believed of sufficient general interest, but some are missing. For instance, a tag describes the forces of a pattern, but no other tag describes the counter-forces.

Flat definition: PLML is defined in a Document Type Definition (DTD) in a flat structure that does not allow structured pattern-matching and searching.

Lack of separation of concerns: the current PLML definition mixes the expression of several concepts together, thus violating the principle of separation of concerns where different aspects are captured in different independent models. For instance, the context definition is completely embedded in a general tag without

being further refined. Therefore, it is not guaranteed that various patterns, if they are relevant to a same context, will be expressive enough to be found out for a similar context of use, unless the person who encodes the context description devotes a lot of attention.

Lack of structure: several tags are only defined in a general way (e.g., a string), with no further decomposition, thus leaving the definition very open and flexible (which is an advantage), but hindering a structured use of the tags by a software (which is a shortcoming for large and efficient use). Fig. 1 shows that this flat structure does not facilitate much rigorousness.

```
<!ELEMENT  pattern (name?, alias*, illustration?, problem?, con-
           text?, forces?, solution?, synopsis?, diagram?, evi-
           dence?, confidence?, literature?, implementation?, re-
           lated-patterns?, pattern-link*, management?)>
<!ATTLIST  pattern patternID CDATA #REQUIRED>
<!ELEMENT  name (#PCDATA)>
<!ELEMENT  alias (#PCDATA)>
<!ELEMENT  illustration ANY>
<!ELEMENT  problem (#PCDATA)>
<!ELEMENT  context ANY>
<!ELEMENT  forces ANY>
<!ELEMENT  solution ANY>
<!ELEMENT  synopsis (#PCDATA)>
<!ELEMENT  diagram ANY>
<!ELEMENT  evidence (example*, rationale?)>
<!ELEMENT  example ANY>
<!ELEMENT  rationale ANY>
<!ELEMENT  confidence (#PCDATA)>
<!ELEMENT  literature ANY>
<!ELEMENT  implementation ANY>
<!ELEMENT  related-patterns ANY>
<!ELEMENT  pattern-link EMPTY>
<!ATTLIST  pattern-link type CDATA #REQUIRED
           patternID CDATA #REQUIRED
           collection CDATA #REQUIRED
           label CDATA #REQUIRED>
<!ELEMENT  management (author?, credits?, creation-date?, last-
           modified?, revision-number?)>
<!ELEMENT  author (#PCDATA)>
<!ELEMENT  credits (#PCDATA)>
<!ELEMENT  creation-date (#PCDATA)>
<!ELEMENT  last-modified (#PCDATA)>
<!ELEMENT  revision-number (#PCDATA)>
```

Figure 1. Document Type Definition of PLML.

In order to address these shortcomings, a method for developing a UI based on patterns is introduced:

1. A definition of *models* involved in UI design, which can then be mapped to a pattern.
2. A specification of these models and the pattern according to a single and consistent *User Interface Description Language* (UIDL).
3. A definition of the method steps with these models.
4. A software for supporting the method called IDEALXML (**I**nterface **D**evelopment **E**nvironment for **A**pp**L**ications specified in UsiXML**L**).

2. Related work

A pattern must be *useful* because this shows how having the pattern in mind may be transformed into an instance of the pattern in the real world [1], as something thing that adds value to our lives as developers and practitioners. A pattern must also be *usable* because this shows how a pattern described in literary form may be transformed into a pattern that we have in our mind. And a pattern must be *used* because this is how patterns that exist in the real world first became documented as patterns in literary form. In the next subsections, we discuss how UI patterns have been tried to become useful, usable, and used.

2.1 Patterns compilation

We can find many references where design patterns in Human-Computer Interaction (HCI) or interaction patterns appear. Compilations of those references can be found in, for instance, *The interaction design patterns page* [10], *The pattern gallery* [11], *HCI patterns pages* [4] or *Interaction design patterns* [33]. In those compilations, several ways of documenting the same type of contents can be identified from natural language to XML-based formats. Manipulating interaction patterns is very difficult and is necessary to provide additional assistance in order to use them in a (semi-automatically) way. In this sense, only a few proposals are available where designers can work using patterns. In software engineering, notations exist like UML and tools where design patterns [14] can be used together. Design patterns are documented using class diagrams from which guidance is provided to designs on how to use them. These tools are not available in other fields like HCI because UI patterns in this field are difficult to use, to document, to compare, and to know. Using UI patterns typically requires assistance for identifying, selecting, adapting, and integrating them. These tasks should be supported by tools to become really usable. For this purpose, the pattern documentation should be improved prior to making them available in tools. Using only natural language is not enough in order to work efficiently with patterns. Other forms exist. Thus, there is no universal way to write a pattern.

Patterns are often referred to as being *descriptive* when they basically consist of a description of the pattern, its problem, the context in which the problem is posed, and the potential solutions that can be brought to solve the problem. Patterns are one form of establishing a mapping between the problem space and the design space. Descriptive patterns are intended to be used mainly by human such as project leaders, designers, analysts, and developers. Descriptive patterns usually seek to maximize *descriptivity* (i.e., the ability of a pattern to be described in details enough to become self-contained) and *genericity* (i.e., the ability of a pattern to be applicable to the widest problem space possible by interpreting the description for a particular

context of use). As opposed to descriptive patterns, patterns are said to be *generative* when they subsume an object-oriented representation that can be automatically obtained in order to generate the final code. Generative patterns are intended to be used by automata (e.g., algorithms, program analysis and synthesis techniques). Generative patterns usually seek to maximize *expressivity* (i.e., the ability of a pattern to be expressive enough so as to obtain a working system) and *generativity* (i.e., the ability of pattern to be expressed in a way that facilitates automated generation of code).

In our context, generative patterns tell us how to create a UI that can be observed in the resulting interactive system to be developed. Non-generative patterns describe recurring phenomena without necessarily saying how to reproduce or to concretize them in a particular interactive application. We should therefore strive to document generative patterns because they not only show us the characteristics of good UIs (e.g., they could also convey information about usability [34]) that are appropriate in their context of use, but they also teach us how to build them or, in other words, to develop them. This does not mean that descriptivity should be left out. We here argue for a UI pattern scheme that combines both the qualities of descriptive and generative patterns by defining a UI pattern template containing both descriptive and generative aspects as opposed to one single dimension at a time.

Therefore, we are expecting to introduce genuine patterns that will maximize all the four properties of purely descriptive and generative patterns. Indeed, descriptive patterns are usually estimated of high genericity and descriptivity, but weak in generativity and expressivity (Fig. 2). Generative patterns are located in an inverse situation: they are rich in generativity and expressivity, but weak in genericity and descriptivity. By combining the qualities of both families of patterns into the genuine patterns, we need to reach a high level for all four properties simultaneously (Fig. 2).

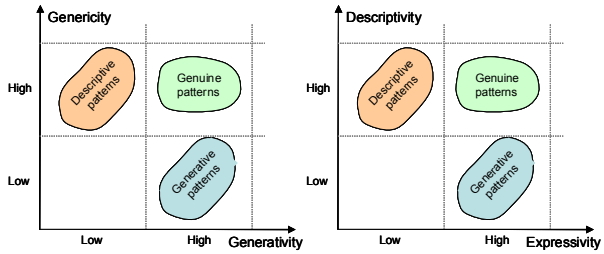


Figure 2. Classification of patterns compilations according to the four properties.

2.2 Pattern software

Different software exists today for supporting the process of using UI patterns. Environments exist where patterns can be introduced [29], suggested [16], viewed [33] or used to develop prototypes [26].

CanonSketch [7] is a tool to describe user interfaces using the notation of Canonical Abstract Prototypes [9]. Introducing a UI using this notation which is independent of any technology represents a generative pattern since HTML code can be automatically generated from the description. However, no other information about the pattern is provided. The Montreal Online Usability Patterns Digital Library [29] is an Integrated Pattern Environment (IPE) that was originally designed with two major objectives: as a service to UI designers and software engineers for UI development and as a research forum for understanding how patterns are really discovered, validated, used and perceived. MOUDIL consists of a pattern editor, a pattern navigator and a pattern viewer. In this way, it supports descriptive patterns effectively, but needs to be connected with other tools to give rise to a running UI.

Greene [16] developed a software prototype to support pattern-assisted design and development. The software supports the pattern creating, browsing, viewing, and editing, but most importantly, it provides decision support to help filter and select patterns based on criteria or drivers specified by the pattern authors as relevant to particular patterns. Internally, patterns are stored as XML documents. Pattern elements are the fields or properties of the patterns (e.g., 'Name', 'Problem', 'Forces', 'Context', 'Solution', etc.). There is a default set of such properties, but, since there is of yet no accepted standard set of properties, this set is definable and extensible by the pattern language author. One can define different pattern types with different fields and links between patterns may be user-defined and typed, thus providing mechanisms that are adequate for making a true knowledge base of patterns. One can search for patterns that contain specified strings in all, or any subset of the fields of the patterns. Although there are currently two decision support mechanisms embodied in the tool to identify appropriate patterns, it does not produce any running UI.

MESCA [18] consists of a knowledge base of UI elements that are considered as patterns. Its advantage relies in its case-based reasoning algorithm for finding out similar UI elements based on search criteria. Again, it does not produce any running UI. The PIM tool [25] probably represents the most advanced tool for UI patterns which are both descriptive and generative: it stores models in the XIML (www.ximl.org) and allows several degrees of pattern searching.

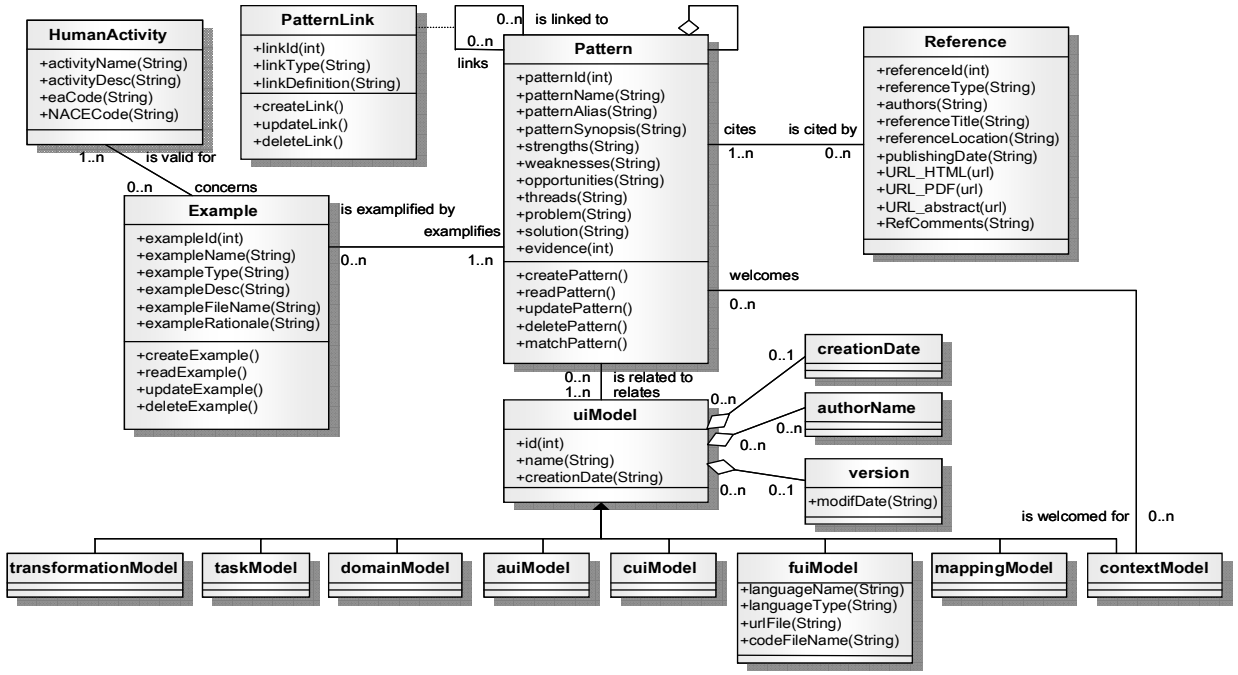


Figure 3. UML Class diagram of a UI pattern extended from PLML [12].

2.3 Methodologies

In the area of Model-Driven Engineering (MDE), several methodologies exist that support the development life cycle of interactive applications, such as UML-based methodologies. WISDOM [7] or IDEAS [23] are object-oriented, they use the UML to specify, visualize, and document the artifacts of the development project. They have been adapted to develop interactive applications because UML does not support UI design. WISDOM and IDEAS evolve incrementally through an iterative process. Other approaches are task centered [26], pattern-oriented [17,23,27,30], involve different techniques such as usability engineering [17], interaction templates [26], multiple design [27], and MDE [28,30]. They provide a methodological guidance on how to use patterns but, again, are not generative. A major observation is that a UI pattern may be informed by many different types of contents belonging to different models which are not all necessary at once, but which could be considered individually when needed. Next, we introduce our representation of a UI pattern so that it is both descriptive and generative.

3. Conceptual model of UI patterns

The PLML [12] language, resulting from a consensus obtained during the CHI'2003 workshop on patterns, is certainly a reference base to be considered for extension. Therefore, by relying on specifications as described in Fig. 1, we extended PLML into a UML Class Diagram for representing UI patterns that are both descriptive and generative (Fig. 3). We now justify why these extensions have been required.

Obviously, each UI pattern should be properly identified; therefore we need an identifier (patternID), a meaningful short name (patternName), an alternate name (patternAlias), and a pattern general description (patternSynopsys). PLML only provides the forces of a pattern as recommended by Alexander [1]. We believe this should be expanded: when we write a pattern the notion of force generalizes the kinds of criteria that software engineers use to justify designs and implementations. But these forces should be counterbalanced with other dimensions which are typically found in the SWOT analysis, a tool for auditing an organization and its environments with four axes: strengths, weaknesses, opportunities, and threads. Strengths and weaknesses are internal factors and opportunities and threads are external factors. Forces are related with the 8 major ergonomic criteria as defined Bastien & Scapin (i.e., compatibility, consistency, work load, dialog control, adaptation, guidance, and error management [2]). By expressing which ergonomic criteria are respected (or addressed), we know in advance the quality of pattern and their purpose. If we want to maximize consistency, patterns related to consistency could be selected from the knowledge base.

The *evidence* scale (evidence) provides an indication of how seriously designers and developers should consider each pattern. A five-point Likert scale is used to depict the evidence related to each pattern:

- 5: two or more experiments support the pattern.
- 4: one experiment supports the pattern.

- 3: two or more studies support the pattern.
- 2: one study supports the pattern.
- 1: one or more observations and no other supporting evidence support the pattern.
- 0: no evidence supports the pattern.

In order to properly link patterns to each other, which is important for not forgetting related or potentially contradicting patterns, a taxonomy of relationships (patternLink) between patterns has been defined: X *uses* Y in its solution, X *is a variant* of Pattern Y, X *has a similar problem* as Y, X *is related in the related patterns section* to Y, X *specializes* Y (in the sense of pattern inheritance), X *connects* to Y as part of the sequence S, in this case, the label includes S and a descriptive text that serves as the glue text in the sequence, X *mentions* Y in its context, this means that Y was applied before Y, X and Y are members of the same class or family, X and Y involve a common participant P and X and Y can be found in the same known context of use U. The problem provides a description of the problem space covered by the pattern while the space attribute describes the solution space ensured by the pattern. Another factor of confidence we can assign to a pattern comes from the bibliographic reference (reference) where it is defined: a pattern defined by an organization, an expert or a practitioner may widely differ in its scope and purpose. For instance, a pattern recommended by an official standardization body could be considered as stronger than a pattern provided by an individual person.

It is very important to document examples showing the application of a pattern so as to facilitate its interpretation and its application [34]. Therefore, example contains a description of a supportive example demonstrating the applicability, the non-applicability, or an exception of the pattern. Each example could be associated, if needed, to one or several domains of human activity (humanActivity) that characterize whether a pattern is generic or specific to a domain. In this way, it is also possible to search the knowledge base of patterns for patterns that are applicable to a particular domain, say for instance chemistry, medical record of patient, museum visits, etc. This concludes the upper part of Fig. 3 containing the descriptive explanatory power of a UI pattern. The below part of Fig. 3 represents the generative power as it relates the pattern to any combination of UI models involved in the Cameleon Reference Framework [6] for developing multi-target UIs, which is decomposed into four steps [6,31,32]:

1. *Task and domain modeling* (Computing Independent Model in MDA): a model is provided for the end user's task, the domain of activity and, if needed, the context of use (user, computing platform, and environment).
2. *Abstract User Interface modeling* (Platform Independent Model in MDA): this level describes potential UIs independently of any interaction modality and any implementation technology.
3. *Concrete User Interface modeling* (Platform Specific Model in MDA): this level describes a potential UI after a particular interaction modality has been selected (e.g., graphical, vocal, multimodal). This step is supported by several tools helping designers and developers to edit, build, or sketch a user interface.
4. *Final User Interface*: this level is reached when the UI code is produced from the previous levels. This code could be either interpreted (in this case, UI rendering is ensured) or compiled (in case, various techniques such as generative programming, template-based approach, static code generation could be used.

Our methodology enables expressing and executing model transformation based on UIs viewpoints. For this purpose, the mapping model links the various models resulting from the above steps through mappings [6]:

- *Reification* is a transformation of a high-level requirement into a form that is appropriate for low-level analysis or design.
- *Abstraction* is an extraction of high-level requirement from a set of low-level requirements artifacts or from code.
- *Translation* is a transformation a UI in consequence of a context of use change. The context of use is, here, defined as a triple of the form (U, P, E) where E is an possible or actual environment considered for a software system, P is a target platform, and U is a user category.
- *Reflection* is a transformation of the artifacts of any level onto artifacts of the same level of abstraction, but different constructs or various contents.

4. Using UI Patterns with IDEALXML

To support the usage of UI patterns as defined in Fig. 3, the IdealXML software has been developed that today consists of 17,000 lines of Java code. It can exploit a knowledge base of UI patterns stored in UsiXML language [32] (www.usixml.org). This UIDL has been selected because it already covered the various models involved in the below part of Fig. 3. The upper part has therefore been equally defined so that it could be expressed in a XML format that is compliant with UsiXML. In order to illustrate how this software can support the four-step method outlined above, let us consider an example related with web design and development: the Sedan-Bouillon web site (http://www.sedan-bouillon.org/) is a web site for providing tourists

with location-aware information on the archeological site. Fig. 4 shows a screen shot where tourist guides could be ordered on-line.

Figure 4. Contact page on the Sedan-Bouillon site.

This web page is a form where the user can ask until three different catalogs related with tourist information of this French region. This request is considered as a transaction that a visitor (*participant*) establishes when he visits this website. This participant should be provided with additional information in order to receive these catalogs. And finally the user should send his request pressing send button. In order to design our application at least three models should be considered: domain, task, and abstract UI models before reaching a final UI. To do that different elements are used: class diagrams for the domain model, ConcurTaskTree notation [23] for the task model, and *Abstract Interaction Objects* (AIOs) for the abstract UI model. For doing this purpose, we can use patterns for each model. So, we can identify three classes in our diagram of classes: participant, transaction and catalogs. These classes and their relationships are gathered in a pattern recommended from [8, 22]. They are *participant-transaction* pattern and *transaction-specificItem* pattern (Fig. 5). In a similar way, a task model is specified according to the ConcurTaskTree notation [23]. Fig. 6 reproduces such a task model where different tasks related with the request filling where the user firstly selects a catalog, then provides personal information of contact and finally send his request. In Fig. 6, we can identify patterns that we can find in many task specifications, for instance, when the user selects, writes, or invokes actions, we can see similar graphical notations and propose edit pattern, invoke-validation-send pattern, form pattern or wizard pattern (Fig. 7) and these patterns can be represented using firstly CTT notation and secondly UsiXML [32]. These task patterns have been previously stored, so they can be reused here or not.

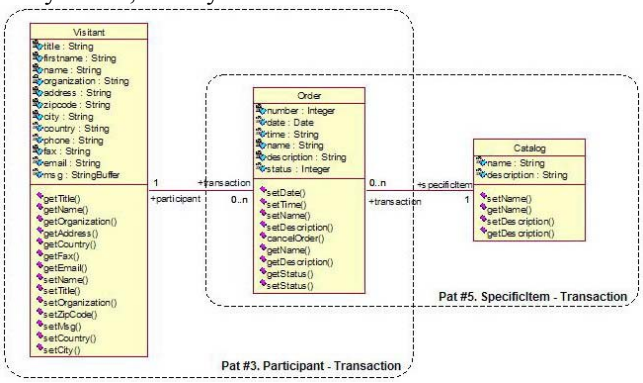


Figure 5. Domain model using patterns.

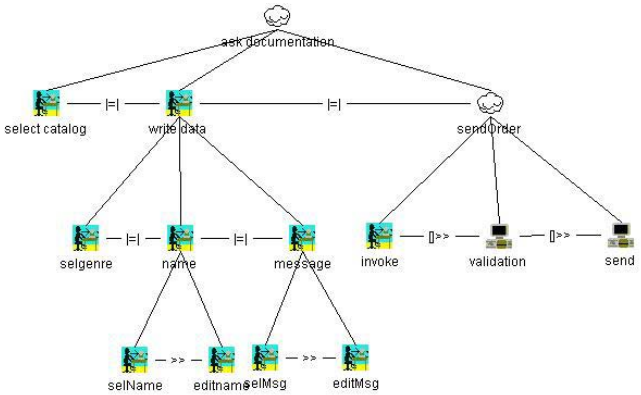


Figure 6. Task model using CTT notation

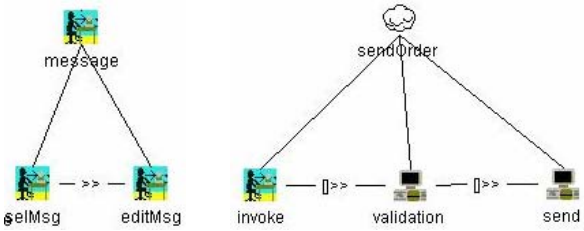


Figure 7. Examples of task patterns: edit pattern and invoke-validation-action.

After representing the task and the domain models, it is possible to link elements of these two models through the mapping model. Such mappings include: *triggers*, *observers*, *updates* (mappings between domain and task), *isReifiedBy*, *isAbstractedInto* (mappings between abstract and concrete UIs), *manipulates* (task and domain) and *isExecutedIn* (task and abstract UI). In this sense, we can identify patterns between models (*intramodel*-patterns) as the mapping model contains a series of mappings between the related models. Therefore, if we have a domain model that represents a domain pattern and a task model that represents a task pattern, it is possible in IDEALXML to enter mappings between so as to create a task+domain pattern. This reasoning is similar for all subsequent models found in the next steps.

After modeling task and domain, an AUI model is

needed that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent from any modality and computing platform. Such AIOs are composed of multiple facets, each facet describing a particular function to be assumed (*input*, *output*, *navigation* and *control*) (Fig. 8). IDEALXML provides an editor where an abstract representation can be specified using *abstract containers*, *abstract individual components* and *facets* (Fig. 8).

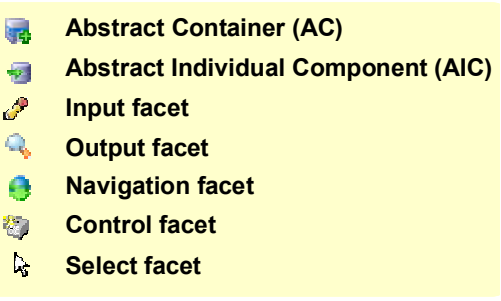


Figure 8. Stylistics for the Abstract User Interface.

Fig. 9 represents a simplified abstract UI: first a container for the request form and then several individual components were defined in order to specify catalogs and components of the form used in this example. All these specifications can be done using IDEALXML where four editors (Fig. 11) are provided in order to model tasks, domain presentation and mappings between them. For example, Fig. 10 depicts a mapping between task, domain, and abstract UI, when the designer identifies tasks where the user invokes actions, these actions can include validation of information and then the action will be executed. Methods and attributes will be invoked too when these

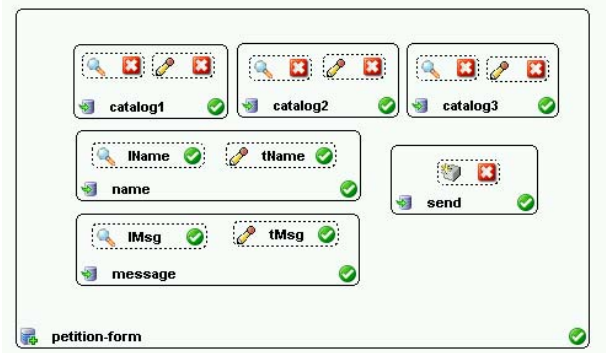


Figure 9. Abstract specification of Sedan-Bouillon form.

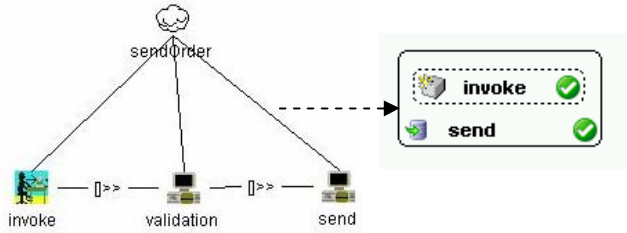
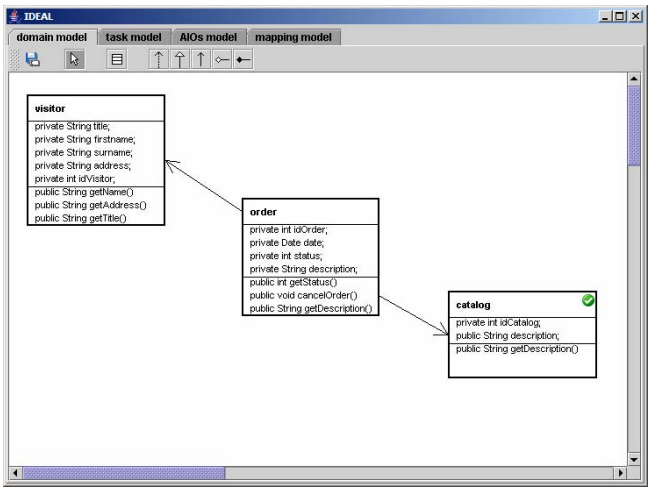
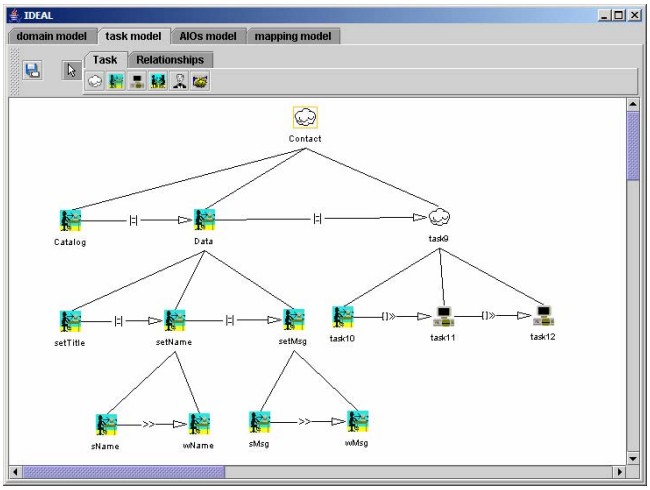


Figure 10. Abstract UI, task and model relationships.





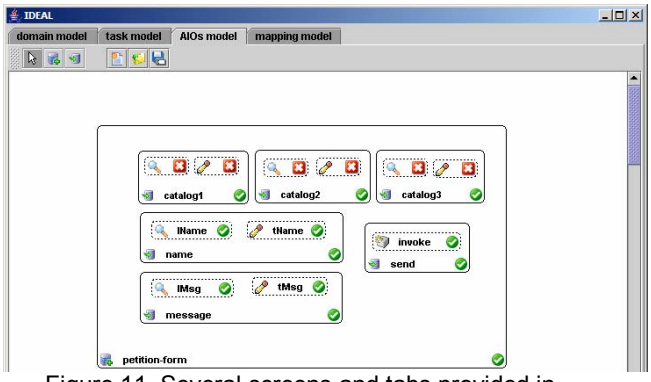


Figure 11. Several screens and tabs provided in IdealXML for the various models in the UI pattern.

5. Conclusion

In this paper, we have introduced IDEALXML, a software that provide facilities for managing UI patterns according to the rules of model-based approach as defined in MDA. With IDEALXML, it is possible to specify task, domain, and UI models in a graphical way and to automatically generate specifications in UsiXML, a XML-based language used to specify UI. Patterns can be expressed at any level (e.g., one model only) or declined at several levels (e.g., multiple models simultaneously). In addition, it is possible to link several different UI for a single task+domain depending on the context of use. In this case, the context determines the solution given in the UI pattern. The original aspect is that the patterns are generative (the UsiXML specifications initiate automated code generation) as opposed to only descriptive and contemplative.

6. References

[1] Alexander, C., *The Timeless Way of Building*, Oxford University Press, New York, 1979.

[2] Bastien, J.M.Ch. and Scapin, D.L., "Evaluating a User Interface with Ergonomic Criteria", *Int. J. of Human-Computer Interaction*, 7(2), 1995, pp. 105–121.

[3] Bayle, E., Bellamy, R. Casaday, G., Erickson, T., Fincher, S., and Grinter, B., "Putting it all Together: Towards a Pattern Language for Interaction Design", *SIGCHI Bulletin*, 30(1), 1998, pp. 17–24.

[4] Borchers, J., *A Pattern Approach to Interaction Design*, John Wiley & Sons, New York, 2001.

[5] Budinsky, F., Finnie, M., Vlissides, J., and Yu, P., Automatic Code Generation from Design Patterns, *IBM Systems Journal*, 35(2), 1996.

[6] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., "A Unifying Reference Framework for Multi-Target User Interfaces, *Interacting with Computers*, 15(3), 2003, pp. 289–308.

[7] Campos, P. and Nunes, N.J., "Towards useful and usable interaction design tools: CanonSketch", *Interacting with Computers*, 19, 2007, pp. 597–613.

[8] Coad, P., Mayfield, M., and North, D., *Object Models: Strategies, Patterns, and Applications*, Prentice Hall, New Jersey, 1997.

[9] Constantine, L., "Canonical Abstract Prototypes for Abstract Visual and Interaction Design", Proc. of DSV-IS'2003, Springer, Berlin, 2003, pp. 1–15.

[10] Erickson, T., The Interaction Design Patterns Page, http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html

[11] Fincher, S., The pattern gallery, http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html

[12] Fincher, S., Finlay, J., Greene, Sh., Jones, L., Matchen, P., Thomas, J., and Molina, P.J., "Perspectives on HCI patterns: concepts and tools", Ext. Proc. of CHI'2003, ACM Press, New York, 2003, pp. 1044–1045.

[13] Gaffar, A., Seffah, A., and Van der Poll, J., "HCI Patterns Semantics in XML: A Pragmatic Approach", Proc. of HSSE'2005, ACM Press, New York, 2005.

[14] Gamma, E., Helm; R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, 1994.

[15] Granlund, Ä., Lafrenière, D., and Carr, D.A., "A Pattern-Supported Approach to the User Interface Design Process", Proc. of HCI Int'2001.

[16] Greene, S. and Matchen, P., "Tool-based decision support for pattern assisted development", Proc. of CHI' 2003 Workshop on User Interface Patterns.

[17] Javahery, H. and Seffah, A., "A Model for Usability Pattern-Oriented Design", Proc. of Tamodia'2002, Inforec Printing House, Bucharest, 2002, pp. 104–110.

[18] Joshi, S.R., McWilliam, W.W., "Case-based reasoning approach to creating user interface components", Proc. of CHI'96, ACM Press, New York, 1006, pp. 81-82.

[19] Kolodner, J., *Case-Based Reasoning*, Morgan Kaufman, San Mateo, 1993.

[20] López-Jaquero, V., Montero, F., Molina, J.P., Fernández-Caballero, A., and González, P., "Model-Based Design of Adaptive User Interfaces through Connectors", Proc. of DSV-IS'2003, Springer, pp. 245–257.

[21] Molina, P., Belenguer, J., and Pastor, O., "Describing Just-UI Concepts Using a Task Notation", Proc. of DSV-IS'2003, Springer, Berlin, 2003.

[22] Nicola, J., Mayfield, M., and Abney, M., *Streamlined Object Modeling*, Prentice Hall, New Jersey, 2001.

[23] Paterno, F., *Model-based design and evaluation of interactive applications*, Springer, Berlin, 1999.

[24] Pribeanu, C., and Vanderdonckt, J., "A Pattern-based Approach to User Interface Development", Proc. of UAHCI'2003, Lawrence Erlbaum, pp. 1524–1528.

[25] Radeke, F., Forbrig, P., Seffah, A., Sinnig, D., "PIM Tool: Support for Pattern-Driven and Model-Based UI Development", Proc. of Tamodia'2006, Springer.

[26] Schneider, K. and Paquette, D., "Interaction Templates for Constructing User Interfaces from Task Models", Proc. of CADUI'2004, Kluwer, pp. 221–232.

[27] Seffah, A. and Forbrig, P., "Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model", Proc. of DSV-IS'2002, Vol. 2545, Springer, Berlin, 2002, pp. 118–132.

[28] Seffah, A. and Gaffar, A., *Model-based user interface engineering with design patterns*, Journal of Systems and Software, 80, 2007, pp. 1408–1422.

[29] Seffah, A., MOUDIL: Montreal Online Usability Digital Library. http://hci.cs.concordia.ca/moudil/

[30] Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., and Seffah, A., "Patterns in Model-Based Engineering", Proc. of CADUI'2004, Kluwer, pp. 195–208.

[31] Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., and Taddeo, L., *Multi-model and Multi-level Development of User Interfaces*, "Multiple User Interfaces", John Wiley, pp. 193–216.

[32] Vanderdonckt, J., "A MDA-Compliant Environment for Developing User Interfaces of Information Systems", Proc. of CAiSE'05, Springer, Berlin, pp. 16–31.

[33] van Welie, M., Patterns in Interaction Design, 2004.

[34] van Welie, M., van der Veer, G.C., and Eliens, A., "Patterns as Tools for User Interface Design", Proc. of TFWWG'2000, Springer, London, 2000, pp. 313–324.