Generative Pattern-Based Design of User Interfaces

Jean Vanderdonckt
Louvain School of Management
Université catholique de Louvain
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
+32 10 47.85.25
jean.vanderdonckt@uclouvain.be

Francisco Montero Simarro
Laboratory on User Interaction & Software Engineering
University of Castilla-La Mancha
02071 Albacete, Spain
+34 967 599200 Ext.: 2468
fmontero@info-ab.uclm.es

ABSTRACT

This paper suggests a method for developing graphical user interfaces based on generative patterns. A generative pattern contains portions of previously designed user interfaces are expressed through models that are either partially or totally instantiated. These portions could be identified and re-applied to a new design case study by generating code by instantiating the specifications contained in the models. The method involves typical models found in user interface development life cycle such as task, domain, abstract user interface, concrete user interface, final user interface, context model, and mappings between them. Any model could virtually be the source of a pattern and could be described, searched, matched, retrieved, and assembled together so as to create a new graphical user interface. For this purpose, a software has been developed that manages generative patterns by combining an existing user interface description language (UsiXML – user interface extensible markup language) with concepts addressing problems raised by pattern description and matching in a pattern-based language (PLML – Pattern Language Markup Language, a language was introduced to uniformly represent user interface patterns). Once instantiated from the generative patterns, the models give rise to a model-driven engineering based on model-to-model transformation and model-to-code compilation.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *evolutionary prototyping, user interfaces*. D.2.11 [**Software Engineering**]: Software Architectures – *Patterns (e.g., client/server, pipeline,blackboard)*. D.3.3 [**Programming Language**]: Language Constructs and Features – *Patterns*. H.5.2 **[Information Interfaces and Presentation (e.g., HCI)**]: User interfaces – *Graphical user interfaces (GUI), User interface management systems (UIMS)*.

General Terms

Algorithms, Design, Human Factors, Standardization, Languages.

Keywords

Descriptive pattern, Generative pattern, Model-Driven Engineering, User interface pattern.

PEICS'10, June 20, 2010, Berlin, Germany

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 [15,34] 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, interactive 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]. Usability guidelines were criticized for not mentioning explicitly the context in which they are applicable [3]. The CHI'2003 workshop on UI Patterns [12] observed that many different, probably inconsistent, sources of UI design patterns exist today [10,11,29,34], 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, re-use, and linking between various collections. PLML is a natural language-based way for writing patterns, thus potentially suffering from intrinsic problems like ambiguity, inconsistency, PLML does not escape from these problems. Therefore, this language is more frequently used for describing UI patterns than for supporting pattern-based UI design process [15] that is effectively and efficiently supported by software. These problems include, but are not limited to (Fig. 1):

- **Lack of expressivity**. Several PLML tags express various pattern aspects that were believed of sufficient general interest, but some are missing. For instance, a tag describes the forces of a pattern, but nothing describes the counter-forces.
- **Flat definition**. PLML is defined in a Document Type Definition (DTD) in a flat structure that does not easily support structured pattern-matching and searching.
- **Lack of separation of concerns**. PLML mixes the expression of several concepts together, thus reducing 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. It is therefore hard to exploit this context description to identify potentially similar contexts of use in which the same pattern could become applicable.
- **Lack of structure**. Several tags are 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 discouraging a structured use of the tags by a software (which is a shortcoming for large and efficient use).

```
<!ELEMENT pattern (name?, alias*, illustration?, problem?, context?,
                   forces?, solution?, synopsis?, diagram?, evidence?, confi-
                   dence?, literature?, implementation?, related-patterns?, pat-
                   tern-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 generative 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 *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. BACKGROUND

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

Many references exist 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* [34]. 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. 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).

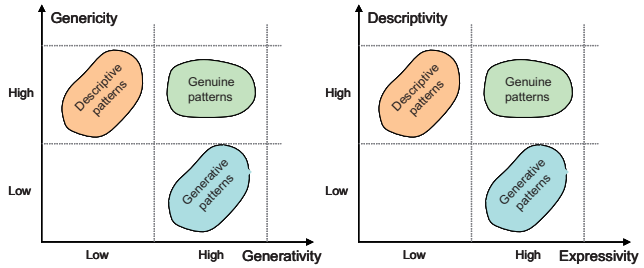


Figure 2. Classification of patterns compilations according to the four properties.

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 document generative patterns since they show the characteristics of good UIs (e.g., they convey information about usability [35]) that are appropriate in their context of use and how to develop them. This does not mean that descriptivity should be left out.

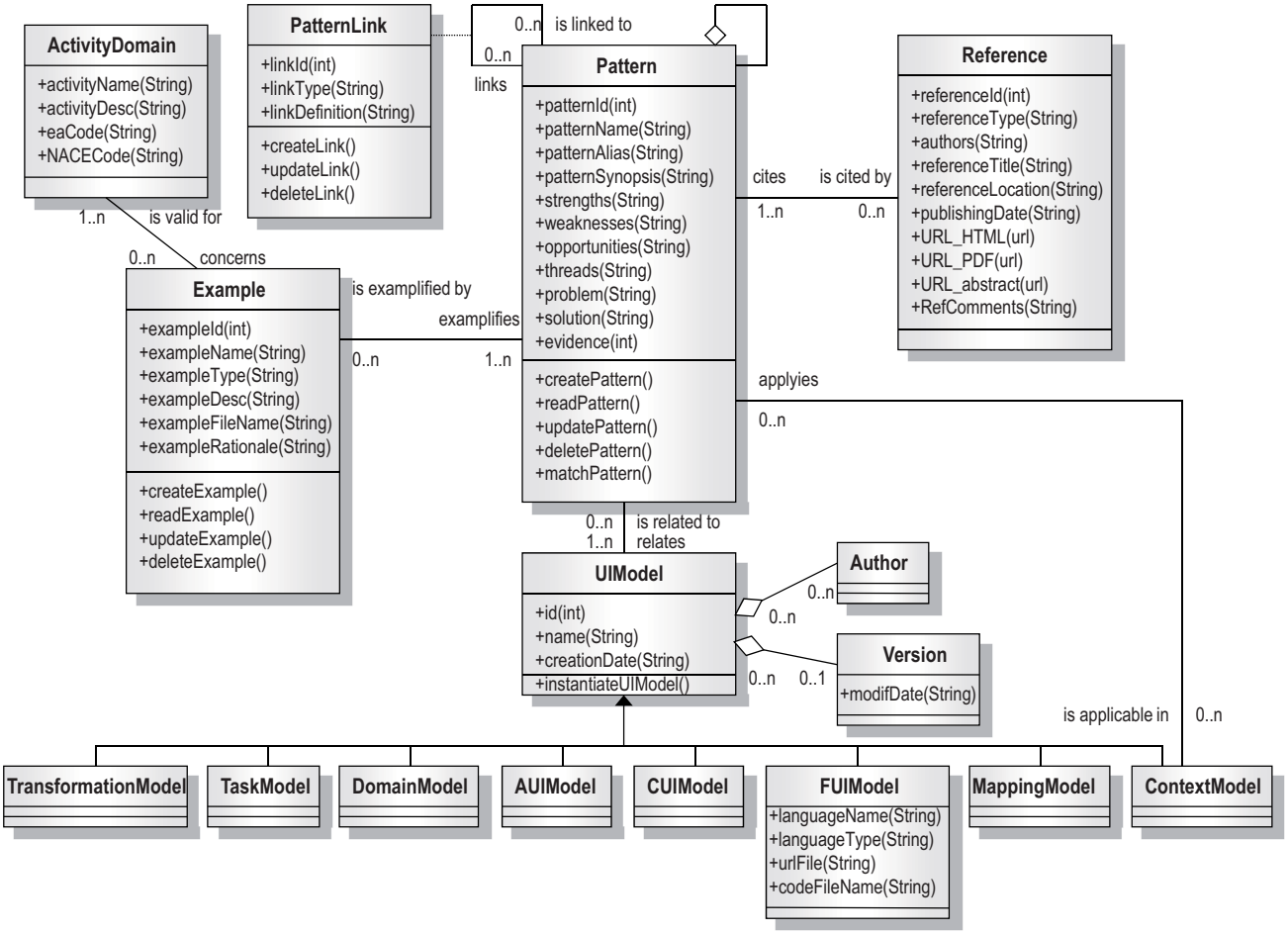


Figure 3. UML Class diagram of a UI pattern extended from PLML [12].

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. Genuine patterns are expected that maximize the four properties of purely descriptive and generative patterns. Descriptive patterns are usually estimated of high genericity and descriptivity, but low in generativity and expressivity (Fig. 2). Generative patterns are in an inverse situation: they are high in generativity and expressivity, but low in genericity and descriptivity. By combining the qualities of both families into genuine patterns, it is expected to reach a high level for the four properties simultaneously (Fig. 2).

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 [34] 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.

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 UI representation so that it is both descriptive and generative.

3. CONCEPTUAL MODEL OF 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. Based on specifications reproduced in Fig. 1, PLML has been expanded 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. 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 counter-balanced 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 body could be considered as stronger than a pattern provided by an individual person.

Examples showing the application of a pattern so as to facilitate its interpretation and its application are fundamental [35]. 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 [6,31,32,33]:

1. *Task and domain modeling* (Platform 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 implementation.
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 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 are ordered on-line.

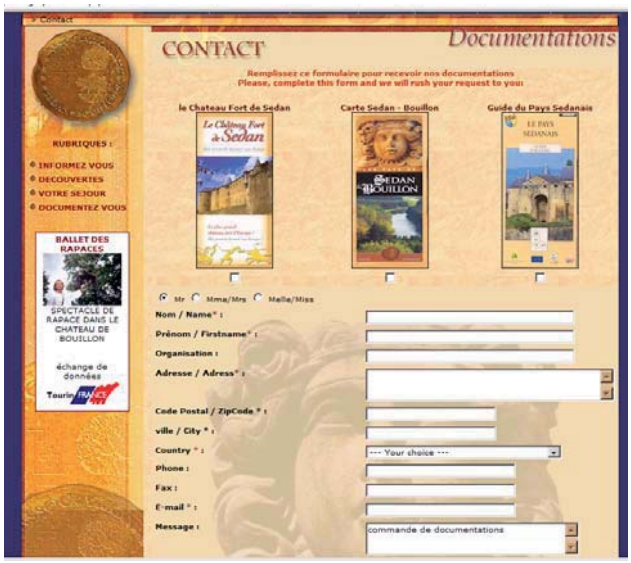


Figure 4. Contact page on the Sedan-Bouillon site.

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. Different elements are used for this purpose: class diagrams for the domain model, ConcurTaskTree notation [23] for the task model, and *Abstract Interaction Objects* (AIOs) for the abstract UI model. 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 structured according two patterns [8]: *participant-transaction* pattern (Fig. 5) and *transaction-specificItem* pattern (Fig. 6). A participant-transaction pattern establishes a relationship between a participant (i.e. agent, applicant, buyer, cashier, customer, dealer, delegate, distributor, employee, investor, manufacturer, member, owner, professional, prospect, recipient, retailer, sales clerk, shipper, student, subscriber, supervisor, supplier, teacher, worker) that is able to perform transactions (i.e. agreement, assignment, contract, delivery, deposit, inquiry, order, payment, problem, report, purchase, refund, registration, rental, sale, shipment, subscription, withdrawal) [8]. Similarly, a task model is specified according to the ConcurTaskTree notation [23]. Fig. 7 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. Fig. 8 reproduces patterns for 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. 9). These patterns are represented using CTT notation and stored in UsiXML [32], a User Interface Description Language.

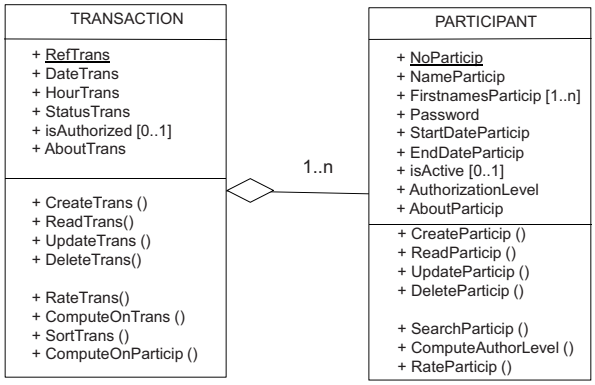


Figure 5. Participant-transaction pattern.

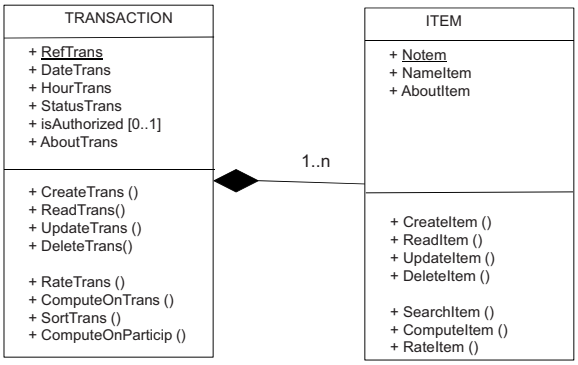


Figure 6. Transaction-specificItem pattern.

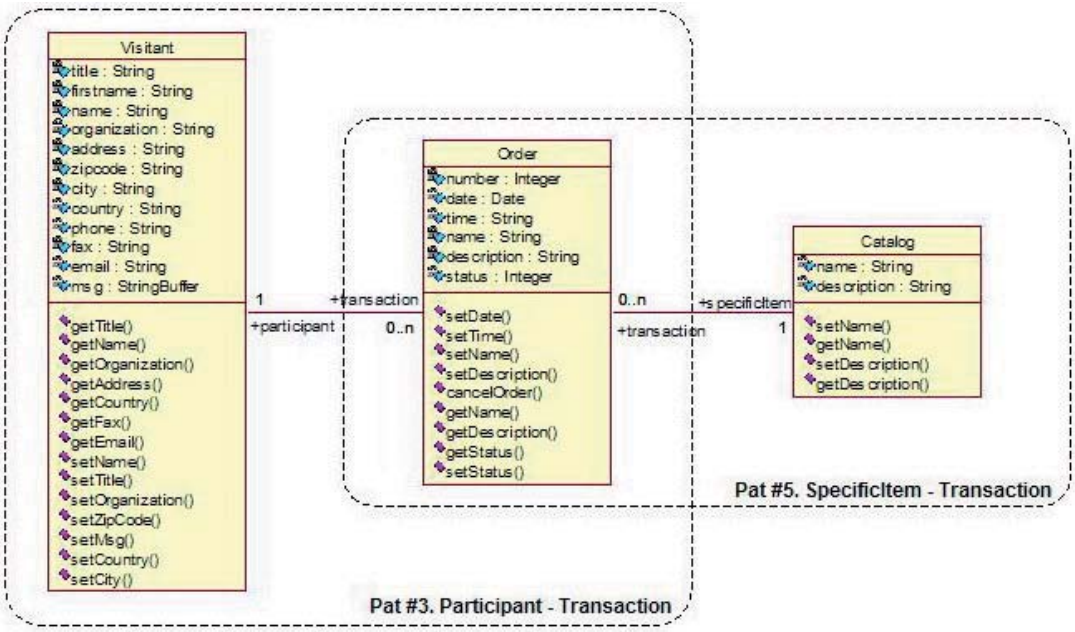


Figure 7. Domain model using patterns.

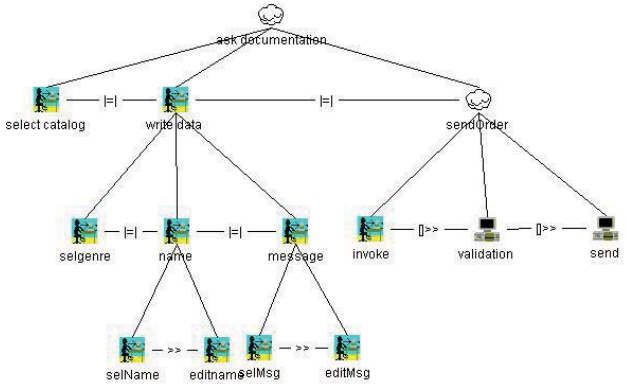


Figure 8. Task model using CTT notation.

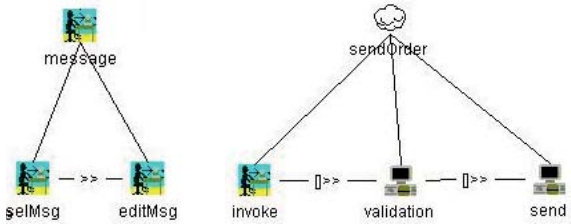


Figure 9. Examples of task patterns: edit pattern and invoke-validation-action.

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. 10). IDEALXML provides an editor where an abstract representation can be specified using *abstract containers*, *abstract individual components* and *facets* (Fig. 10). Fig. 11 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. 12) are provided in order to model tasks, domain presentation and mappings between them. For example, Fig. 13 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 actions are done.

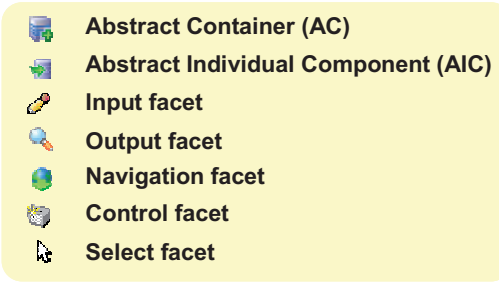


Figure 10. Stylistics for the Abstract User Interface.

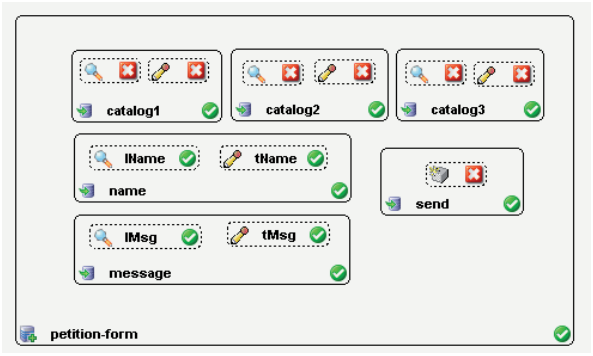


Figure 11. Abstract specification of Sedan-Bouillon form.

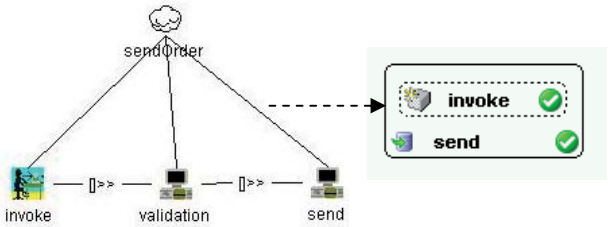
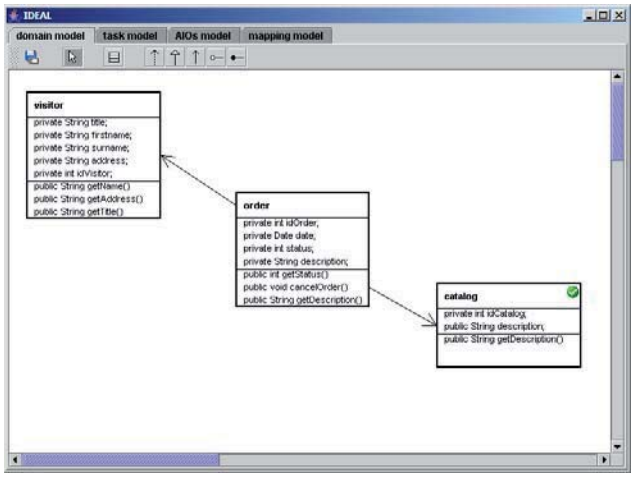
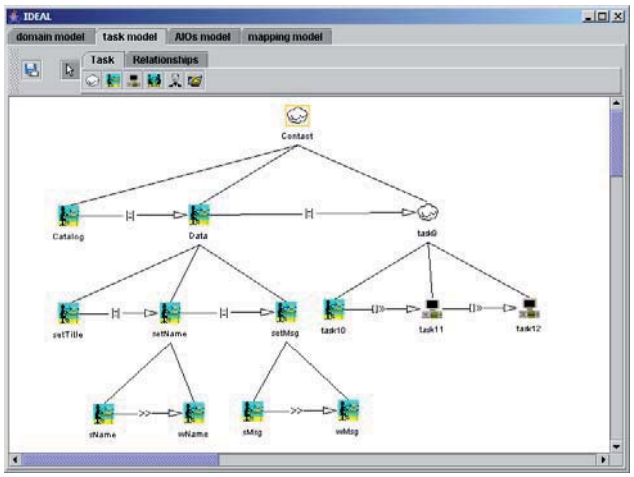


Figure 12. Abstract UI, task and model relationships.





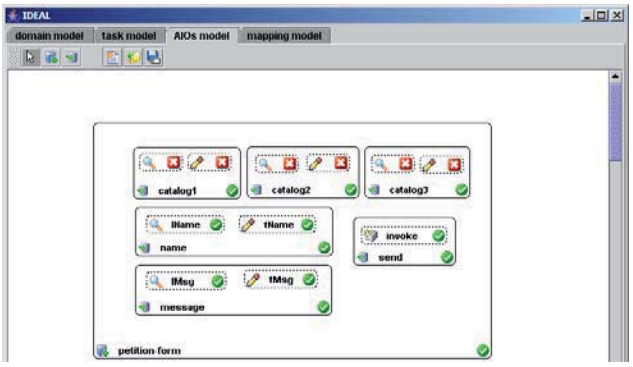


Figure 13. Several screens and tabs provided in IDEALXML for the various models in the UI pattern.

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. ACKNOWLEDGMENTS

We acknowledge the support of the ITEA2 Call3 Project UsiXML (User Interface eXtensible Markup Language) under reference 2008026 and the Région Wallonne from Belgium.

7. REFERENCES

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

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

[5] Budinsky, F., Finnie, M., Vlissides, J., and Yu, P. Automatic Code Generation from Design Patterns. *IBM Systems Journal* 35, 2 (1996), pp. 151–171. Accessible at http://www.research.ibm.com/designpatterns/pubs/codegen.pdf

[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, 5-6 (2007), pp. 597–613.

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

[9] Constantine, L. 2003. Canonical Abstract Prototypes for Abstract Visual and Interaction Design. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). J. Jorge, N.J. Nunes, J. Cunha (Eds.). Lecture Notes in Computer Science, Vol. 2844. Springer, Berlin, pp. 1–15.

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

[11] Fincher, S. The pattern gallery. Accessible at http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html

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

[13] Gaffar, A., Seffah, A., and Van der Poll, J. 2005. HCI Patterns Semantics in XML: A Pragmatic Approach. In *Proc. of the 2005 workshop on Human and social factors of software engineering HSSE'2005* (St. Louis, May 16, 2005). ACM Press, New York, pp. 1–7.

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

[15] Granlund, Å., Lafrenière, D., and Carr, D.A. 2001. A Pattern-Supported Approach to the User Interface Design Process. In *Proc. of 9th Int. Conf. on Human-Computer Interaction HCI International'2001* (New Orleans, 5-10 August 2001). M.J. Smith, G. Salvendy, D. Harris, R.J. Koubek (Eds.). Vol. 1, Lawrence Erlbaum Associates, Mahwah.

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

[17] Javahery, H. and Seffah, A. 2002. A Model for Usability Pattern-Oriented Design. In *Proc. of 1st International Workshop on Task models and Diagrams for user interface design TAMODIA'2002* (Bucarest, 18-19 July 2002). Pribeanu, C., Vanderdonckt, J. (Eds.). Academy of Economic Studies of Bucharest, Economic Informatics Department, INFOREC Printing House, Bucarest, pp. 104–110.

[18] Joshi, S.R., McWilliam, W.W. 1996. Case-based reasoning approach to creating user interface components. In *Proc. of CHI'96*. ACM Press, New York, pp. 81-82.

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

[20] López-Jaquero, V., Montero, F., Molina, J.P., Fernández-Caballero, A., and González, P. 2003. Model-Based Design of Adaptive User Interfaces through Connectors. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin, pp. 245–257.

[21] Molina, P., Belenguer, J., and Pastor, O. 2003. Describing Just-UI Concepts Using a Task Notation. In *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003* (Madeira, 4-6 June 2003). Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin, pp. 361-371.

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

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

[24] Pribeanu, C., and Vanderdonckt, J. 2003. A Pattern-based Approach to User Interface Development. In *Proc. of 2nd Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2003* (Creete, 22-27 June 2003). Vol. 4, C. Stephanidis (Ed.). Lawrence Erlbaum Associates, Mahwah, pp. 1524–1528.

[25] Radeke, F., Forbrig, P., Seffah, A., and Sinnig, D. 2006. PIM Tool: Support for Pattern-Driven and Model-Based UI Development. In *Proc. of 5th Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006* (Hasselt, 23-24 October 2006). K. Coninx, K. Luyten, K. Schneider (Eds.). Lecture Notes in Computer Science, Vol. 4385. Springer-Verlag, Berlin (2007), pp. 82–96.

[26] Schneider, K. and Paquette, D. 2004. Interaction Templates for Constructing User Interfaces from Task Models. In *Proc. of 5th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2004* (Funchal, 14-16 January 2004). Kluwer Academics Pub., Dordrecht, pp. 221–232.

[27] Seffah, A. and Forbrig, P. 2002. Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model. In *Proc. of 9th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002* (Rostock, 12-14 June 2002). Lecture Notes in Computer Science, Vol. 2545, Springer-Verlag, Berlin, 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. 2004. Patterns in Model-Based Engineering. In *Proc. of 5th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2004* (Funchal, 14-16 January 2004). Kluwer Academics Pub., Dordrecht, 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*. In "Multiple User Interfaces", A. Seffah, H. Javahery (Eds.), John Wiley, pp. 193–216.

[32] Vanderdonckt, J. 2005. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05* (Porto, 13-17 June 2005). O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31. DOI= http://dx.doi.org/10.1007/11431855_2.

[33] Vanderdonckt, J., Limbourg, Q., Florins, M. Deriving the Navigational Structure of a User Interface. In *Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003). IOS Press, Amsterdam, 2003, pp. 455-462.

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

[35] van Welie, M., van der Veer, G.C., and Eliens, A. 2000. Patterns as Tools for User Interface Design. In *Proc. of the 1st Int. Workshop on Tools for Working with Guidelines TFWWG'2000 Group (Biarritz, 7-8 October 2000).* J. Vanderdonckt, Ch. Farenc (Eds.). Springer-Verlag, London, pp. 313–324.