Model-Based Design of Adaptive User Interfaces through Connectors

Víctor López-Jaquero, Francisco Montero, José. P. Molina,
Antonio Fernández-Caballero, Pascual González

Laboratory of User Interaction and Software Engineering (LoUISE)
University of Castilla-La Mancha, Albacete, Spain
{victor|fmontero|jpmolina|caballer|pgonzalez}@info-ab.uclm.es

Abstract. For a long time standard desktop applications have ruled the market. Nevertheless, the availability of information has made the user demand new interaction techniques in completely different contexts and devices, but requesting the same functionality. With this new situation application design should be able to adapt to these differences. To design these adaptive interfaces the specification of these user interfaces should support plasticity at runtime. In this paper a first approach is proposed to support these plasticity features from a formal point of view within a model-based user interface design methodology. Connector paradigm is used to coordinate the communication between Concrete Interaction Objects and Abstract Interaction Objects in a flexible way enough to support adaptivity.

1 Introduction

For a long time standard desktop applications have ruled the market. Personal computers became the main focus as the final target for both business and leisure applications. At that time, a user was some kind of expert that had usually even some programming skills. Thus, research in designing process was centred in those kinds of target platforms for that kind of users. Then Internet flood the market with completely different concepts, content was the main focus instead of the traditional focus on functionality as data processing. Nevertheless, the availability of information has made the user demand new interaction techniques. These new techniques need to fulfil this new demand for information at any place and using any device. The ways information is used, the contexts (situations and places) and the devices involved in interaction are so assorted that it is almost impossible to predict every use for the information available [19].

On the other hand, as a new generation of devices and new interaction contexts are introduced another important fact needs to be taken into account: the diversity in the level of experience of the users. When an application is made available through the Internet it will be available to people from different countries, from different cultures, with different skills and preferences. We talk about user-centred design. But, how "user-centred" can this design be if we don't know the user that will interact with the system?

User interface design must address these two up-to-date challenges: (1) designing for many different interaction devices and, (2) designing for many different users.

To address these problems user interfaces should be able to adapt themselves to the platform they run on and to each individual user (see figure 1). These properties for user interfaces are referred to as adaptivity [2].

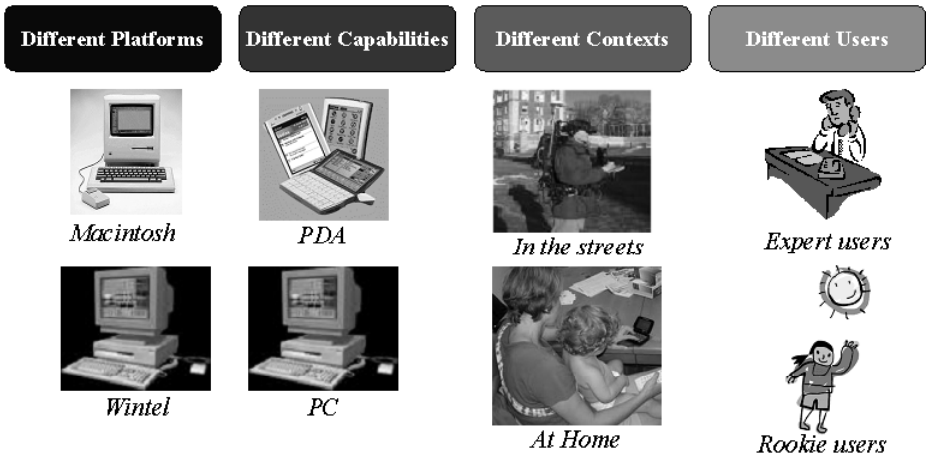

Fig. 1. Diversity in interaction: platform, capabilities, context and user aware design.

There are different ways to achieve this adaptivity, but one of the main ways to support adaptivity is to design user interfaces with *plasticity*. The term 'plasticity' is inspired from the property of materials that expand and contract under natural constraints without breaking, thus preserving continuous usage. Plasticity [4] is the capacity of a user interface to withstand variations of both the system physical characteristics and the environment while preserving usability. One of the more common ways to preserve usability at runtime is checking whether the identified interaction patterns [13] are being used or not in the UI the user interacts with. Usability metrics assessed at run time are a useful tool for that purpose too. In order to design plastic user interfaces it is necessary to develop specification techniques for dynamic user interfaces, which will evolve at run time while preserving the same functionality and at the same time preserving usability too. In this paper our solution for the specification of dynamic user interfaces is introduced.

2 Designing User Interfaces

HCI is becoming more and more important, because of the high cost associated to user interface construction for applications, and the high exigencies users demand in terms of usability. Different studies have shown that 48% of an application code is dedicated to user interface development, and that 50% of implementation stage time is dedicated to user interface construction [16].

These facts have motivated the creation of different research projects [6][24][18][12] that face these problems from an automatic user interface generation point of view. These projects try to fill the gap in Software Engineering between functional modelling and user interface development. Among these projects, model based approaches [17] arise as a useful and powerful tool to develop user interfaces. These approaches take as input a requirements specification that is converted into different declarative models. The most widely used ones are the task, the user, the domain, the dialogue and the presentation models. These declarative models are used to generate automatically a user interface compliant with the requirements captured in these models. In next section our model-based methodological approach for user interface generation is introduced briefly.

3 *IDEAS*: A Model-Based Approach for User Interface Design

There are different proposals for model-based user interfaces design; *IDEAS* is one of those proposals. *IDEAS* is a methodology for user interfaces development within the framework of an automatic software production environment. This environment is supported by the object-oriented model *OASIS* [11].

Abstraction is one of the basic principles needed to understand and model the reality. The object-oriented paradigm favours this principle as it conceives the object oriented development process as an iterative and incremental approach that progressively allows a detailed specification of the system to be obtained.

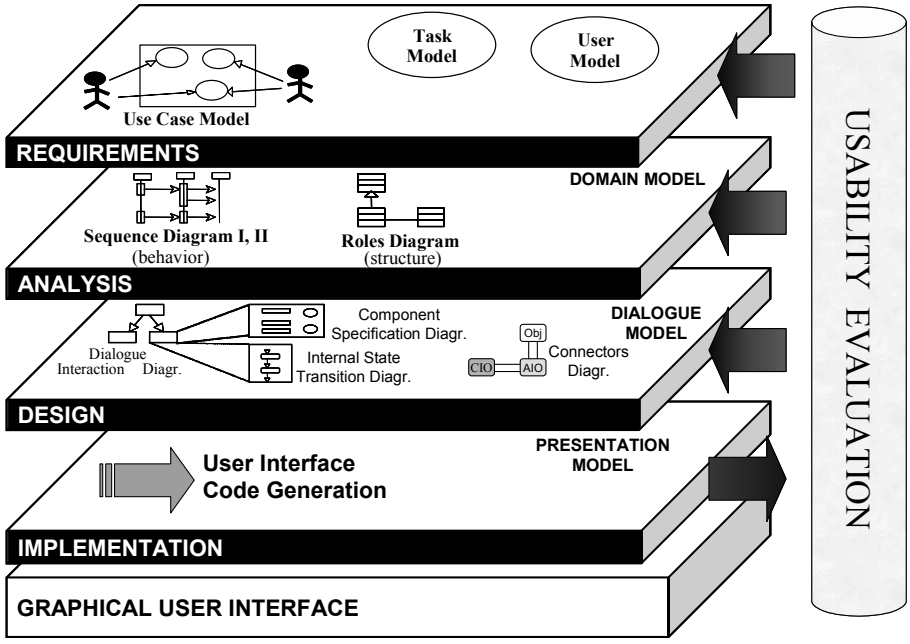

Fig. 2. *IDEAS* methodology stages.

The user interface development process within *IDEAS* is tackled following this principle. This process is not flat, but it is structured in multiple levels and multiple perspectives. The vertical structuring shows the reification processes followed from the first and most abstract level passing through the following levels to finally reach the system implementation, which constitutes the last level. On the other hand, the horizontal structuring shows the different perspectives offered by the different models developed in every one of the vertical levels. Thus, different models are used at the same abstract level to describe the different aspects of the graphical user interface.

Following these ideas, we propose the user interface development process depicted in figure 2. Due to space constraints, we cannot detail the different models proposed, so we will briefly describe the methodological process.

3.1 Requirements Level

At requirements level three models are created: the *Use Case Model*, the *Task Model* and the *User Model*. The *Use Case Model* captures the use cases identified within the information system. Then, for every one of the use cases there will be one or more tasks that the user may perform to accomplish the functionality defined by the use case.

These tasks will be modelled in the *Task Model*. The *Task Model* defines the ordered set of activities and actions the user has to perform to achieve a concrete purpose or goal. We propose a template based on the one proposed by Cockburn [5] to describe in natural language all these issues. The *User Model* describes the characteristics of the different types of users. The purpose of this model is to support the creation of individual and personalized user interfaces.

3.2 Analysis Level

At analysis level the *Domain Model* is generated. This model consists of two diagrams. The first one is the Sequence Diagram, which defines the system behaviour. The second one is the Roles Diagram, which defines the structure of the classes that take part in the associated sequence diagram together with the relationships among these classes, specifying the role of each one of them.

3.3 Design Level

At design level the *Dialogue Model* is developed. All the models that have been generated up to now do not contain any graphical aspect of the final user interface. It is from now on that these issues start to be addressed and the way in which the user-system interaction will be performed is especially important.

The purpose is to describe the syntactic structure of the user-computer interaction. It establishes when the user can invoke commands, select or specify the input data and when the computer can require data from the user or display the output data. These items are modelled by means of Abstract Interaction Objects (AIOs) [3].

3.4 Implementation Level

At implementation level the *Presentation Model* is created. The *Presentation Model* describes the Concrete Interaction Objects (CIOs) composing the final graphical user interface, its design characteristics and visual dependencies among the objects. This model leads to the visualisation of the final graphical user interface according to the final platform style guides. The final graphical user interface generation is performed by using XUL [14] [15], an XML based language, in order to make it as independent as possible from the final platform where the application is going to run. Although XUL language in not as portable as other XML-compliant user definition languages, such as UIML [21], it has a full working runtime environment that renders XUL language directly without translating the code into another language such as HTML, WML or Java, as UIML does. This fact reduces the gap between the XML presentation model generated and the final running code, making easier to modify the running instance of our UI XML model to provide adaptivity according to user's skills and preferences [8], or according to the capabilities of the devices. Nowadays, XUL language can only be run on desktop platforms (running under some different operating systems such as Microsoft Windows, Linux or Mac OS), but we think that this situation may change as long as Embedded Linux [7] could became the standard operating system for new generation mobile devices.

The starting point for generating the graphical user interface in XUL is the *Dialog Model* developed at design level, which, as stated before, models the structure and the behaviour of the graphical user interface by means of AIOs. Therefore, the graphical user interface structure is generated automatically from the Component Specification Diagram created at design level.

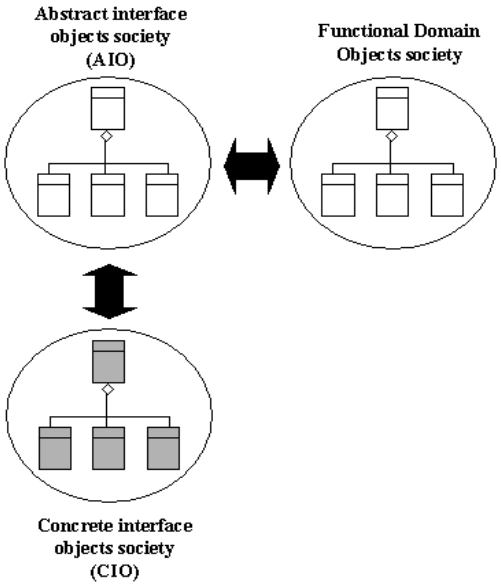

Fig. 3. Object societies involved in user interface operation.

As a result of *IDEAS* methodology applied to an application development three different societies of objects will appear: (1) the functional domain object society, which represents the objects that perform the functionality required in order to achieve the identified tasks, (2) the abstract interaction object society, which includes the objects that represent graphical user interface in an abstract manner, and finally (3) the concrete interaction object society, which will contain the objects that represent the graphical user interface in a specific platform.

Therefore, user interface operation will consist in the interaction between the objects included in the same society (intra-society interaction), the interaction between functional domain objects and abstract interaction objects, and the interaction between abstract interaction objects and concrete interaction objects (intersociety interaction) (see figure 3).

4 About Adaptivity and Plasticity

Adaptivity to user and platform is one of the most exciting challenges in user interface research field. Adaptivity refers to the ability of user interfaces to adapt to different platforms with different capabilities and in different contexts, and to the ability of those interfaces to adapt to each user to meet user skills, preferences or handicaps (accessibility) [21].

As stated earlier, there are different ways to achieve this adaptivity, but one of the main ways to support adaptivity is to design user interfaces with plasticity, this way preserving usability. Usability is the measure of the quality of a user's experience when interacting with a product or system — whether a Web site, a software application, mobile technology, or any user-operated device. This means that the same task will be presented in different ways for different platforms or will even be different for the same platform according to user characteristics; but all of these different presentations of the task, that the user will interact with, should still remain usable (see figure 4).

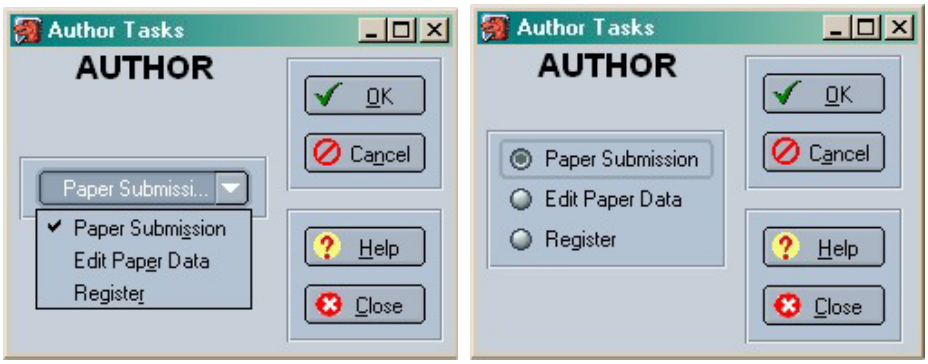

Fig. 4. Different presentations for the same task.

5 Connectors for Adaptive and Plastic UI Specification

A connector [1] consists of a set of roles and the specification of glue to keep them together. Roles model the behaviour for each part involved in interaction. Glue, on the other hand, provides the coordination between instances for each role [25].

Connectors were originally proposed for software architecture specification to provide a mechanism for software components interconnection, and to support reconfiguration in the architecture. To use connectors in the construction process of a specific system, roles will be instantiated. Nevertheless, a component will not be able to instantiate the role if it doesn't comply with the specified service that role should play.

A connector is specified describing: (1) input variables that will be used as input ports, (2) output variables that will be used as output ports, and (3) a set of actions, which will be fired according to a guard condition. Both, variables and actions can be declared as public or private items. Private items are only available to the connector where they have been declared.

Communication between components is achieved in two different ways. On one hand, input and output variables from different components are interconnected, and on the other hand methods from several components may be synchronised (see figure 5).

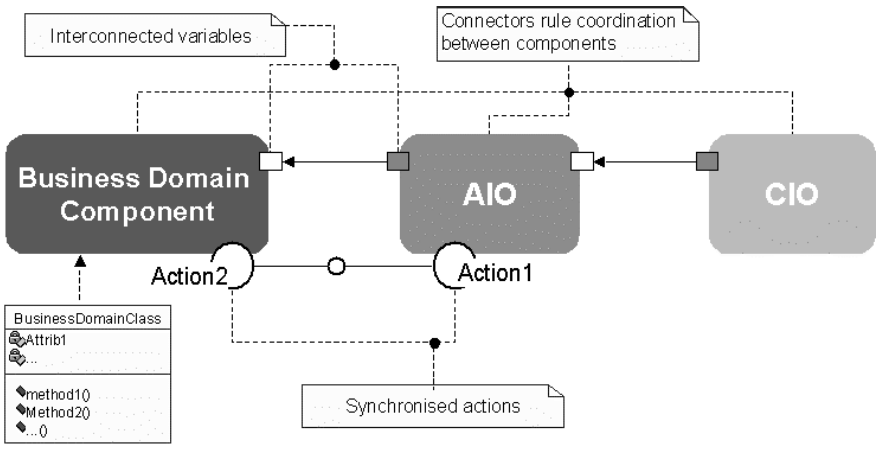

Fig. 5. Connectors in *IDEAS* methodology.

5.1 Connectors in User Interface Design

When applying connectors to our object societies (Functional domain object society for business model, Abstract interaction object society for the abstract user interface and Concrete interaction object society for the actual user interface displayed) we will need to encapsulate interacting objects within component interfaces, interconnected using the connector paradigm [10]. All these three societies are composed of a structural (static) part (Σ_S) and their behaviour (Σ_B).

There will be two different kinds of communication between components [8] in our object societies at runtime. On one hand, there will be communication between functional domain objects requesting basically input/output operations, and on the other hand there will be a communication between user interface abstract and concrete components. A mapping process is needed between processing objects (requesting or providing data) and abstract interaction objects that will interpret the requests from functional domain objects. Then abstract interaction objects will map these requests in a one-to-many relationship with concrete objects (see figure 6).

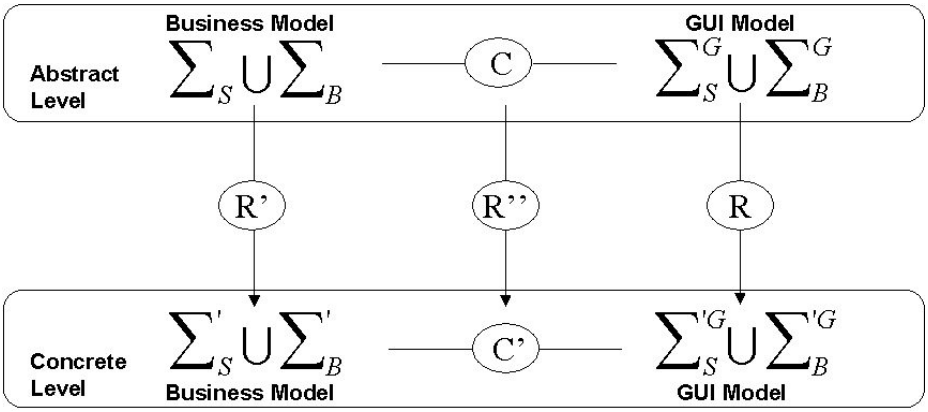

Fig. 6. Mappings between abstract level and concrete (base) level.

This component interface for communication between objects in Abstract interaction object society and Concrete interaction object society should be based upon World Wide Web Consortium Document Object Model (DOM). DOM provides a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents [23]. DOM has been designed to handle HTML and other user interfaces specified using XML-based languages (for example XUL language). Using DOM model an interface created using XUL can be updated at runtime to reflect user preferences and even to introduce mixed-initiative [9] user interfaces where software agents [26] are engaged in a collaborative process to complete the tasks by means of DOM model to manipulate user interface on behalf of the user.

5.2 A Formal Approach to Connector Design

The name "Document Object Model" was chosen because it is an "*object model*" in the object oriented design sense: documents are modelled using objects, and the model encompasses not only the structure of a document, but also the behaviour of a document and the objects of which it is composed. As stated before, in our model-based methodology, *OASIS* specification language is used. *OASIS* is a formal approach for conceptual model specification following the object-oriented approach.

The visibility between objects is determined by an interfacing mechanism. In *OASIS* every object encapsulates its own state and behaviour rules. As usual in object-oriented environments, objects can be seen from two points of view; static and dynamic. From the static perspective the attributes are the set of properties describing the object structure. The object state in a definite instant is the set of structural property values. From the dynamic perspective the evolution of objects is characterized by the "change of state" notion. Making use of these features of *OASIS*, connectors are specified expressing DOM properties, events and objects using this formalism.

6 An Illustrative Example

To illustrate how to use connectors to specify user interfaces in a flexible way enough to support an adaptive behaviour, an excerpt of a study case based on [20] is provided as an example. This study case we have chosen models a typical conference review system. For the sake of simplicity, we will focus on the task when the accepted papers are chosen according to the reviews provided by reviewers.

To specify how AIOs, CIOs and functional domain objects interact (our three object societies) we will need to create the connectors to model which information is sent when interacting, and the actions and its guard conditions for each task that the connector can carry out. In figure 7 the AIOs involved in the task modelled in the example are shown. Notice that when selecting final accepted papers two classes from functional domain will be accessed: (1) *Paper*, (2) *Review*.

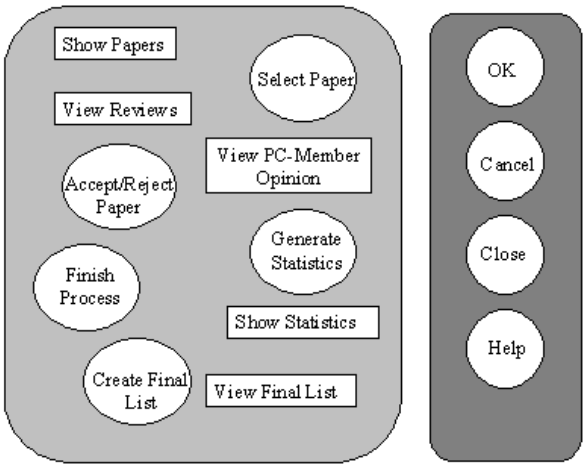

Fig. 7. Abstract Interaction Objects (AIO) involved in our case study.

Taking into account all the models built throughout the methodology stages described at section 3, connectors are specified to complete a connector diagram

(figure 9). In this connector diagram the interaction between the user interface components is described graphically.

In this diagram there are three different elements: (1) components (AIOs, CIOs, and functional domain objects), (2) input/output variables, and (3) actions. Components are depicted using a box with rounded edges. Input variables are depicted using small white boxes, while output variables are depicted using small black boxes. Actions are represented using a white circle and a label (see figure 5). There are two types of actions: synchronized and not synchronized. The actions, which are connected to each other, are synchronized, so whenever one is executed the other one will be executed too (always checking guard conditions before). Input and output variables are connected too. When an output variable is changed, automatically the input variable it is connected to will reflect those changes and guard conditions for actions will be checked to find out whether they should be fired or not.

In figure 8 you can see a possible presentation for the task modelled using connector paradigm. In this example we have used XUL language for the final concrete user interface implementation.

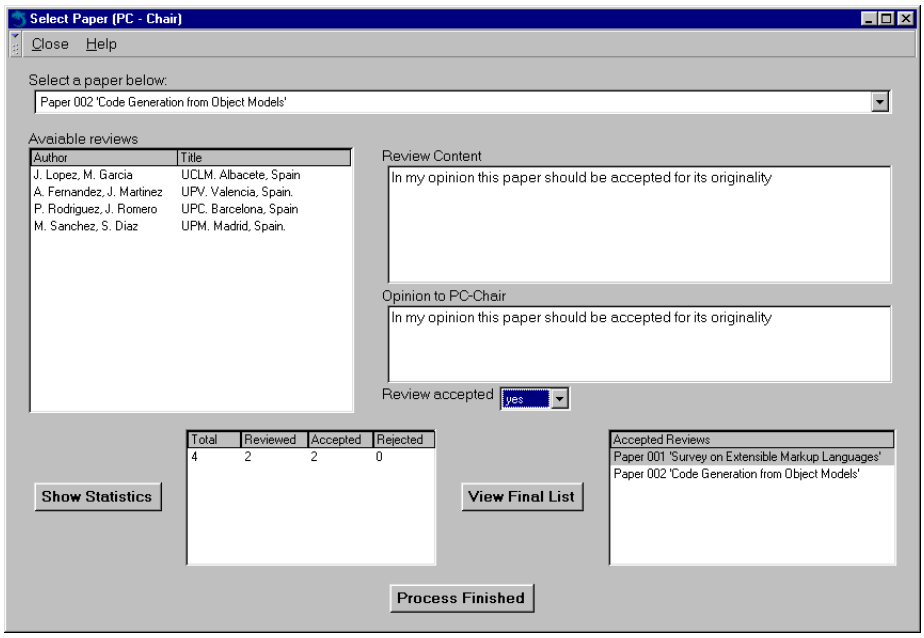

Fig. 8. A concrete presentation for the specified example task.

7 Conclusions and Future Work

Although model-based approaches have been a research field for more than a decade, it is now when the new interaction paradigms are requesting new design techniques

for these new paradigms. User interaction today demands adaptive, and therefore plastic, user interfaces.

However, plastic interfaces will need to preserve the functionality they were designed for while meeting user characteristics and different platforms capabilities. To design these plastic interfaces the specification of these user interfaces should support plasticity at runtime. In this paper a first approach has been proposed to support this plasticity features from a formal point of view.

Nevertheless, still much work is left to complete the mapping of connectors from an abstract design to a concrete user interface at runtime. This mapping would allow an automatic adaptive user interface generation from a model-based point of view.

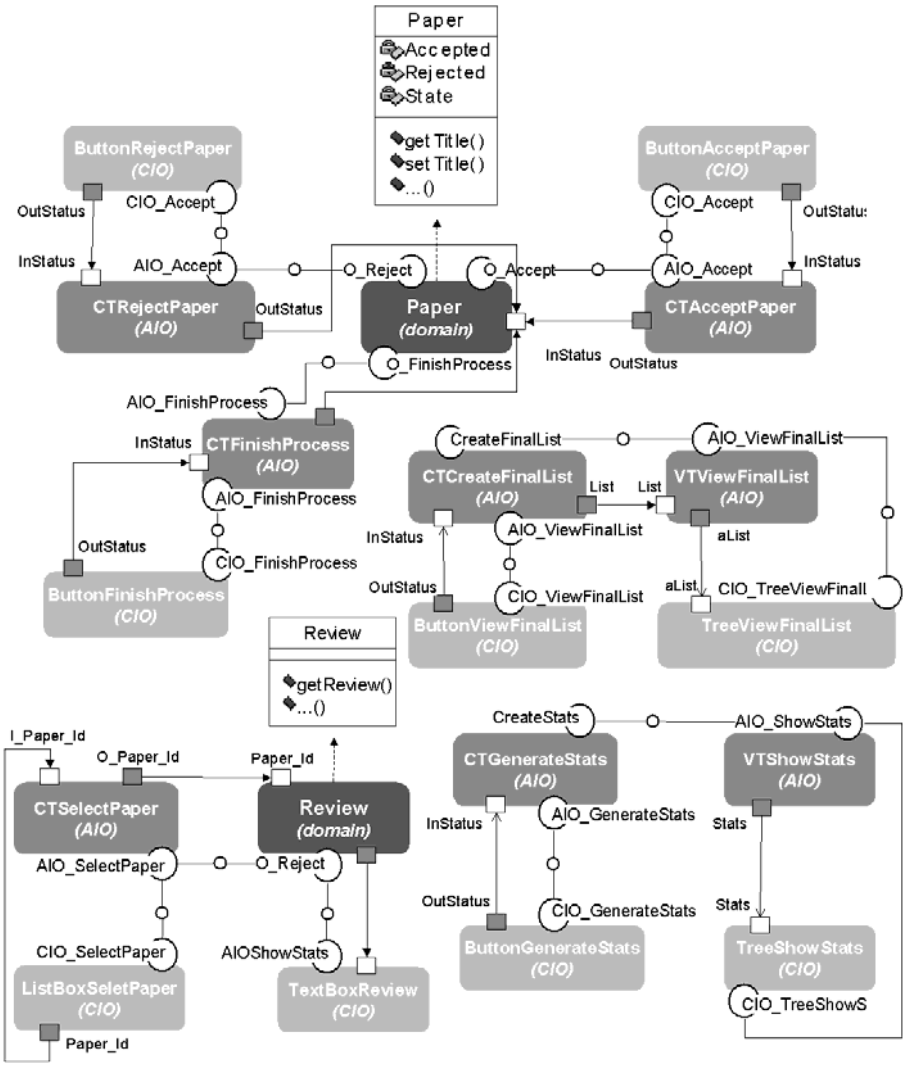

Fig. 9. Connectors diagram for study case.

Acknowledgements

This work is supported in part by the Spanish CICYT TIC 2000-1673-C06-06 and CICYT TIC 2000-1106-C02-02 grants.

References

1. Allen, R., Garlan, D. A Formal Basis for Architectural Connectors, *ACM TOSEM*, 6(3), pp. 213-249, July, 1997.
2. Benyon D., Murray D.. Developing adaptive systems to fit individual aptitudes. Proceedings of the 1st international conference on Intelligent User Interfaces, pp. 115-121, Orlando, Florida, United States, ACM Press, 1993.
3. Bodart, F. ,Vanderdonckt, J. On the Problem of Selecting Interaction Objects, *Proc. of HCI'94*, Cambridge University Press, Cambridge, pp. 163–178, 1994.
4. Calvary, G., Coutaz, J., & Thevenin, D. A Unifying Reference Framework for the Development of Plastic User Interfaces. In Proceedings of IFIP WG2.7 (13.2) Working Conference EHCI'2001 (Toronto, May 2001), M. Reed Little & L. Nigay (Eds.), Springer Verlag Publ.,LNCS 2254, pp.173-192.
5. Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley, 2001.
6. Elwert, T., Schlungbaum, E. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In: *Designing, Specification and Verification of Interactive Systems.* Wien: Springer, pp. 193-208, 1995.
7. Embedded Linux. http://www.linuxdevices.com.
8. Fernández-Caballero, A., López-Jaquero, V., Montero, F. and González, P. Adaptive Interaction Multi-agent Systems in E-learning/E-teaching on the Web. In Web Engineering: Proc. of International Conference, ICWE 2003 (Oviedo, Spain). J.M. Cueva Lovelle, B.M. González Rodríguez, L. Joyanes Aguilar, J.E. Labra Gayo, M. del Puerto Paule Ruiz (Eds.). Springer Verlag, LNCS 2722, p. 144-154.
9. Horvitz, E., Principles of Mixed-Initiative User Interfaces. Proc. ACM SIGCHI Conf. Human Factors in Computing Systems, ACM press, New York, pp. 159-166, 1999.
10. López-Jaquero, V., Montero, F., Fernández, A., Lozano, M. *Towards Adaptive User Interface Generation: One Step Closer To People*. 5th International Conference on Enterprise Information Systems, ICEIS 2003. Angers, France. April 23-26, 2003.
11. Letelier, P., Ramos, I., Sánchez, P., Pastor, O. OASIS version 3: A Formal Approach for Object Oriented Conceptual Modeling. SPUPV-98.4011. Edited by Universidad Politécnica de Valencia, Spain, 1998.
12. Lozano, M., Ramos, I., González, P. User Interface Specification and Modeling in an Object Oriented Environment for Automatic Software Development. IEEE 34th International Conference on Technology of Object-Oriented Languages and Systems, pp. 373-381, 2000.
13. Montero, F., Lozano, M., González, P. and Ramos, I. A first approach to design web sites by using patterns. In Proceedings of the First Nordic Conference On Pattern Languages of Programs. VikingPLoP. Hojstrupgard. 2002. pp. 137-158. ISBN: 87-7849-769-8.
14. Mozilla Project. http://www.mozilla.org, 2003.
15. Oeschger, I., Murphy, E., King, B., Collins, P., Boswell, D. Creating Applications With Mozilla. O'Reilly, September, 2002.

16. Myers, B. A., Rosson, M. B. Survey on User Interface Programming. In Striking a Balance. *Proceedings CHI'92*. Monterey, May 1992, New York: ACM Press, 195-202, 1992.
17. Paternò, F.. Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, 2000.
18. Puerta, A.R. A Model-Based Interface Development Environment. IEEE Software, pp. 40-47, 1997.
19. Rettin, M.. Designing for Small Screens. AMC SigWeb, Chicago, 2002. http://www.marcrettig.com/writings/rettig,SmallScreens.pdf
20. Schwabe, D. A Conference Review System. First Workshop on Web-Oriented Software Technology. Valencia, June 2001. http://www.dsic.upv.es/~west2001
21. UIML. http://www.uiml.org, 2003.
22. W3C. WAI. http://www.w3.org/WAI/, 2003.
23. W3C. DOM. http://www.w3.org/DOM/, 2003.
24. Vanderdonckt, J., Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In ACM Proc. of the Conf. On Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel & T. White (Eds.), ACM Press, New York, 1993, pp. 424-429.
25. Wermelinger, M., Lopes, A., Fiadeiro, J.L. Superposing connectors, *in Proc. 10h International Workshop on Software Specification and Design*, IEEE Computer Society Press, pp. 87-94, 2000.
26. Wooldridge, M., Jennings, N.R. Agent Theories, Architectures, and Languages: A Survey, Proc. ECAI-Workshop on Agent Theories, Architectures and Languages (eds. M.J. Wooldridge and N.R. Jennings), Amsterdam, The Netherlands, pp. 1-32, 1994.