Model-based Reverse Engineering of Legacy Applications User Interfaces

Francisco Montero, Víctor López-Jaquero, Pascual González
LoUISE Research Group, I3A, University of Castilla-La Mancha
Campus, s/n, 02071 (SPAIN)
+34 967 59 92 00 - {fmontero, victor, pgonzalez}@dsi.uclm.es

ABSTRACT
User Interface Description Languages (UIDL) are languages used in Human-Computer Interaction (HCI) in order to describe User Interfaces (UI) independently of any implementation technology. These languages can be used effectively and efficiently when proper software is available. In this paper a suite of software tools is introduced. This suite of tools is useful for supporting Reverse Engineering activities of UIs. Prototyping, evaluation and specification of user interfaces are enabled by using our suite of tools: GuiLayout++, PureXML, reTaskXML, and AcauiXML.

Author Keywords
Model-based, user interfaces, legacy applications, reverse engineering, UIDL.

General Terms
Measurement, Design, Human Factors, Languages.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces, Graphical User Interfaces (GUI).

INTRODUCTION

Engineering is the process of designing, manufacturing, assembling and maintaining products and systems. There are two types of engineering, forward engineering and reverse engineering. Forward engineering is the traditional process of moving from high-level abstractions and logical designs to the final implementation of a system. The process of duplicating an existing part, subassembly, or product, without drawings, documentation, or a computer model is known as reverse engineering.

Reverse engineering is now widely used in numerous applications, such as manufacturing, industrial design and jewelry design and reproduction. For example, when a new car is launched on the market, competing manufacturers may buy one and disassemble it to learn how it was built and how it works. In software engineering, good source code is often a variation of other good source code.

In this last scenario, in software development, designers give shape to their ideas by using high and low prototypes by using paper and pencil, interactive paper prototypes, programmed façades, and prototype-oriented languages, but a set of models are needed to develop a final application. The fidelity of the prototypes previously described ranges from low to high. Does fidelity affect a prototype's usefulness as a testing tool? Most usability experts agree that low-fidelity prototypes are very useful in the early stages of design.

Reverse engineering provides a solution to this problem, because the final user interface model may be the source of information for other models, for instance, concrete user interface, abstract user interface or another. All those models are considered in traditional model-based and model-driven user interface development environments.

In this paper reverse engineering, prototyping and model-based user interface development techniques are used jointly in order to provide facilities for user interface development of legacy application. Following are some of the reasons to use reverse engineering in user interface development:

- The original source code and developers no longer exists.
- The original developers of an application no longer maintain the product, and the software may become obsolete.
- The software application design documentation may be lost or never existed.
- We want to eliminate or modify some bad features of an application.
- Exploring new alternatives to improve software performance and features.
- Documentation

This paper is organized as follows. Section 2 describes and provides an overview of related works with Reverse Engineering. Section 3 introduces the main contribution of this paper, a software tool for supporting reverse engineering by using prototyping and user interface description languages. With this tool presentation and navigation models associated with a model-based user interface development environment can be achieved. Finally, a section related to discussion and conclusions is included.

RELATED WORK

Reverse Engineering (RE) has been an active research topic for some years now. RE has been applied to many different fields. Regarding Computer Science, it has been applied to both hardware and software systems. First, to focus our work, the term RE will be discussed. The most widely-known taxonomy for RE is [2]. In this work the authors present a taxonomy that illustrates the different paths that software development can follow. Reverse engineering is the process of analyzing a subject system to: (1) Identify the system's components and the interrelationships and (2) Create representations of the system in another form or at a higher level of abstraction. Thus, RE is supposed to happen from implementation to design and from design to requirements. Furthermore, RE can also happen at the same development stage. In this case, some different operations are identified: *Restructuring, Reengineering*, *Redocumentation* and *Design Recovery*. *Restructuring* is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics).

It does not involve modifications because of new requirements. On the other hand, *Redocumentation* is the creation or the revision of a semantically equivalent representation within the same relative level of abstraction. Such as the pretty printers for code. *Reengineering* is the examination and alteration of the subject system to reconstitute it in a new form and the subsequent implementation of the new form. It generally includes some form of reverse engineering followed by some form of forward engineering or restructuring. Finally, in *Design Recovery* domain knowledge, external information, and deduction or fuzzy reasoning are used to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself. In [4] a multi-path development process for user interfaces was introduced. In this process, user interface design could begin at different entry points in the process, not necessarily staring from the first or last stages. RE was also considered in this approach.

When working with legacy systems user interfaces all three RE operations are very useful. *i.e.Restructuring* could be used to port the user interface from one target platform to another. This issue was addressed in VAQUITA [1]. In VAQUITA HTML contents were reverse engineered to be ported to another target platform. The main limitation of VAQUITA was its dependency on HTML. An on-line version of VAQUITA, namely ReversiXML, is also available [10]. WebRevEnge [8] also supports HTML Reverse Engineering. Nevertheless, WebRevEnge obtains the corresponding CTT task model. User interface *Reengineering* goes one step beyond, as it considers making changes in the functionality. An example would be using one of the tools aforementioned supporting *restructuring*, i.e., VAQUITA, and then add some extra functionalities before porting to another target platform.

Obviously, to support actual RE, more than the user interface should be reversed engineered. This includes functional core and databases. i.e., it has been applied to migrate legacy databases [7].

Model driven techniques applied to user interface design look promising for RE, because of the clear separation of concerns that it exhibits. When applying RE the different facets of an application must be processed. Having a clear separation of concerns can help in this process. Furthermore, in the goals of these model-driven approaches for user interface design *restructuring* and *reengineering* are two purposes usually considered. Starting from a set of models, the aim is generating the user interface for different target platforms, supporting *restructuring* to accommodate the user interface to the peculiarities of each platform (an in general each context of use). The original set of models could be modified afterwards, for instance, to remove obsolete functionalities.

The creation of the models required in any model-driven approach is an interesting issue. A visual syntax can greatly improve the usability of a notation. Directly creating the models, i.e. by using an XML syntax, can be error prone and slow. Different tools have been used so far for visually designing these models, including *IdealXML*[5], *SketchiXML* [3] or CTTE (http://giove.isti.cnr.it/ctte.html). *IdealXML* support the automatic generation of abstract user interfaces out of task models, enabling the user the rapidly create user interface prototypes. This idea was combined with some automatic evaluation techniques in *GuiLayout++* [6]. In this tool, some RE support was already included. The designer loads a screenshot of the user interface, and then he draws on it the different widgets and areas identified in the screenshot. Finally, these elements were used to automatically generate an abstract user interface.

In this work we use an approach similar to the one we used on *GuiLayout++*, but in this case we aim at reverse engineering the user interface to generate the concrete user interface, including both the widgets, layouts and navigation.

REVERSE ENGINEERING USER INTERFACE FOR LEGACY APPLICATION

Aiming at supporting RE for user interface design a suite of tools has been developed. These tools support the reverse transformation between the usual models used in user interface development.

Currently, these tools work individually, and they interchange the models they used by means of an XML specification expressed in UsiXML. Nevertheless, we are working in a unified tool to make easier their use. In Fig. 1 the tools developed and the models they manipulate are illustrated.

Fig. 1 shows the main models linked to user interface design. Reading the figure from left to right a forward development of user interfaces can be observed (forward engineering). On the other hand, reading the figure from right to left reverse engineering can be observed.

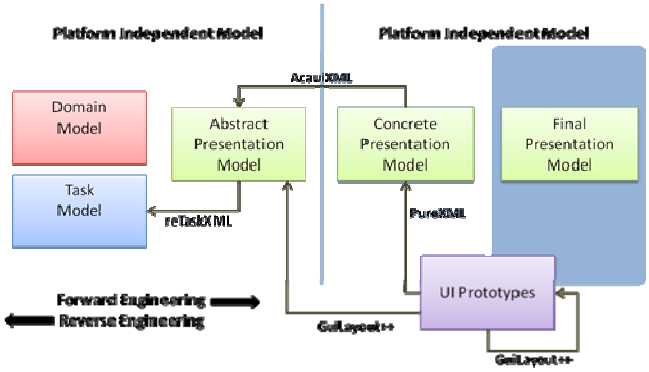


Figure 1. Suite of tools and reverse engineering of UI process.

The suite of tools currently developed is:

- *reTaskXML*: is tool that tackles reverse engineering an abstract specification of a user interface to obtain the corresponding task model (see Fig. 2).
- *AcauiXML*: is a tool that supports reverse engineering a concrete user interface model to obtain a corresponding abstract user interface. UsiXML syntax is used for this purpose (see Fig. 3).
- *PureXML*: is a tool that supports loading user interface screenshots, from a user interface design or an existing application, and creating prototypes for those screenshots. These prototypes are then used to obtain a concrete user interface expressed in terms of UsiXML (see Fig. 5).
- *GuiLayout++* [6]: is a tool that supports the creation and evaluation of user interface prototypes (both low and high fidelity). The prototypes created are then used to generate the corresponding abstract user interface specification in UsiXML language (see Fig. 4).

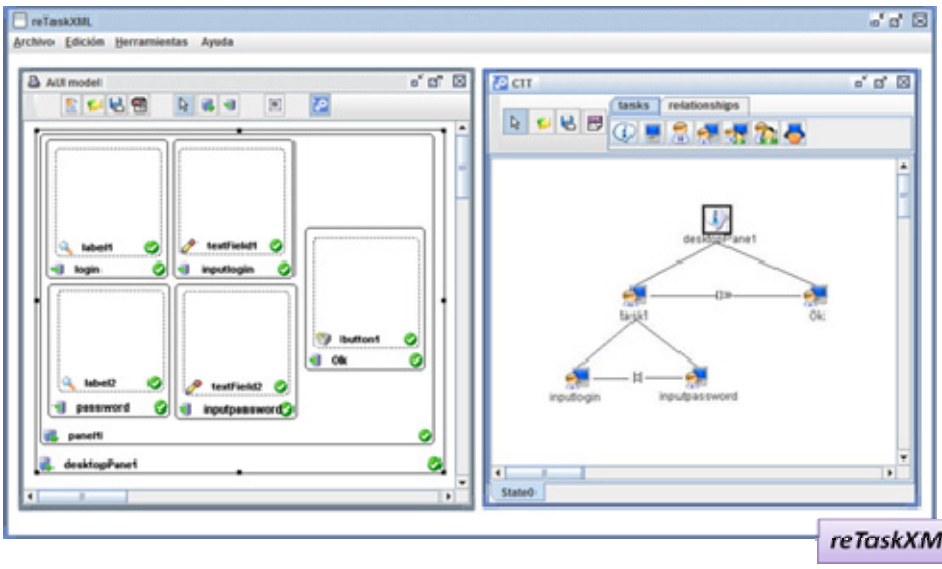


Figure 2. reTaskXML: from abstract user interface to task model.

The reverse engineering process

As aforementioned, different situations can arise promoting the use of reverse engineering techniques. Legacy applications are one of the most usual situations when one would use reverse engineering. i.e., the designer would like to move the user interface of a legacy application from a character-based screen to a graphical one, or we could require retargeting the legacy application from standard desktop PC to the web.

This process can be supported by our suite of tools to help the developer in the reverse engineering process.

In Fig. 1, the process is depicted. In the beginning the developer has only a screenshot of the legacy application. This screenshot can be loaded into *GuiLayout++*, where the designer can assess some metrics in the screenshot. i.e., the uniformity in the alignment in the layout of the components or the percentage devoted to each task can be analyzed. The different areas used for each purpose and its layout are specified by drawing boxes of the corresponding type. The different types of boxes are marked in different colors (see Fig. 4a). Then, the designer can change the screenshot to address the issues detected by *GuiLayout++*. An example of one of the metrics computed by *GuiLayout++* is shown in Fig. 4b. The graph in this figure displays the results for the metric to assess the amount of space devoted to each purpose in a web page. This is one of the metrics proposed in [11]. Alternatively, the designer can directly load a screenshot in *PureXML* (see Fig. 5).

In *PureXML*, the designer loads a screenshot. In the example used in this paper, a well-known printing dialogue is used. Then the designer used the widget palettes provided (see the right part in Fig. 5) to add widgets. These widgets are organized in tabs, so they can be more easily found. Their *Look&Feel* takes inspiration from Balsamiq [9]. The designer uses *Drag&Drop* to drag the corresponding widget into the screenshot, and then he adjusts its size to the real one in the screenshot. i.e., when the designer finds a dropdown list in the screenshot, a dropdown list is dragged from the *List,Trees and Tables* tab of the widget palette, and then its size is adjusted. The designer proceeds for all widgets in the screenshot. Notice how a tree is created simultaneously to represent the widgets added to the screenshot (see the top left part in Fig. 5). One important issue that PureXML addresses is navigation. A user interface is usually composed of more than one window, and therefore more than one screenshot.

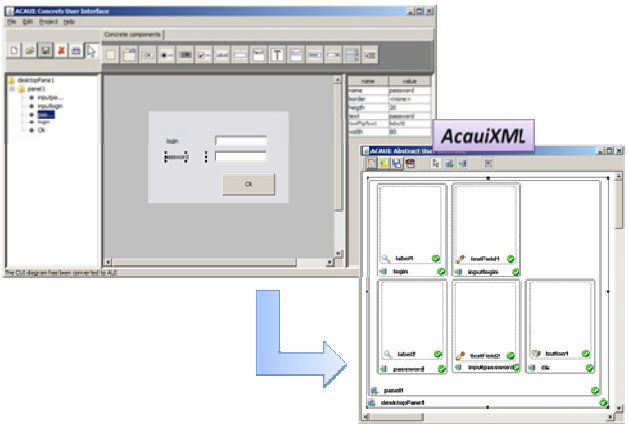


Figure 3. AcauiXML: from concrete user interface to abstract user interface.

Thus, in PureXML the designer can load many screenshots. They are shown in the same place as the tree in the left part of the window. The designer can toggle both views by using the tabs *Design* and *Navigation*.

In the example in the bottom left part there is a window where the designer has modeled a transition from *Properties* button to the *Properties* dialogue window. In the properties of the transition the designer can specified the events that trigger the transition. Thus, the designer is supported in modeling which widgets the user interface is composed of, and how the user navigates from one window to another.

By selecting any widget added, the designer can specify some properties of the widget. The set of properties supported is taken from the current UsiXML 2.0 concrete user interface draft. Thus, the designer can specify attributes such as, the fonts, the colors, etc. All the project can be saved in UsiXML 2.0 concrete user interface XML syntax. This output can be then used in AcauiXML.

AcauiXML takes a concrete user interface specification and it obtains an abstract user interface (see Fig. 3). The visual syntax currently used is the same one we designed for *IdealXML*.

Finally, the abstract user interface generated by AcauiXML can be used to obtain a task model. The task model is a perfect starting point for reengineering an application, and starting from the task a forward engineering process to produce a new application out of the starting legacy one. In this task model unused features could be removed, and some extra features could be added.

DISCUSSION AND CONCLUSION

Currently, user interface description languages present many challenges and difficulties, among these challenges are the following: effective, efficiency and satisfactory software tools for supporting prototyping, evaluation and developing. But, many times developing must be done following a reverse path.

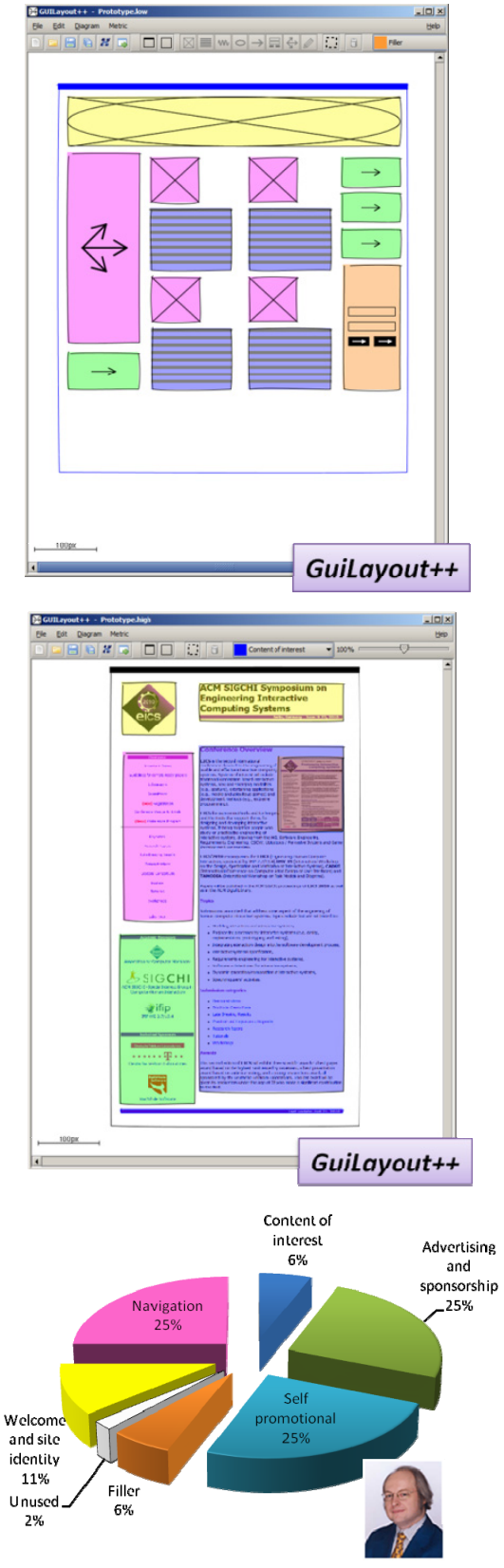
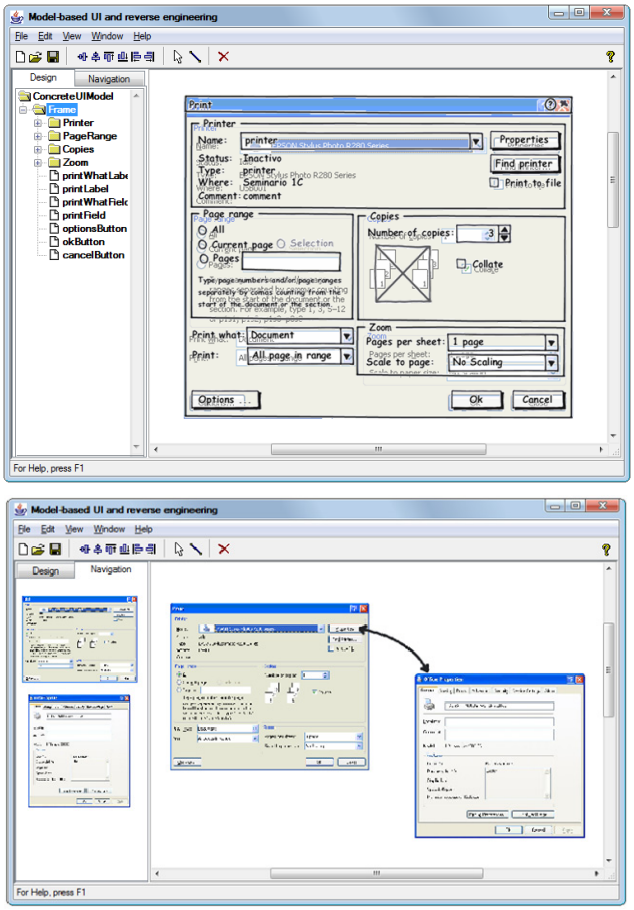


Figure 4. GuiLayout++: prototyping and assessing the user interface: (a) Prototyping, (b) Assessing.



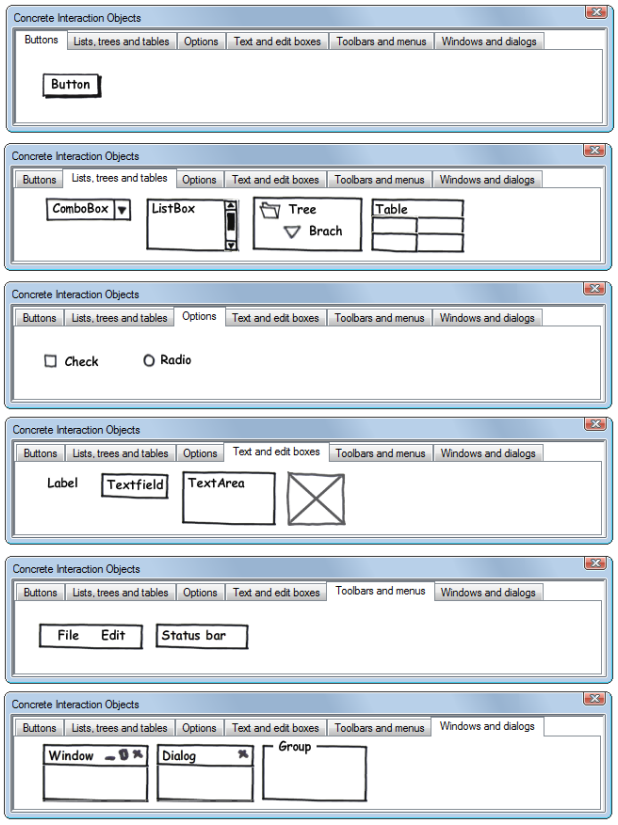


Figure 5. PureXML: from final user interface to concrete user interface: (a) The editor window tabs (Design and Navigation), (b) The tabs for the UI component palette.

In this paper a suite of tools is introduced. These tools are related to evaluation and reverse engineering. Prototyping is used in some of these software tools also. We identified a good joint-venture in the combination of prototyping and UI description languages. Prototyping is a very good common ground to communicate stakeholders involved in a development, between them, for instance, designers, final users, clients and developers.

Moreover, prototyping and evaluation are key elements in any user-centered design technique. *GuiLayout++* is useful in this scope, we can specify applications and we can evaluate what space distribution is used. In a similar scenario, *PureXML* is introduced. By using *PureXML*, concrete specifications of user interfaces can be built.

We identified an agile tendency in the development of user interfaces and software products in general. Under this point of view, several activities in the development process can be done simultaneously. Prototyping, specification and evaluation activities should be supported by available tools. In this paper a suite of tools with this premise in mind was introduced.

Additional improvements in our suite of tools can be considered. Previous software tools were done independently and integration efforts are considered in this moment.

A previous version of UsiXML was used for developing some previous software tools, namely 1.8, but a new release and review of this language will be made soon officially available and modifications and updates will be done. Therefore, some of our tools require some updating efforts

Reverse engineering is an interesting path in the specification, modification and optional development of software tools. In this sense, additional efforts will be done in the forward engineering direction. That is, suite of tools introduced in this paper will be integrated with *idealXML*[5] in order to provide a new version from previously non-documented or available software products.

ACKNOWLEDGMENTS

This research has been partially funded by Spanish Ministry of Science and Innovation grant TIN2008-06596-C02-01 and the grant PEII09-0054-9581 from the Regional Government of Castilla-La Mancha. We would like to thank also Miguel Oliver, Abraham Martínez y Francisco Javier Muñoz for collaborating in the implementation of the tools.

REFERENCES

1. Bouillon, L., and Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms with VAQUITA. In *Proc. of IEEE Working Conf. on Reverse Engineering WCRE'2002* (Richmond, 28 October-1 November 2002). A. van Deursen, E. Burd (Eds.). IEEE Computer Society Press, Los Alamitos (2002), pp. 339-348.
2. Chikofsky, E.J., and Cross, J.H., II. Reverse engineering and design recovery: a taxonomy. IEEE Software 7, 1 (Jan. 1990), pp. 13-17
3. Coyette, A., Kieffer, S., and Vanderdonckt, J. Multi-fidelity Prototyping of User Interfaces. In *Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2007* (Rio de Janeiro, 10-14 September 2007), Lecture Notes in Computer Science, vol. 4662, Springer-Verlag, 2007, pp. 149-162.
4. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, vol. 3425. Springer, Berlin (2004), pp. 200-220.
5. Montero, F., López Jaquero, V. IdealXML: An Interaction Design Tool and a Task-Based Approach to User Interface Design. 6th International Conference on Computer-Aided Design of User Interfaces (CADUI 2006), Bucharest, Romania, 6-8, June, 2006.
6. Montero, F., López-Jaquero, V. Guilayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation. Proceedings of the 1st Workshop on USer Interface eXtensible Markup Language Workshop UsiXML'2010 (Berlin, June 20, 2010). Thalès, Paris, pp. 39-44.
7. Pérez, J. Anaya, V., Cubel, J.M., Domínguez, F., Boronat, A., Ramos, I., Carsí, J.A. Data Reverse Engineering of Legacy Databases to Object Oriented Conceptual Schemas, Software Evolution Through Transformations: Towards Uniform Support throughout the Software Life-Cycle Workshop (SET'02), Barcelona (Spain), 2002.
8. WebRevEnge - Tool for Reverse Engineering: From HTML to CTT Task Models. http://giove.isti.cnr.it/tools/WebRevEnge/home
9. Balsamiq. Balsamiq mockups. http://balsamiq.com/
10. Bouillon, L., Limbourg, Q., Vanderdonckt, J., and Michotte, B., Reverse Engineering of Web Pages based on Derivations and Transformations. In *Proc. of 3rd Latin American Web Congress LA-Web'2005* (Buenos Aires, October 31-November 2, 2005), IEEE Computer Society Press, Los Alamitos, 2005, pp. 3-13.
11. Nielsen, J., Tahir, M. Homepage Usability: 50 Websites Deconstructed, 2001.