Solving the Mapping Problem in User Interface Design by Seamless Integration in IDEALXML

Francisco Montero[1,2], Víctor López-Jaquero[1,2], Jean Vanderdonckt[2], Pascual González[1], María Lozano[1], and Quentin Limbourg[2]

[1] Laboratory on User Interaction & Software Engineering (LoUISE), University of Castilla-La Mancha, 02071 Albacete, Spain
{fmontero, victor, pgonzalez, mlozano}@info-ab.uclm.es
[2] Belgian Laboratory of Computer-Human Interaction (BCHI), Université Catholique de Louvain, 1348 Louvain-la-Neuve, Belgium
{Montero, lopez, vanderdonckt, limbourg}@isys.ucl.ac.be

Abstract. The mapping problem has been defined as the way to map models involved throughout the development life cycle of user interfaces. Model-based design of user interfaces has followed a long tradition of establishing models and maintaining mappings between them. This paper introduces a formal definition of potential mappings between models with its corresponding syntax so as to create a uniform and integrated framework for adding, removing, and modifying mappings throughout the development life cycle. For the first time, the mappings can be established from any source model to any target model, one or many, in the same formalism. Those models include task, domain, presentation, dialog, and context of use, which is itself decomposed into user, platform, and environment. IDEALXML consists of a Java application allowing the designer to edit any model at any time, and any element of any model, but also to establish a set of mappings, either manually or automatically based on a mapping model.

1 Introduction

One of the existing approaches in development of software consists in establishing a model of the future software to be developed and to produce code from this model. This approach is the cornerstone of the Model-Driven Architecture (MDA) [3] and largely contrasts with traditional approaches where the software is directly coded without any model or specifications. The development of the User Interface (UI), one component of the software, does not escape from this observation [16]. Typically, A *UI model* is referred to as a set of concepts, a representation structure and a series of primitives and terms that can be used to explicitly capture knowledge about the UI and its related interactive application using appropriate abstractions [33].

Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on relevant ones [22]. The models can be developed as a precursor to implementing the physical system, or they may be derived from an existing system or a system in development as an aid to understanding its behaviour. The most recent innovations have focused on notations and tools that allow users to express systems perspectives of value to software architects and developers in ways that are readily mapped into the programming language code that can be compiled for a particular operating system platform. The current state of this practice employs the Unified Modelling Language (UML) [15, 26] as the primary modelling notation. However, despite UML Use Cases, Activity Diagrams and other notations can effectively capture functional requirements or specify detailed behaviours, UML does not specifically support the modelling of user interface aspects [19]. So, the last generation of model-based approaches to user interface design agree on the importance of task models [18, 20].

In this paper, we present an initial pattern-based general solution to the mapping problem in model-based interface development. The main function of a model-based interface development system (MB-UIDE) is to provide the software tools that allow developers to construct user interfaces by means of creating and refining an interface model [23]. The success of MB-UIDE systems has been limited. On one hand, there are systems that can generate specific-type interfaces with a high degree of automation. On the other hand, none of the knowledge based approaches for interface generation used by model-based systems is applicable beyond its intended narrow target domain nor can they be generalized to other targets [1, 2, 4, 14, 24]. In our proposal knowledge in form of patterns is used in the development process to help the developer in the model process. We are working in three directions methodologies [8, 22, 29], languages [25, 27, 31] and tools to support both.

The structure of the paper is as follows: the first section deals with related works and then User Interface extensible Markup Language (UsiXML) [13] and IDEALXML environment are presented using an example. Models of a MB-UIDE are introduced using this example, and screenshots of IDEALXML are shown in terms of the same example. We finish the paper by outlining the conclusions.

2 Related Works

To uniformly present work related to the mapping problem [9, 11, 23], we will select some significant and representative efforts made in already existing environments supporting a model-based approach and present them according to a similar framework that represents the various levels and models where a UI development process may appear.

Puerta [23, 24] presented a general framework to solve the mapping problem in model-based interface development systems. They identified the nature of the mapping problem as one of bridging levels of abstraction in an interface model, by explicitly representing mappings in an interface model, by providing tools that allow developers to set and inspect the mapping, and by affording developers knowledge-based approaches to prune the design space of potential mappings. MOBI-D [23], the interface development environment, deals with only a few of the interesting mapping situations in any user interface design. MOBI-D provides a decision-support tool called TIMM for the abstract-to-concrete mappings.

The Mastermind Dialog Language (MDL) [28] is a deterministic notation for expressing task hierarchies and the binding of task and presentation models. MDL has a syntax for specifying task models and additional features for binding tasks with presentations. The high-level syntax of MDL is a collection of module declarations.

MDL defines three categories of module, each of which represents a different technique for defining a process. In a *task*, a process is defined as a hierarchy of user tasks, the leaves of which denote actions. In an *extern*, a process is defined implicitly as a continuously available collection of anonymous actions. Finally, in a *binding*, a process is defined as the coordination of one task and one or more externs.

Teallach [12] uses mapping rules in several places in its architecture to allow mappings between the various models. For example, a set of mapping rules exist between the task model and its abstract presentation model counterpart. In addition to these mappings, an additional set of rules exist between the abstract and concrete presentation models. These mapping rules take into consideration the information captured in the user model, to provide the intended users of the system with a generated interface suitable to their requirements.

Vampire [7] enables designers to manually establish relationships between parts or whole of UIs drawn in a UI builder and a task model presented in a lateral window. In this way it is more easy to understand how each task is presented by which UI components, such as windows, dialog boxes and how leaf nodes of such tasks are mapped onto widgets. However, the relationships remain manual without any further exploitation in the rest of the development life cycle.

The next section introduces a language and a tool—UsiXML [13] and IDEALXML—using an example. Patterns [21] are used when we want to write models using IDEALXML. A *pattern* is an abstraction of a doublet, triplet, or other small grouping of entities that is likely to be helpful again and again in MB-UIDE. An entity is any element that we use in building a model, for instance if we want to model a domain, we will use a class diagram. In this context, patterns consist of classes, attributes and methods, but if we want to model a use case, we will have tasks and relationships between them. Patterns can be gathered using UsiXML. Patterns are found by trial-and-error and by observation [5]. By building many user interfaces models and by observing many applications of the lowest-level building blocks and the mappings established between them, one can find patterns. With such patterns, as Alexander observed, *the things which seem like elements dissolve*, and we are able to use a higher-level building block for modelling (task, domain, abstract UI or mapping).

As we can see in this section we have not an integrated method or tool to address the mapping problem or to use experience gathered using patterns in general in a way that is uniform and rigorous. This work tries to give a step forward in this sense.

3 UsiXML and IDEALXML Environment

MB-UIDEs [8, 22, 29] seek to describe the functionality of a user interface using a collection of declarative models. In such a context, constructing a user interface involves building and linking a collection of models. So, a model-based approach to UI design and implementation provides multiple, separate models of different facets of the UI. This approach is complicated by the multi-model binding problem, which concerns how a designer is able to bind behavior that is described in another model. Many user interface description languages have been introduced so far that address different aspects of a User Interface. This proliferation results in many XML-compliant dialects that are not widely used and that do not allow interoperatibility between tools that have been developed around the UIDL.

IDEALXML is a tool to support UsiXML[13], like GrafiXML or VisiXML. UsiXML consists of a User Interface Description Language allowing designers to apply a multi-directional development of user interfaces at multiple levels on independence, and not only device independence. IDEALXML consists of a Java application allowing the designer to edit any model and element of any model at any time where experience, using patterns, can be used. But also to establish a set of mappings, either manually or automatically based on a mapping model.

In order to develop a UI using UsiXML and IDEALXML environment we follow these steps: (1) requirements analysis, (2) edit the task model, (3) edit the domain model, (4) identify patterns in the domain model according to the task model, (5) derive an AUI by application of patterns and generalization of relevant mappings, (6) from the AUI, retrieve a pattern CUI thanks to transformation by patterns, (7) repeat this until all parts of the task model are gone and, finally, (8) assemble the code generated by GrafiXML. Our example will be a typical page in many website. A page where a user can request one or several catalogs filling a form where that user provides his/her name, address, email and reference of his/her preferences of information. Baring these steps in mind, in UsiXML are considered different models.

A *task model* describes the various tasks to be carried out by a user in interaction with an interactive system. A version of ConcurTaskTrees (CTT) [20] has been selected to represent user's tasks along with their logical and temporal ordering. IDEALXML provides tools to specify UsiXML in a graphical way. In Fig. 1 we can see tasks and relationships between those tasks.

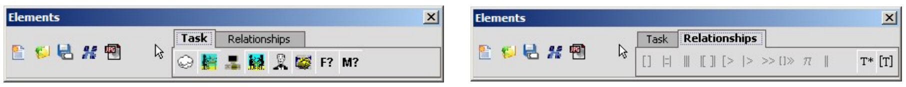


Fig. 1. Toolbox associated with task model tab

Using a CTT notation and IDEALXML we can specify our use case (Fig. 2). We have abstract tasks ( ) (AskCatalog, SendRequest). These tasks consist of several actions related with interactive tasks ( ) that involve an active interaction of the user with the system (selecting, editing, etc.) and system tasks ( ) are actions that are performed by the system (validation, send, etc.). Relationships are established between tasks for instance parallelism (|||) where T1 is interleaved with T2 (T1 and T2 are tasks) or enabling (>>) where T1 has to be finished in order to initiate T2 and T2 is synchronized with T1 on some piece of data. By building many task models and by observing many applications of the lowest-level building block and the relationships established between them, we can find patterns. Many patterns [30, 32, 34] can be modeled using the UsiXML language and edited using IDEALXML. So, patterns that can be found in [34] (Web design patterns section) such as login, registering, simple search, advanced search, breadcrumbs and main navigation can have an associated UsiXML description. In general, any pattern related with a task can be modeled using CTT notation and UsiXML. Many of these patterns have an associated user interface as well—this aspect will be dealt with in the abstract UI model. Tasks are mapped to domain elements (attributes and methods) following *manipulate* relationships.

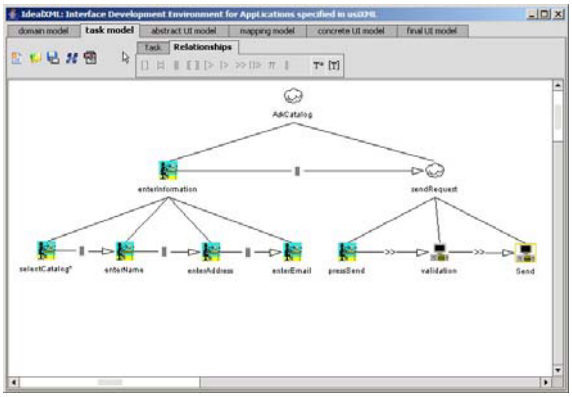
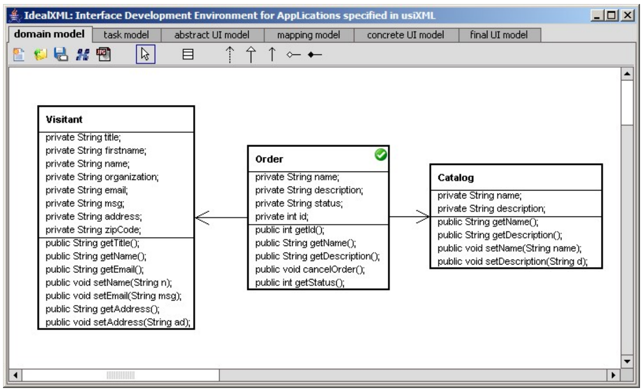


Fig. 2. Task model editor using a variant of CTT notation



```
Visitant
private String title;
private String firstname;
private String name;
private String organization;
private String email;
private String msg;
private String address;
private String zipCode;
public String getTitle();
public String getName();
public String getEmail();
public void setName(String n);
public void setEmail(String msg);
public String getAddress();
public void setAddress(String ad);
```

```
Order
private String name;
private String description;
private String status;
private int id;
public int getId();
public String getName();
public String getDescription();
public void cancelOrder();
public int getStatus();
```

```
Catalog
private String name;
private String description;
public String getName();
public String getDescription();
public void setName(String name);
public void setDescription(String d);
```

Fig. 3. Domain model editor in IdealXML

A *domain model* describes the real-world concepts, and their interactions as understood by users and the operation that are possible on these concepts. Domain model concepts are classes, attributes, methods and domain relationships (Fig. 3). The IDEALXML environment allows the user to construct class diagrams. In our example, we have three entities: Visitant, Order and Catalog (Fig. 3). In each of these classes we have attributes and methods. Attributes enable a description of a particular feature of a class and methods are presences which are called either by objects of the domain or by interface components.

The development of a domain model can be supported by using patterns. Patterns for business object modelling [7, 17] are not the same as design patterns which aim to increase reuse and framework *pluggability*. Business patterns, also known as analysis patterns [10], focus on creating an object model that clearly communicates the business requirements. The key to modelling business processes is not to focus on the steps of the process, but instead to focus on the people, places, things and events involved in the process. So, Nicola [17], for instance, gathered twelve collaboration patterns that are used for developing domain models. Examples of these patterns are: actor-role, outerPlace-place, item-specificItem, assembly-part, container-content, and role-transaction. In the example, role-transaction and specificItem-transaction patterns were used.

An *Abstract User Interface* (AUI) model is a user interface model that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specificities. An AUI is populated by *Abstract Interaction Objects* and Abstract user interface relationships. *Abstract Interaction Objects* (AIO) may be of two types: Abstract Individual Components (AIC) and Abstract Containers (AC). An *Abstract Individual Component* is an abstraction that allows the description of interaction objects in a way that is independent of the modality in which it will be rendered in the physical world. An AIC may be composed of multiple facets. Each facet describes a particular function an AIC may endorse in the physical world.

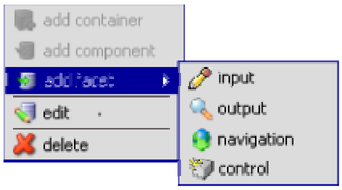


Fig. 4. Contextual menu in abstract UI model editor

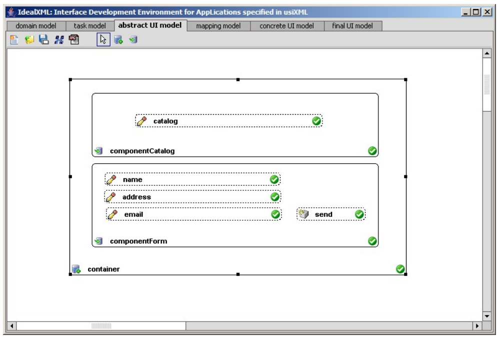


Fig. 5. Containers, components, facets and abstract UI models

Four main facets are identified (Fig. 4): an *input* facet describes the input action supported by an AIC, an *output* facet describes what data may be presented to the user by an AIC, a *navigation* facet describes the possible container transition a particular AIC may enable, and a *control* facet describes the links between an AIC and system functions (i.e., methods from the domain model when existing). An AC is an entity allowing a logical grouping of other abstract containers or abstract individual components. AC are said to support the execution of a set of logically/semantically connected tasks. They are called presentation units in [33]. AIC and AC may be reified at the concrete level, into one or more graphical containers like windows dialog boxes, layout boxes or time slots in the case of auditory user interfaces. In this model it is possible to establish relationships. An important relationship is the Dialog control relationship. This relationship allows a specification of a flow of control between the abstract interaction objects and can be derived from task model relationships.

In a similar way, like other models, patterns can be used here to build abstract UI. Many patterns that can be found in the literature and in websites can be described using UsiXML and edited in IDEALXML, and these drafts using abstract components are mapped with task following *isExecutedIn* mappings.

In Fig. 5, we can see a container with two components, one of them is associated with *catalogs* and the other with the *user information*. The second component has several facets because these facets are related with an entity, in other case each component normally has one facet.

A *mapping* model is a well-known issue in transformation driven development of UI [23]. Rather than proposing a collection of unrelated models and model elements, our proposal provides a designer with a set of pre-defined relationships allowing a mapping of elements from heterogeneous models and viewpoints. This may be useful, for instance, for enabling the derivation of the system architecture, for traceability in the development cycle, for addressing context sensitive issues, for dialog control issues, for improving the preciseness of model derivation heuristics. Several relationships may be defined (Table 1) to explicit the relationships between the domain model and the UI models:

Table 1. Mappings in UsiXML

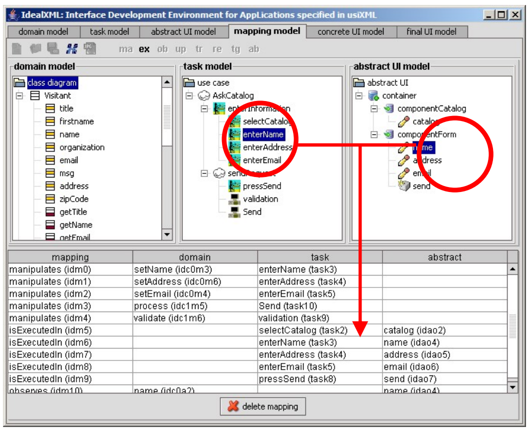
Relationship	Description
observes	It is a mapping defined between an interaction object and a domain model concept (ex. A mapped attribute is modified or a mapped method is executed)
updates	It is a mapping defined between an interaction object and a domain model concept (specially, an attribute)
triggers	It is a mapping defined between an interaction object and a domain model concept (specifically, an operation)
isReifiedBy	Indicates that a concrete object is the reification of an abstract one through a reification transformation
isAbstracteInto	Indicates that an abstract object is the reification of a concrete one through an abstraction transformation
manipulates	Maps a task to a domain concept. It may be an attribute, a set of attributes, a class, or a set of classes
isExecutedIn	Maps a task to an interaction object allowing its execution
hasContext	Maps any model element to one or several context of use

Puerta, [23] identified different kinds of mappings. Some of them are considered in the selected notations used for specifying models. For instance, CTT notation is used in our proposal because it includes relationships between tasks where task-dialog mappings are gathered. Analogously, presentation-dialog mappings are included in the Abstract UI notation where dialog control relationships allow a specification of a flow of control between the abstract interaction objects.

IDEALXML, considering these mappings (Table 1), can handle the mapping problem between models thanks to the UsiXML language that serves as a uniform language between heterogeneous models (Fig. 6). Users can select elements (attribute, method, task or AIO) and define mappings between them. In IDEALXML we can define *observes*, *updates*, *manipulates* and *isExecutedIn*. Other mappings (*isAbstractedInto*, *isReifiedBy* and *hasContext*) will be considered when integration between IDEALXML and GrafiXML is done.

A *Concrete User Interface* (CUI) model is a UI model allowing a specification of an appearance and behavior of a UI with elements that can be perceived by users. A CUI model is composed of Concrete Interaction Objects (CIO) and concrete relationships. Concrete interaction objects and relationships are further refined into graphical objects and relationships and auditory objects and relationships. A CIO is defined as an entity that users can perceive and/or manipulate. Dialog control defined in an Abstract UI model allows a specification of a flow of control between the concrete interaction objects. The philosophy of our proposal is shown in the Fig. 8. We have experience (patterns) [7, 17, 30, 32, 34], it is gathered and documented using UsiXML and it is used in IDEALXML in user interface development process following a MB-UIDE.

Furthermore, in IDEALXML, two tools make it possible to obtain a graphical rendering from a CUI specification. GrafiXML is equipped with an export module that allows a generation XHTML code and Java Swing objects. TransformiXML allows an interpretation of a CUI specification directly in flash. In this case a CUI may be assimilated to the final user interface. (Fig. 7).



mapping	domain	task	abstract
manipulates (idm0)	setName (idc0m3)	enterName (task3)	
manipulates (idm1)	setAddress (idc0m6)	enterAddress (task4)	
manipulates (idm2)	setEmail (idc0m4)	enterEmail (task5)	
manipulates (idm3)	process (idc1m5)	Send (task10)	
manipulates (idm4)	validate (idc1m6)	validation (task9)	
isExecutedIn (idm5)		selectCatalog (task2)	catalog (idao2)
isExecutedIn (idm6)		enterName (task3)	name (idao4)
isExecutedIn (idm7)		enterAddress (task4)	address (idao5)
isExecutedIn (idm8)		enterEmail (task5)	email (idao6)
isExecutedIn (idm9)		pressSend (task8)	send (idao7)
observes (idm10)	name (idc0a2)		name (idao4)

Fig. 6. Mapping model in IDEALXML

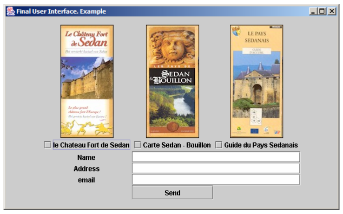
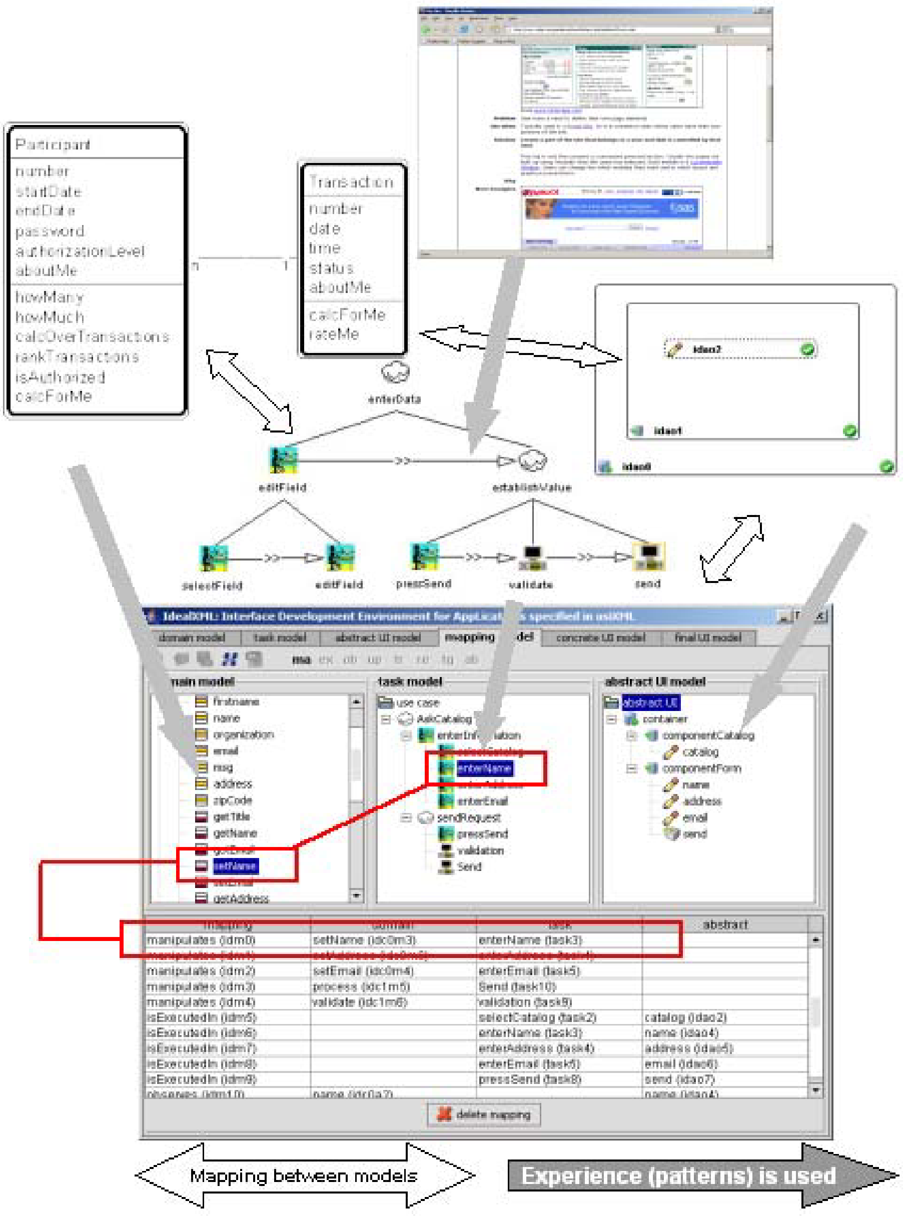


Fig. 7. Final UI with Java Swing objects

Editing a concrete UI in UsiXML directly can be considered as a tedious task, for this reason a specific editor called GrafiXML [13] has been developed to face the development of CUI models. Associated with each element in domain, task, abstract UI or concrete UI we have information that finally is gathered in a declarative way using UsiXML. Different specifications may be useful to adapt it to different categories of users or different environments. At the moment, different transformations are possible: from task and domain to task and domain, from abstract UI to abstract UI, and from concrete UI to concrete UI. It may be done using TransformiXML [13] (tool



mapping	domain	task	abstract
manipulates (idm0)	setName (idc0m3)	enterName (task3)	
manipulates (idm1)	setAddress (idc0m5)	enterAddress (task4)	
manipulates (idm2)	setEmail (idc0m4)	enterEmail (task5)	
manipulates (idm3)	process (idc1m5)	Send (task10)	
manipulates (idm4)	validate (idc1m6)	validation (task9)	
isExecutedIn (idm5)		selectCatalog (task2)	catalog (idao2)
isExecutedIn (idm6)		enterName (task3)	name (idao4)
isExecutedIn (idm7)		enterAddress (task4)	address (idao5)
isExecutedIn (idm8)		enterEmail (task5)	email (idao6)
isExecutedIn (idm9)		pressSend (task8)	send (idao7)
observes (idm10)	name (idc0a7)		name (idao4)

Fig. 8. IDEALXML environment: experience and mapping problem

under development). With this tool a user chooses an input file containing models to transform, generated using IDEALXML or GrafiXML. Patterns are a good source of inspiration for atomic transformation techniques because most patterns are based on a combination of several simpler techniques. That's why patterns aren't always easy to understand in depth.

4 Conclusion

The related work section emphasizes that so far no integrated method or tool exists to address the mapping problem in general in a way that is both uniform and rigorous. In this paper, we present an integrated environment tool, IDEALXML, that can handle the mapping problem (Fig. 8) using UsiXML language. IDEALXML makes it possible to specify in a WYSIWYG manner the task model, the domain model, the abstract user interface model and the mapping model.

The task model is based on the CTT notation introduced by [20]. The domain model is represented with a class diagram. The abstract UI model has the form of a hierarchical structure of embedded boxes whose leaves are abstract individual components and their facets. Mapping model establishes relationships between models, it is useful for enabling the derivation of the system architecture and for traceability in the development cycle. This paper integrates all of the traditional models: task, domain, abstract UI, mapping, concrete and final into one single environment with their respective editing environment. This integration makes is possible to establish mappings in a logical way (rather than being implicitly coded in the tools). These mappings can then be exploited manually thanks to a pattern-based approach or automatically thanks to a transformation engine (TransformiXML). In this way, we can achieve some continuity, some seamlessness through the development life cycle.

Acknowledgments

This work is partly supported the Spanish PBC-03-003 and CICYT TIN2004-08000-C03-01 grants. Also, we gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating HCI similar to human-human communication of the European Sixth Framework Program.

References

1. Ali, M.F., Pérez-Quiñones M.A., Abrams M. Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.): *Multiple User Interfaces: Engineering and Application Framework*. John Wiley and Sons, New York, 2003
2. Berti, S., Mori, G., Paternò, F., Santoro, C. A Transformation-Based Environment for Designing Multi-Device Interactive Applications. *Proc. of 9th Int. Conf. on Intelligent User Interfaces IUI'2004* (Funchal, January 13-16, 2004). 2004, 352–353.
3. Brown, A. An introduction to model driven architecture. Part I: MDA and today's systems. IBM. 2004
4. 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, 289–308.
5. Coad, P. Object-oriented patterns. Communications of the ACM, Sep. 1992, vol. 35, no. 9, pp. 152–159.
6. Coad, P., North, D., Mayfield, M. Object Models: Strategies, Patterns and Applications. Prentice Hall, 1997.

7. Eisentein, J., Rich, C. Agents and GUIs from task models. In proceedings of 7th ACM Conference on Intelligent User Interfaces IUI 2002, pp. 47–54. ACM Press. New York, 2002
8. Eisenstein, J., Vanderdonckt, J., Puerta, A. Applying model-based techniques to the development of UIs for Mobile Computers. Proceedings IUI´01: International Conference on Intelligent User Interfaces, pp. 69–76. ACM Press. 2001
9. Elnaffar, S., Graham, N. Semi-automated linking of user interface design artifacts. In *Proceedings of Computer Aided Design of User Interfaces (CADUI '99)*, pp. 127–138, Kluwer Academic Publishing, 1999.
10. Fowler, M., Analysis Patterns: Reusable Object Models. Addison-Wesley. 1996
11. Fowler, R. Direct Mapping and User Interface. Technology of Object-Oriented Languages and Systems. *In: Proceedings of the Technology of Object-Oriented Languages*, p. 574, IEEE Computer, 1999
12. Griffiths, T., Barclay, P., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C., and Pinheiro da Silva, P. Teallach: a Model-based User Interface Development Environment for Object Databases. *Interacting with Computers 14*, pp. 31–68, 2001.
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and López-Jaquero, V., USIXML: a Language Supporting Multi-Path Development of User Interfaces. *Proc. of 9th IFIP Engineering Human Interaction and Interactive Systems*, 2004
14. 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 10th Int. DSV-IS'2003.* 2003, 245–257.
15. Markopoulos, P., Marijnissen, P. UML as a representation for Interaction Design. Proceedings OZCHI 2000, 240-249.
16. Myers, B., Hudson, S., Pausch, R. Past, Present and Future of user interface software tools. ACM Transactions on Computer-Human Interaction (TOCHI), vol. 7, no. 1, pp. 3–28, 2000.
17. Nicola, J, Mayfield, M., Abney M. Streamlined Object Modeling. Prentice Hall. 2002
18. Paris, C., Lu, S., Vander Linden, K. Environments for the Construction and Use of task models. In *The Handbook of Task Analysis*, D. Diaper and N. Stanton (eds), 2003, chapter 23, pages 467-482.
19. Paternò, F. ConcurTaskTrees and UML: how to marry them?. http://giove.cnuce.cnr.it/Guitare/ Document/ConcurTaskTrees_and_UML-new.htm
20. Paternò, F. Model-based design and evaluation of interactive application. Springer-Verlag. 1999
21. Pescio, C. Principles Versus Patterns. IEEE Computer. September, 1997.
22. Puerta, A.R. A Model-based Interface Development Environment. *IEEE Software 14*, 4. 1997, 40–47.
23. Puerta, A.R. and Eisenstein, J. Towards a General Computational Framework for Model-Based Interface Development Systems. *Knowledge-based Systems* 1999
24. Puerta, A.R. and Eisenstein, J. XIML: A Multiple User Interface Representation Framework for Industry. John Wiley & Sons, New York 2003.
25. Rumbaugh, J., Jacobson, I., Booch, G. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999
26. Souchon, N. and Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. *Proc. of 10th Int. DSV-IS'2003.* 2003, 377–391.
27. Stirewalt, R.E.K. and Rugaber, S. The Model-Composition Problem in User-Interface Generation. *Automated Software Eng. 7,* April 2000, 101–124.

28. Szekely, P., Sukaviriya, P., Castells, J., Muthukumarasamy and Salcher, E. Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Appproach. *Proc. of 6th IFIP EHCI'95*, pp. 120–150, Chapman Hall, 1996
29. Tidwell, J. UI Patterns and Techniques. http://www.mit.edu/~jtidwell/
30. Trætteberg, H., Molina, P.J., Nunes, N.J. Making Model-Based UI Design Practical: Usable and Open Methods and Tools, *Proc. of IUI'2004* (Funchal, January 13-16, 2004), pp. 376–377, ACM Press, New York. 2004
31. Van Duyne, D., Landay, J., Hong, J. The design of sites: patterns, principles and proceses for crafting a customer-centered web experience. Addison-Wesley, 2002
32. Vanderdonckt, J. and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. *Proc. of the ACM INTERCHI'93.* ACM Press, New York 1993, 424–42
33. Welie, M., Patterns in interaction design. http://www.welie.com