IDEALXML: an Experience-Based Environment for User Interface Design

Francisco Montero, María Lozano, Pascual González

Grupo de Investigación LoUISE
Universidad de Castilla-La Mancha
{fmontero, mlozano, pgonzalez}@info-ab.ulcm.es

Abstract. Coad said that *a pattern is a template of interacting objects, one that may be used again and again by analogy* [9]. Since 1997, HCI community has been working in the development of user interface patterns and pattern languages. Nevertheless, HCI community lacks a unified language for expressing patterns like software engineering has in UML yet, this lack of a unified language poses an interesting challenge for HCI research community to bring software engineering experiences in pattern-based design into user interface development. In this paper an environment, IDEALXML, is introduced to take another step forward towards pattern-based user interfaces design. These patterns are gathered by using textual descriptions and diagrams. In the proposed tool, some of the most salient for notations, available to HCI design have been integrated.

1 Introduction

Experience is the accumulation of knowledge or skills that result from direct participation in events or activities. Developers have a strong tendency towards reusing designs that worked well for them in the past. Unfortunately, this design reuse is usually limited to personal experience, and there is usually no or few shaning of knowledge among developers. Even though many guidelines have been proposed, few of them have been validated. Furthermore, the user interfaces (UI) of computed applications are difficult to build for novice developers, because in their creation developers must deal with many low level-details.

In the last several years, Model-Based User Interface Environments (MB-UIDEs) [11, 35, 38] refers to a paradigm which uses an explicit, largely declarative representation capturing application semantics and other knowledge needed to specify the appearance and behavior of an interactive system within this paradigm.

The user interface developer instead of writing a large procedural program defines a model of facts, which controls behavior of reusable code, and a much smaller procedural program. The goal of the model-based UI design is to identify reusable components of a UI and to capture more knowledge in the model, while reducing the amount of new (procedural) code that has to be written for each new application. But MB-UID methods and tools still have unsolved problems [11, 28, 37]. Some of the most unsolved important issues in MB-UID methods and tools are: (1) the lack of a

common representation language such as UML [5, 32] in software and architecture design. Currently, there is no standard in IU development. In software design discipline, at least there is an agreement in the notation to be used: UML. However, for user interfaces, we are still far away of having such standard. And, (2) We lack for commercial tools supporting the from current IU design methods. Few tools are available on the market, therefore, it is hard to attract practitioners in the field of UI development to model-based practices. There are also methods without any tool support at all: making much harder for practitioners to use such methods. The quality of traditionally design UIs depends in a strong factor on the experience of the designers and their skills in the platform and development tools they use.

In this paper we introduce *Interface Development Environment for AppLications specified in usiXML* (IDEALXML) as a solution to integrate experience (using patterns) into model-based user interface development, using different notations related with UI development and XML. The main objective behind IDEALXML is to provide a single editor where designers can use and transfer design knowledge that has emerged from experience. Designers read and use patterns so as to profit from their own experience or from others. In this tool designers read and use patterns to take advantage from the experience gathered.

This paper is organised into four further sections. Section 2 develops the concept of experience in user interface design. Section 3 refers to patterns as tools for user interface design, theirs features, notations (i. e. PLML) and the possibilities of improvements. Section 4 introduces IDEALXML by means of a study case. Finally, section 5 presents some conclusions.

2 Experience in user interface design

Many guidelines have been published in document sources throughout the literature related with user interface development. Guidelines have two main origins: psychological theory and practical experience. A wide range of HCI guidelines are available, some very general and some specific to particular systems. These guidelines fall into several categories [3, 45]: *principles*, *design rules* and *standards*.

The best user interface design guidelines are high level and widely applicable directing principles, for example: *know the user population*, *engineer for errors*, *maintain consistency and clarity*, etc.

People sometimes confuse principles with design rules. Design rules are instructions which can be obeyed with minimal interpretation by the user. For example, *specifying that all dates in a system should be displayed or entered in a concrete way: mm-dd-yyyy* is a design rule. Principles and design rules are related. Principles must be interpreted and translated into a strategy for producing clear-cut design rules.

Specific guidelines are often expressed as standards. Standards are developed and promoted by a wide range of organizations for many different reasons, so we have:

– International standards bodies such as International Organization for Standardization (ISO), American National Standards Institute (ANSI) and

Worldwide Web Consortium (W3C). Examples of ISO standards are ISO 9241 – Ergonomics for Office work with Virtual Display Terminals [19] and ISO 13407 – Human Centred Design for Interactive Systems.

- Industry standards are published by major players in the software industry, for example: The User Interface Guidelines for Microsoft Windows [25], Apple Macintosh Human Interface Guidelines [24] and IBM Web Design Guidelines [18].
- De Facto standards are standards largely because they have been widely adopted. QWERTY keyboards are a good example of this.

Despite the fact that guidelines have been proved useful, they still suffer from a series of shortcomings that impede their use and significantly reduce their scope [3, 31, 45]. First, Fig. 1 depicts that guidelines found in the three types of sources range from principles to standards have an abstract interpretation.

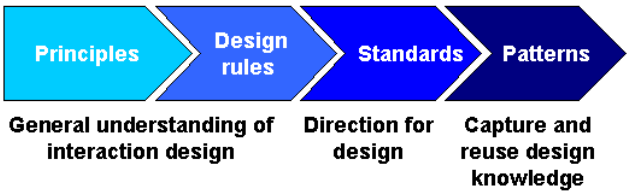

Fig. 1. Guidelines and patterns in interaction design

Second, as a consequence of the previous observation, general guidelines cannot be applied per se, thus requiring some concrete interpretation for the intended context of use. General guidelines are difficult to interpret when and how they need to be applied at design time or evaluated at execution time.

Third, the jargon or specific vocabularies used in the sources coming from various disciplines (e.g., cognitive modeling, psychology, human factors, ethnography) may also prevent designers from easily understanding and applying them correctly. Once again, extensive relevant experience may be needed to avoid incorrect generalization or specialization of guidelines that will invalidate their results.

Fourth, guidelines can be classified according to the linguistic level of interaction model [30], i.e. goal, pragmatic, semantic, syntactic, lexical, alphabetical, and physical. Guidelines located at the lower levels of this layered model tend to be easier to interpret and apply than those located at the higher levels.

Fifth, the workload involved by considering a single guideline depends on several guideline properties namely, but not limited to: its linguistic level, the quality of its statement, and their scope.

Some of these shortcomings can be overcome by using patterns. Patterns provide an effective shorthand for communicating complex concepts effectively between designers. They can be used to record and encourage the reuse of best practices and they capture the essential parts of a design in a compact form e. g. for documentation of existing models. The following section more details are given on how patterns achieve this.

3 Patterns

A pattern is a form of design representation formulated by Christopher Alexander in *A Pattern Language* [2] for use in architecture. *A Pattern Language* espouses a design approach that focuses on the interactions between the physical form of buildings and the way in which that form inhibits or facilitates personal and social behaviors. Each pattern follows a prescribed form that is based on evidence for, and examples of, the use of the pattern, together with instructions for how to achieve its effect. Various domains have subsequently adopted and adapted the notion, notably *design patterns* in software [16].

Christopher Alexander said *each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice* [2].

In Software Engineering a design pattern is *a systematic way of naming, explaining, and evaluating an important and recurring design in object-oriented systems* [16]. Patterns have structure in their written representation. The essential properties are at least the description of context, problem, forces, and of course the solution, whereas the solution is, typically, regarded to be the most valuable part. Design patterns are represented as relationships between classes and objects with defined responsibilities that act in concert to carry out the solution. To illustrate a design pattern [16], consider the `Adapter` pattern, one of the original 23 patterns described in *Design Patterns*. `Adapter` provides a solution to the scenario in which a client and server need to interact with each other, but cannot because their interfaces are incompatible. To implement an `Adapter`, you create a custom class that honors the interface provided by the server and defines the server operations in the terms the client expects (Fig. 2). This is a much better solution than altering the client to match the interface of the server.

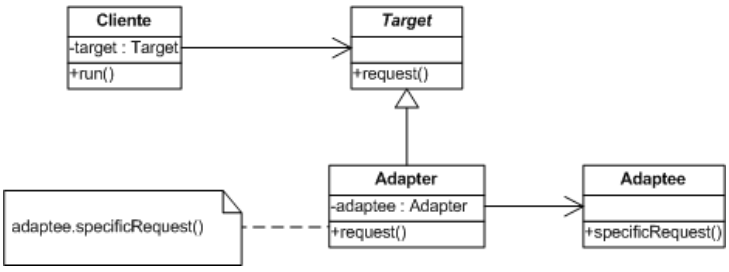

Fig. 2. Structure of Adapter design pattern

Software Engineering has a *lingua franca* or *common ground*, the Unified Modeling Language (UML) [5, 32]. UML is OMG's most-used specification, and the way the world models not only application structure, behavior, and architecture, but also business process and data structure. Several UML diagrams are used to illustrate important ideas in design patterns: a *class diagram* depicts classes, their structure, and the static relationships between them, an *object diagram* depicts a particular object

structure at run-time and an *interaction diagram* shows the flow of requests between objects.

Graphical notations, while important and useful, aren't sufficient. These notations simple capture the end product of the design process as relationships between classes and objects. To reuse the design, we must also record the decisions, alternatives and trade-offs that led to it. Concrete examples are important too, because they help us see the design in action.

```
<!ELEMENT pattern (name?, alias*, illustration?, problem?,
                context?, forces?, solution?, synopsis?, diagram?,
                evidence?, confidence?, literature?, implementation?,
                related-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?, revision-number?,
            creation-date?, last-modified?, credits?)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT creation-date (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT revision-number (#PCDATA)>
<!ELEMENT last-modified (#PCDATA)>
```

Fig. 3. XML DTD uses to describe a HCI pattern [34]

Since 1997 the HCI community has been working to develop UI and HCI patterns and pattern languages [4, 12, 14]. In HCI we have nothing similar to UML, but a first approach for HCI patterns representation was introduced at CHI 2003 workshop on "HCI Patterns: Concepts and Tools", called *Pattern Language Markup Language* (PLML) specification [15, 34]. PLML wants to bring order to the many (inconsistent) forms pattern authors have used. PLML suggests a way in which patterns and pattern languages from various authors could refer to patterns in other collections, could identify common elements across collections, ways in which patterns from disparate authors could be combined into specific, thematic collections –perhaps even combined into larger meta-collections [6, 15, 34, 36]. The discussions of what might be included in such a specification were driven by dual concerns of what we considered to be important in the domain, and the variety of forms that had already

been instantiated by various pattern authors. A pattern template was described by means of a XML DTD which is shown below [34]:

In Fig. 4 our conceptual model of an user interface pattern is shown. This figure depicts different classes (`Pattern`, `Reference`, `PatternLink`, `Example`, `Model`, etc.), their structure and the static relationships between them. `Model` class will be of special relevance in our purpose.

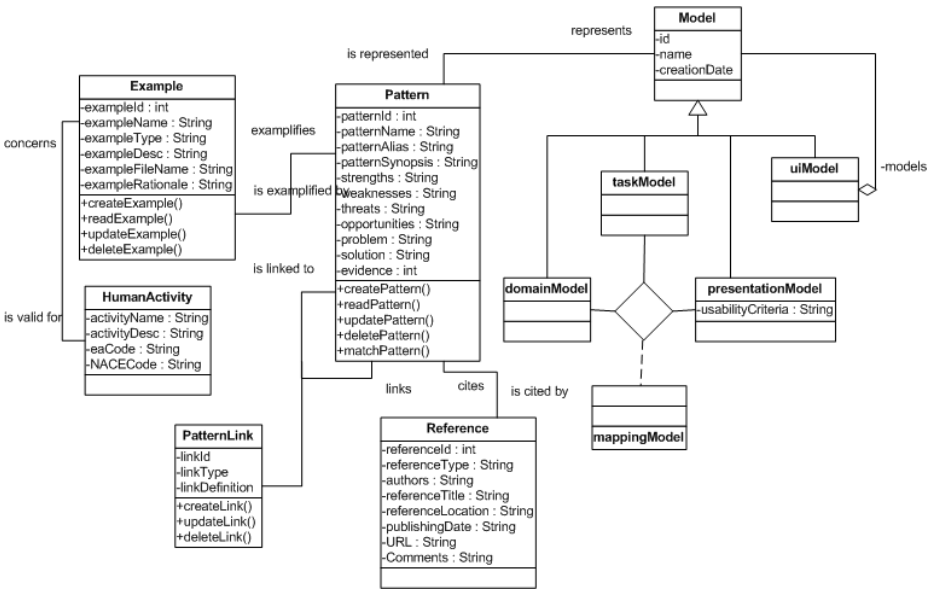

Fig. 4. Conceptual model of an user interface pattern

This diagram was inspired from PLML, but additional elements were included. We used `strengths`, `weaknesses`, `threats` and `opportunities` to consider the `forces` in the environment where a pattern will be used. They are positive and negative conditions. `Strengths` and `weaknesses` are internal to the user interface development process. `Opportunities` and `threats` originate from outside of UI design process. Pre-defined link types were included in PLML, they are useful to reflect the common ways collections are currently structured (is-a, is-contained-by and contains), but others relations can be identified (e.g., [36]) and can be included as objects from `PatternLink` class.

In Fig. 3, an element called `<diagram>` is used to communicate to the user of the pattern—the designer—details that are more readily expressed (and apprehensible) in schematic form. Sometimes this is a free-hand sketch; sometimes it is a more formal representation using different notations related with user interface development and sometimes this field is not included when a pattern is written. This a big problem when the designer is novice because he/she has not enough experience to use a pattern and patterns will not be useful to him/her. In this moment, *Model-Based User Interface Development Environments* are tools that are aimed at providing the designer with better methods of constructing user interfaces. They utilize the use of abstract models (domain, task, presentation, user, mapping, etc.) to automatically

generate the interfaces. In our opinion, a pattern is a model although some models are not patterns, and an *user interface pattern is an abstraction of a doublet, triplet, or other small grouping of elements that is likely to be helpful again and again in user interface development* (inspired from [10]).

A pattern design, Composite [16], was used to represent an associated set of models related with an user interface development. Composite pattern allows to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. These models are included in the description of a pattern in a section called `<diagram>`.

Models can be represented by using different notations, for instance, in object-oriented programming using class, object and interaction diagrams, all of them are UML diagrams. In Human-Computer Interaction (HCI) there is no standard for UI development but we have many notations with a similar purpose, for example, class diagrams are used to represent domain models, ConcurTaskTrees [39] is a notation for task model specifications which was developed by Paternò [33] to overcome limitations of notations previously used to design interactive applications and many languages are available that can describe any user interface in a manner that is device-independent and many user interface description languages are available (i.e. UIML[42], XUL[48], XAML[23], XIML[13], UsiXML[43], useML[41], AUIML[1], XAUI[47], etc.).

One of these languages, USer Interface eXtensible Markup Language (UsiXML) [22] is a proposed common representation for interaction data. UsiXML fulfills the requirements that are essential for a language of its type: (1) it supports design, operation, organization, and evaluation functions, (2) it is able to relate the abstract and concrete data elements of an interface, and (3) it enables knowledge-based systems to exploit the captured data. UsiXML [22] can be used as a *common ground* [40] or *lingua franca* [12] to store diagrams related with user interface patterns.

Finally, Alexander [2] used asterisks (*) to mark the significance of the pattern, two asterisks marking a *true invariant*, one marking a pattern which has made progress towards identifying such an invariant, but which needs further work, and no asterisks indicating confidence that an invariant has not been established, and that variations are to be expected. This information is gathered in an element called `evidence` in the conceptual model with a similar meaning,

4 IDEALXML

IDEALXML is a *work-in-progress* environment where designers can edit and modify experience-associated models. These models, patterns in some occasions, are related with interactive systems development. IDEALXML manipulates a pattern repository, that was developed using Borland® JDataStore™ [7], of patterns organized following a hierarchical structure. At the top, this structure has different models related with a MB-UIDE: domain, task, presentation and mapping, context and user models shall be done using experience of [20] in a future work. Patterns from several sources were included in this repository, i.e. [9, 29] provided domain model-related patterns, [33]

offered task model-related patterns and [27, 40, 44, 46] introduced several patterns related with presentation model. These patterns were the initial base of knowledge.

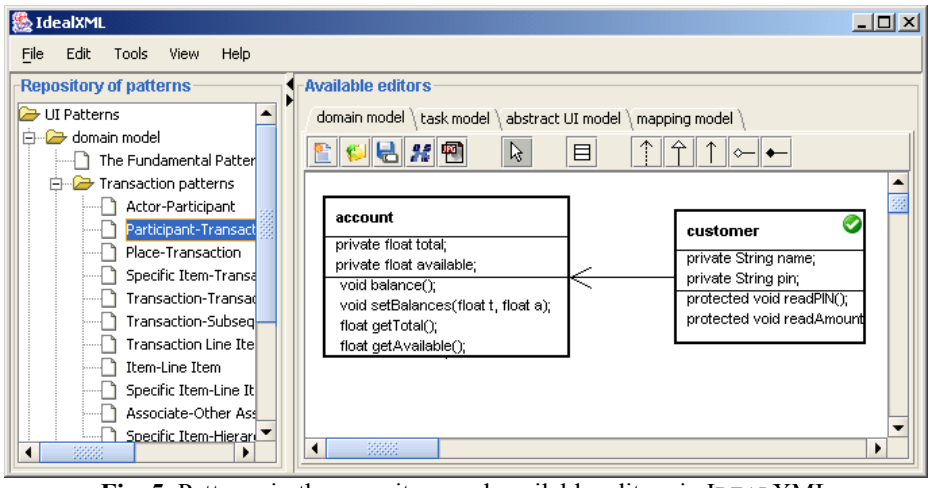

Fig. 5. Patterns in the repository and available editors in IDEALXML

Additional decomposition was done in several levels, i. e. [9] presents his domain patterns organized into various pattern families (*The fundamental pattern, transaction patterns, aggregate patterns, plan patterns* and *interaction patterns*).

We use usability criteria [3] in order to organize patterns related with presentation models [26, 40, 44, 46], because they are useful for novice designers. [8] presented two studies that they conducted to determine the effect of a short training program in applying usability criteria and recommendations on the evaluation and improvement of a web site, which contained usability problems. They suggest that web site designers, who have no university training in ergonomics, can use usability criteria (*guidance, workload, explicit control, adaptability, error management, consistency, significance of codes* and *compatibility*). Molina [26] introduced his conceptual patterns organized into various levels (*Hierarchical Action Tree, IU Service, UI Population, UI Instance* and *UI Master/Detail*). An tree structure is used to show gathered patterns (see Fig. 5). Designers can edit pattern features and theirs models and use them in IDEALXML.

This database gathered patterns are used again and again when user interfaces and interactive systems are developed. IDEALXML is an MB-UIDE and designers can, using several graphical notations, specify domain models, task models, abstract presentation models and mapping models between them. Some of these models are stored in the repository and new ones can be added. In the next sections, we will see an example of using experience with IDEALXML. In this example, we will show parts of a moderate size problem: the simulation of an Automated Teller Machine (ATM).

In ATM, a session is started whenever a customer inserts an ATM card into the card reader slot of the machine, then the customer is asked to enter his/her PIN, and he/she is then allowed to perform one or more transactions. If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being

retained in the machine. A customer must be able to make a cash withdrawal from any suitable account linked to the card.

4.1 Domain model editor

The domain model specifies the information about an application that is independent of how the objects are displayed, and how the operations are invoked. It is usually a hierarchy of classes. Patterns on domain model can be found in [9, 29].

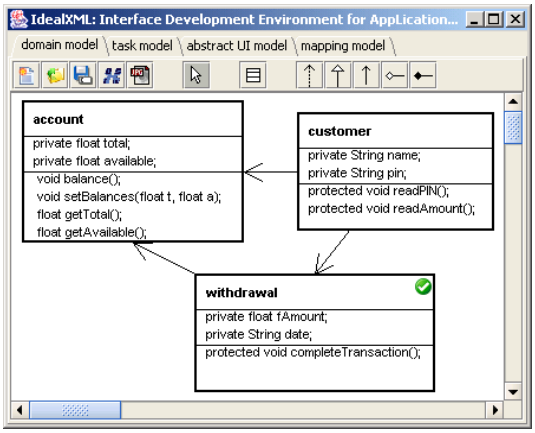

Fig. 6. Domain model is edited using IDEALXML

In [29], for instance, object modeling techniques are used for describing the business processes in representations that can be more easily mapped into software. Consider the simple business process of withdrawing money from an account. The object-oriented version of withdrawing money from an account involves objects for the account (thing), the customer (person), and the withdrawal (event) (See Fig. 6). Relationships among objects of those classes are patterns and they are gathered in [9, 29], there we can find *Collection-Worker* pattern, *Participant-Transaction* pattern, and *Transaction-SpecificItem*. We can have stored in a repository these patterns and reuse them semi-automatically (this experience need to be adapted to a concrete problem, in our example, `Customer` is a *Participant*, `Withdrawal` is a *Transaction* and an `Account` is a *SpecificItem* and a *Worker*).

All these patterns are available in a repository in a section called *domain patterns*. For instance, we can find in our repository *Collection-Worker* pattern, it is useful to represent relationships among a `customer` and his/her `accounts`. We can drag and drop patterns, from repository to IDEALXML environment, and modify these patterns using IDEALXML.

4.2 Task model editor

The task model describes the tasks that the user can perform with a system including sub-tasks, their goals, and the procedures used to achieve the goals. IDEALXML environment provides a CTT editor, similar to CTTE [39]. An essential use case editor is a work in progress, essential use cases, sometimes called a business use case, are a simplified, abstract, generalized use case that captures the intentions of a user in a technology and implementation independent manner. We think that an essential use case is an abstract generalization and they can be refined using a CTT task model. Task models represent the intersection between user interface design and more engineering approaches by providing designers with a means of representing and manipulating a formal abstraction of activities that should be performed to reach user goals. In our example, essential use cases (generated with Microsoft Visio) and IDEALXML editor can be used to specify task models (Fig. 7).

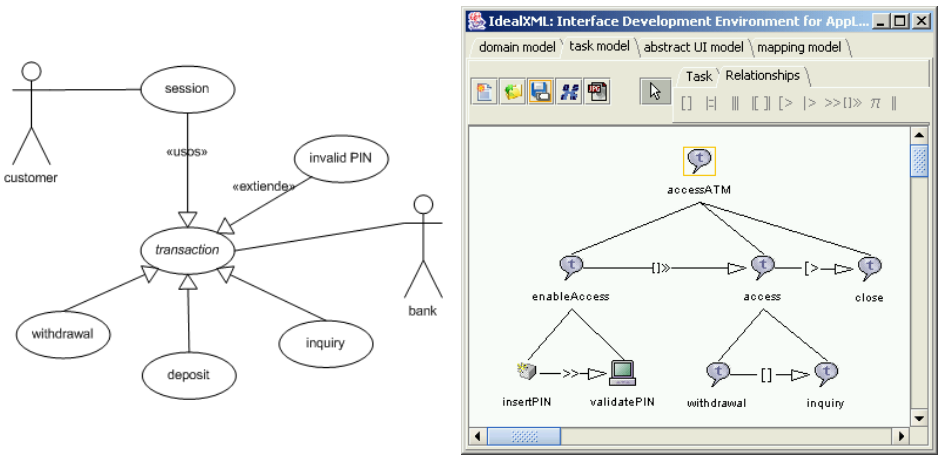

Fig. 7. Essential use cases in ATM example and IDEALXML

4.3 Presentation model

The presentation model describes the visual aspects of the interface. It is divided into abstract presentation model and concrete presentation model. Abstract presentation model provides an abstract view of an interface that is independent from the underlying concrete model and concrete presentation model is the concrete instance of an interface which can be presented to an user; there may be many concrete instances of an abstract presentation model. In this moment, IDEALXML is interested only in abstract presentation models manipulation and this tool uses a *wireframe* graphical notation inspired from UsiXML [22]. A very simple abstract presentation model, related with ATM example, is shown in the Fig. 8, it is an abstract presentation associated with a starting session where customer is asked to enter his/her PIN.

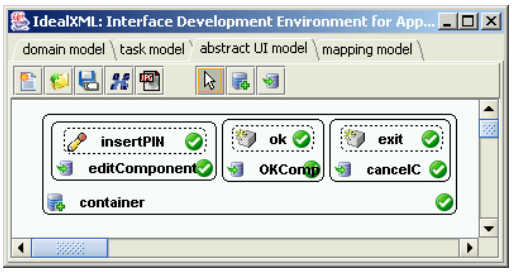

Fig. 8. IDEALXML and abstract presentation models.

Abstract presentation patterns can be found in works of [26, 40, 44, 46]. An example of a pattern language based on Conceptual User Interface Patterns was introduced in [26, 27], called JUST-UI. This collection tries to identify patterns in such UIs and abstract them to work in terms of the problem domain.

Ergonomic criteria are used to classify presentation patterns. They were viewed as a means of defining and operationalising dimensions of usability (*Guidance, Workload, Explicit control, Adaptability, Error Management, Consistency, Significance of codes* and *Compatibility* [3]).

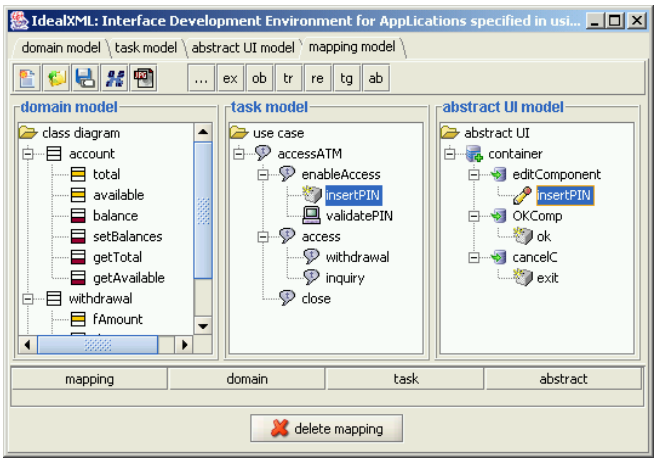

Fig. 9. IDEALXML and mapping between the models

4.4 Mapping model

Under a MB-UIDE philosophy, an user interface design consists of a set of models and associations (mappings) between the models [17]. It has been noted by [13] that one of the main challenges faced by model-based systems is how to associate elements from one model with their related counterpart(s) in other models, henceforth called the *mapping model* [17].

UsiXML identified several relationships to make explicit the relationships between the models [21]. We can establish a set of mappings using IDEALXML, at this

moment manually. But there is a work-in-progress version where these mappings are established semi-automatically by using mapping patterns.

5 Conclusions

In both software and architecture design exist a lingua franca for the specification of the recurring patterns that appear throughout their development: UML. These patterns gather the experience and good practices learnt designers in their day-to-day work.

Knowing patterns makes it easier to understand existing systems. Interaction patterns bring the same benefits into user interface design, they can also make us better designers. Most model-based user interface environments use interaction patterns, and describing a user interface in terms of the patterns it uses will make it a lot easier to understand. Having interaction patterns as a common vocabulary means we do not have to describe the pattern solution each time it is used; we can just name it and expect your reader to know it. We use these patterns in our own designs, and we have found them invaluable. It's easy to imagine more sophisticated ways of using patterns such as as pattern-based CASE tools or hypertext documents. But patterns are of big help even without sophisticated tools. This paper presents IDEALXML, a pattern-based environment where interaction patterns can be stored, edited, manipulated and used in user interfaces development again and again in a semi-automatic way allowing for the reuse of gathered experienced in user interface design.

Acknowledgements

This work was supported by the Spanish CICYT project TIN2004-08000-C03-01.

References

1. Abstract User Interface Markup Language (AUIML). http://www.alphaworks.ibm.com/tech/auiml
2. Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Constructions*. New York: Oxford University Press.
3. Bastien, J.M.C. and Scapin, D.L., *Evaluating a User Interface with Ergonomic Criteria,* International Journal of Human-Computer Interaction, Vol. 7, 1995, pp. 105–121
4. Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., et al. (1998). Putting it all Together: Towards a Pattern Language for Interaction Design. *SIGCHI Bulletin, 30*(1), 17-24
5. Booch, G., Rumbaugh, J, Jacobson, I. *The Unified Modeling Language User Guide* (Addison-Wesley, 1999) ISBN 0-201-57168-4
6. Borchers, J. *A Pattern Approach to Interaction Design.* John Wiley & Sons, Chichester, UK, 2001.
7. Borland® JDataStore™. http://www.borland.com/jdatastore
8. Chevalier, A., Ivory, M. Web Site Designs: Influence of Designer's Experience and Design Constraints. *International Journal of Human-Computer Studies, 58,* 2003, p. 57-87.

9. Coad, P. Object-oriented patterns. Communications of the ACM. Sept 1992. v35 n9 p152(8)
10. Coad, P.: Object-Oriented Patterns. Commun. ACM 35(9): 152-159 (1992)
11. da Silva, P.P., and Paton, N.W., A UML-Based Design Environment for Interactive Applications, Proceedings of the 2nd Int. Workshop on UIDIS, E. Kapetanios and H. Hinterberger (eds), 60-71, IEEE Computer Society, 2001
12. Erickson T. Patterns Languages as Languages. CHI'2000 Workshop: Pattern Languages for Interaction Design, 2000
13. eXtensible Interface Markup Language (XIML) a universal language for user interfaces: http://www.ximl.org
14. Fincher, S. (2000, 7th September 2000). *The Pattern Gallery*, from http://www.cs.ukc.ac.uk/people/staff/saf/patterns/gallery.html
15. Fincher, S., Finlay, J., Greene, S., Jones, L., Matchen, P., Thomas, J, Molina, P.: Perspectives on HCI patterns: concepts and tools. CHI Extended Abstracts 2003
16. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusets, US: Addison-Wesley
17. Griffiths, T., Barclay, P., Paton, N., McKirdy, J., Kennedy, J., Gray, P., Cooper, R., Goble, C., Silva, P. Teallach: A Model-Based User Interface Development Environment for Object Databases. In *Interacting with Computers* 14(1), pages 31-68, December 2001
18. IBM guidelines. http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/572
19. ISO Draft International Standard (DIS) 9241, *Ergonomic Requirements for office work with visual display terminals,* International Standards Organization, Geneva, 1999.
20. Jameson, A. User Modeling: Proceedings of the Sixth International Conference, UM97., Paris, C., & Tasso, C. (Eds.) Vienna: Springer Wien New York. 1997
21. Limbourg, Q., Vanderdonckt, J., Addressing the Mapping Problem in User Interface Design with UsiXML, Proc. of 3rd Int. Workshop on TAMODIA'2004 (Prague, November 15-16, 2004), Ph. Palanque, P. Slavik, M. Winckler (eds.) 2004, pp. 155-163
22. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on EHCI-DSVIS'2004 Hamburg, July 11-13, 2004
23. Longhorn markup language (XAML): http://longhorn.msdn.microsoft.com/portal_nav.htm
24. Macintosh Human Interface Guidelines. http://developer.apple.com/documentation/mac/HIGuidelines/HIGuidelines-2.html
25. Microsoft. Official Guidelines for User Interface Developers and Designers. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_UIDesignDev.asp
26. Molina, P., Belenguer, J., Pastor, O.: Describing Just-UI Concepts Using a Task Notation. DSV-IS 2003: Madeira, Portugal. 2003
27. Molina, P., Pastor, O., Martí, S., Fons, J., Insfrán, E.: Specifying Conceptual Interface Patterns in an Object-Oriented Method with Automatic Code Generation. UIDIS 2001
28. Molina, P: A Review to Model-Based User Interface Development Technology. MBUI 2004
29. Nicola, J., Mayfield, M., Abney, M. Streamlined Object Modeling. Prentice Hall. 2001
30. Nielsen, J. *A Virtual Protocol Model for Computer-Human Interaction*, International Journal of Man-Machine Studies, Vol. 24, No. 3, 1986, pp. 301-312
31. Nielsen, J., Mack, R.M., *Usability Inspection Methods*, John Wiley & Sons, New York, 1994
32. Object Management Group, *UML Home Page*, http://www.uml.org
33. Paternò, F. Model-based design and application evaluation of interactive application. Springer. 1999
34. Pattern Language Markup Language (PLML): http://www.hcipatterns.org
35. Schulungbaum, E. Model-based user interface software tools – current state of declarative

models. Graphics, visualization and usability centre, Georgia Institute of Technology, GVU Tech #96 #30. 1996
36. Schümmer, T., Borchers, J., Thomas, J., Zdun, U.: Human-computer-human interaction patterns: workshop on the human role in HCI patterns. CHI Extended Abstracts 2004
37. Szekely, P. Readings in inteligent user interfaces. Reflections on Beyond Interface builders: Model-based Interface Tools. Morgan Kauffmann, Los Altos, Ca.1998
38. Szekely, P. Retrospective and challenges for model-based interface developments. In: Bodart F., Vanderdonckt, J. (eds.) Design, Specification and Verification of Interactive Systems. 1995
39. The ConcurTaskTree Environment (CTTE): http://giove.cnuce.cnr.it/ctte.html
40. Tidwell, J. UI Patterns and Techniques. http://time-tripper.com/uipatterns/index.php
41. *useML: A Human-Machine Interface Description Language.* http://www.edm.luc.ac.be/uixml2004/accepted/zuehlke2004.html
42. User Interface Markup Language (UIML): http://uiml.org
43. UsiXML: http://www.usixml.org
44. van Duyne, D., Landay, J., Hong, J. *The Design of Sites.* Addison Wesley, 2003.
45. Vanderdonckt, J., *Development Milestones towards a Tool for Working with Guidelines*, Interacting with Computers, Vol. 12, N°2, 1999, pp. 81-118. Accessible at http://www.elsevier.nl/gej-ng/10/23/72/31/21/show/toc.htt
46. Weile, M. …patterns in interaction design: http://www.welie.com
47. XML language for describing Abstract User Interfaces (AUI). http://giove.cnuce.cnr.it/teresa
48. XML User Interface Language (XUL): http://www.mozilla.org/projects/xul/