Flexible Re-engineering of Web Sites

Laurent Bouillon, Jean Vanderdonckt, Kwok Chieu Chow

Université catholique de Louvain, IAG, Information Systems Unit
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
{bouillon,vanderdonckt}@isys.ucl.ac.be kwokchieu_chow@skynet.be

Abstract

Re-engineering transforms a final user interface into a logical representation that is manipulable enough to allow forward engineering to port a UI from one computing platform to another with maximum flexibility and minimal effort. Re-engineering is used to adapt a UI to another context. This adaptation is governed by two main tasks: the adaptation of the code itself to the new computing plating platform and the redesign of the UI to better suit the new constraints of the target platform (interaction capabilities, screen size,…). To support this process, we have developed a reverse engineering tool that allows a flexible recovery of the presentation model from Web sites, adapting the reverse engineering to the target platforms, and a forward engineering tool that converts this presentation model into any final executable UI, in particular expressed in VRML, WML, …

1. Introduction

Portability of a User Interface (UI) generally refers to the capability of a UI to be ported from one computing platform to another with an effort that remains minimal. Galaxy (www.ambiencia.com) enables developers to design a UI on one platform, say MS Windows, and to export it without any change to other platforms, e.g., Mac OS X and Linux. The individual UIs then adhere to the Look and Feel that is proprietary to the respective platforms. The development of a UI does not frequently envisage the future desire for portability. Consequently, when the need arises to port a UI from one platform to another, it is rather difficult to support this porting. Developers do not necessarily want to start again from scratch to design a UI for a new platform since a UI already exists that could be a potential source of inspiration, if not a starting point. In this second case, transcoding tools automatically transform a UI code from the original platform to a new UI code for the target platform. This transformation can occur at design-time (i.e., the transformation is made only once and re-inserted in the formation is made only once and re-inserted in the new platform) or at run-time (i.e., the transformation is performed on demand when the UI is requested). Any HTML page can be transformed into a WML deck of cards on the fly when the mobile phone user accesses a web page [9].

Portability and transcoding tools suffer from shortcomings: the former only produce the same UI layout for all platforms while the latter only apply code to code transformations that are peculiar to any couple (source platform, target platform). Both remain inflexible (no design alternatives), uncontrolled (no human intervention to fine tune the transformation), and very specific (hard to generalize to other couples). They do not necessarily consider constraints imposed by the target platform such as: operating system, programming language, screen resolution, interaction capabilities. To overcome these shortcomings and to address the needs of UI portability, we argue that a UI reverse engineering process can be combined with UI forward engineering process to produce not only more usable UIs in a logical way, but also to benefit from the reverse engineering to port a UI to any other target platform.

The remainder of this paper is structured as follows: the second section provides related work on the reverse and forward engineering of UIs. The third section is about Cameleon's reference framework in which our research takes place. The fourth section is devoted to the definitions of some re-engineering terms. The fifth presents our tools for the reverse engineering of HTML and the forward engineering in VRML. The sixth section contains a case study showing the possibilities of these tools, and the last section concludes this paper.

2. Related Work

Preliminary work has already been conducted in the field of UI reengineering, especially to migrate text-based UIs (TUI) to graphical UIs. Another trend in the re-engineering of UI is the migration of the UI to a platform that uses another modality. Most of the tools allowing re-

verse engineering by constructing an abstract representation of the system are described in this section.

MORPH [11] [12] identifies basic user interaction tasks in legacy code (TUI) by applying static program analysis techniques, including control flow analysis, data flow analysis, and pattern matching. The resulting model is then used to transform the detected abstractions in a graphical environment from a specific widget toolkit. The original code is then modified to take into account the new dialogue structure of GUIs.

The PIMA Project [8] [18] aims at producing applications that are device independent. A Platform Independent Application can be created either by a design tool or by abstracting a concrete UI thanks to the generalization process. Generalization is done by reverse engineering the code of the UI. This process starts with the detection of interaction elements. Secondly, the properties and semantic information of these elements can be inferred. A specialized engine with a device profile then creates another application specialized for a particular device. The major difference with our approach is that the user has more control over the reverse engineering and the translation step. Another important difference is that the output of our tool can be reused by other softwares, as the abstract language –XIML- is publicly available (under a licence).

The reengineering process in tamex[19] allows one to produce HTML UIs composed of data contained in several other Web pages. The approach followed by Tamex is based on the concept of task-specific mediation: information sources within an application domain are encapsulated within wrapper agents (data extraction) which interact with an intelligent intermediary agent, the mediator (aggregate data). XML is used as an intermediate data structure for information exchange and as a modeling language for the mediator's domain ontology and task structure. The information extraction is done with an XPath-based algorithm for generating extraction rules from HTML.

The same authors developed another tool [7], which analyzes interaction traces from a legacy system in a single screen or between several screens in order to regenerate new user-centered tasks.

Reweb's reengineering process [18] restructures Web applications in order to avoid their inevitable degradation. It uses a set of transformation rules aiming at the improvement of maintainability, usability and portability. It also restructures the design thanks to a web application model and by incorporating frame-based navigation.

In [10], the reengineering of legacy system is based on the extraction of the user interface descriptions from COBOL/CICS source code and their translation to abstract behaviour descriptions based on process algebra (CCS). A part of the analysis is done using Refine/COBOL. The translation itself is done manually by remodeling the behavior implied by the existing interface into a series of abstractions that have meaning in the context of a GUI.

UIML [1] allows the developer to specify the presentation and dialog components of a UI in a device independent language, which can then be used to produce several UI for different computing platforms.

WebRevenge [16] is a tool that analyzes Web site code in order to automatically reconstruct the underlying logical interaction design. Such a design is represented through task models that describe how activities should be performed to reach users' goals.

Teresa [13] follows a task-based approach for the generation of UI for multiple devices. The process of generation is decomposed into 4 parts: 1) the creation of a unique task model. For each task, the designer specifies the interaction objects needed to accomplish the task, and the platforms on which the task is available. 2) A separate task model is then automatically generated for each platform. 3) Based on this task model, a logical UI is then created for each platform, by identifying the enabled task sets (tasks that should appear on the same screen) and the interactors needed to fulfill the tasks. 4) The final UI code is generated. The user has the freedom to modify the heuristics and models at each step of the process to fine-tune the UI.

The goal of XWEB [15] is to produce UIs for several devices starting from a multi-modal description of the abstract UI (it covers only the forward engineering phase). This system operates on specific XWEB servers and browsers (tuned to the interactive capacities of particular platforms), which communicate thanks to an appropriate protocol (XTP). Interactors belonging to the abstract UI are very generic, as just the semantic nature of the interactors is specified. These interactors are designed to capture the nature of a specific data type. This generic description allows multi-modal generation of UI, because there is no assumption about the interaction technique in the description. The clients currently implemented are for desktop, speech and pen-based wall displays.

PUC (personal universal controller) [14] introduces an intermediary graphical or speech interface on several platforms (such as a Pocket PC) for complex appliances. The PUC can communicate both-ways with an appliance, thanks to modification of the hardware of this appliance. The specifications of the appliance's functions (grouped hierarchically with their state-dependence information) are captured in a model and used by the mobile platform to construct the UI.

A more exhaustive review of reverse and re-engineering approaches is available at http://www.isys.ucl.ac.be/bchi/research/soare.htm.

After analyzing these various approaches, we will look at a global approach which makes reverse engineering of Web sites towards any platform in a logical and more flexible way than these tools.

3. The multi-context framework

The Cameleon Reference Framework [4] locates UI development steps for context-sensitive interactive applications. A context is defined as a triple of the form "e, p, u" where e is an element of the environments set considered for the interactive system, p is an element of the platforms set considered for the interactive system and u is an element of the users set for the interactive system. A simplified version (Fig. 1) structures development for two contexts of use, here for two platforms: the one on the left represents the source and the one on the right represents the target.

The development process can be decomposed into four steps:

1. Task and concepts: describe the various tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
2. Logical UI: A canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent of the concrete interactors available on the targets. The elements used in the logical UI are abstractions of existing widgets.
3. Physical UI: concretizes a logical UI into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. This interface is now composed of existing UI widgets.
4. Final UI: The UI produced at the very last step of the reification process is supported by a multi-target development environment. It is expressed as source code.

The downward arrows represent reification steps (forward engineering), from the more abstract to the operational interface. Reification is the transformation of a description (or of a set of descriptions) into a description (or a set of descriptions) whose level of abstraction is lower than that of the source one(s). In the multi-target reference framework, it is the inference process that covers the inference process from high-level abstract descriptions to run-time code. Upward arrows stand for abstraction steps. This process transforms a description into a description whose semantic content and scope are richer/higher than the content and scope of the initial description content. In the context of reverse engineering, abstraction is the elicitation of descriptions that are more abstract than the descriptions that serve as input to this process. Finally, horizontal arrows correspond to the translation of the interface from one type of platform to another, or more generally, from one context to another.

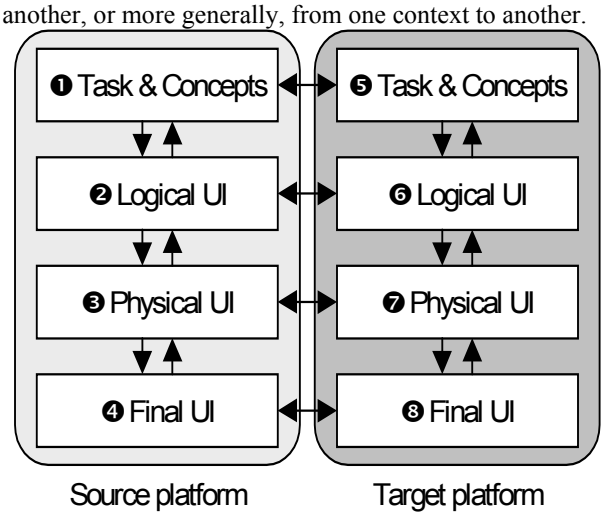

Fig.1 The Cameleon Reference Framework.

The complete definitions of the terms used in this framework can be found at http://giove.cnuce.cnr.it/cameleon/glossary.html.

Not all steps should be achieved in a sequential ordering dictated by the levels. Instead, locating **what** steps are performed, when, from which entry point and toward what subsequent step are important. In Fig. 1, transcoding tools start with a final UI for a source platform (❹) and transforms it into another final UI for a target platform (❽). Similarly, portability tools start with a physical UI for a source platform (❸) and transforms it into another physical UI for a target platform (❼), that in turn leads to a new final UI for that platform (❽). To overcome shortcomings identified for these tools, there is a need to raise the level of abstraction by working at the logical level. *UI Reverse Engineering* abstracts any initial final UI (❹) into concepts and relationships denoting a logical UI (❷), which can then be translated into a new logical UI (❻) by taking into account constraints and opportunities for the target platform. *UI Forward Engineering* then exploits this logical UI(❻) to regenerate a new UI adapted to this platform, by recomposing the physical UI(❼) which in turn is reified in an executable UI(❽).

4. Reengineering of web pages

Reengineering can be decomposed into three separate phases: the detection of the interaction objects, the transformation of these objects to adapt to the particularities of the new platform and the generation of the code in the

new language. The complete reengineering process is shown in figure 2. The input of this reverse engineering is any Web page saved locally. This Web page is reverse engineered by Vaquita in a presentation model expressed in XIML [17]. XIML is a representational notation for describing any UI in a technology and platform independent language. Three main concepts are used to describe the UI in XIML: models, elements belonging to these models and relations between those elements.

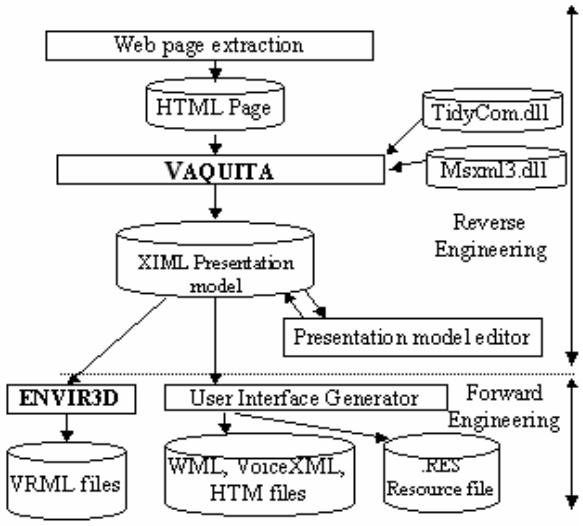

Fig.2 The complete re-engineering process

The presentation model is then transformed into another presentation model for another context (from ❷ to ❻ on fig.1). This step completes the detection and transformation phases of the reengineering process. The presentation model can then be manually modified, thanks to a specific editor, in order to refine the results obtained by Vaquita. The output of the tool can then be used as input in the forward engineering phase, to generate new UIs for other contexts of use. Envir3D generates VRML files starting from XIML presentation models. Other tools will support the generation of UIs expressed in languages such as WML, VoiceXML, etc…

5. Tool Support for Reverse Engineering

5.1 Flexible Reverse Engineering

Vaquita (downloadable at http://www.isys.ucl.ac.be/bchi/research/vaquita.htm) [2,21] reverse engineers any HTML page into a presentation model expressed in XIML. The Web page is firstly cleaned thanks to the tidycom library (http://perso.wanadoo.fr/ablavier/ TidyCom), in order to remove unused tags and to have a standard XML code format. The particularity of our approach is the flexibility of the reverse engineering and this at two levels, the detection and transformation phases. At the detection step, the reverse engineering process can be governed by various options and heuristics:

- Objects, tags, and elements filtering: positive filtering includes any HTML item with given properties (e.g. the developer wants to keep all control widgets) while negative filtering discards it (e.g., no banners).
- Layout options: various layout heuristics establish layout relationships (e.g. alignment, centering, balance) depending on the positions of objects on the page. The developer can then control each heuristic.
- Associations and dissociation: heuristics group objects that are close to each other because they are semantically related (association) and ungroup objects that are isolated without any connection when they are unrelated.
- Contents heuristics: heuristics handle sentence folding or elision, table consideration ordering, etc.

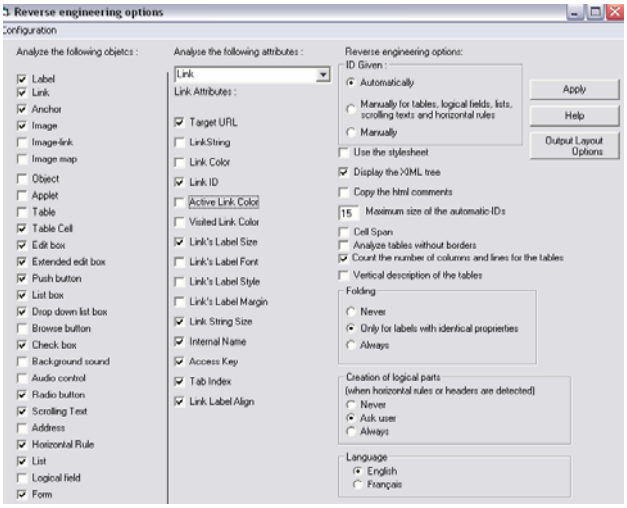

Fig. 3 – options screen in Vaquita

All these options and heuristics are regrouped on the option page of Vaquita. In fig.3, a part of these options is shown. Fig. 4 shows a typical Vaquita interactive session where the screen is divided into three regions: the original HTML code above, the hierarchical decomposition of reverse engineered objects in the bottom left window, and their properties in the bottom right window. Clicking in any of the three frames is automatically reflected in the other coordinated frames to show the corresponding elements or piece of code or property.

The output of Vaquita is a presentation model expressed in XIML. The model is composed of several embedded presentation elements which corresponds to AIOs. More information over the presentation model in XIML can be found in [2] and [21] or on http://www.ximl.org.

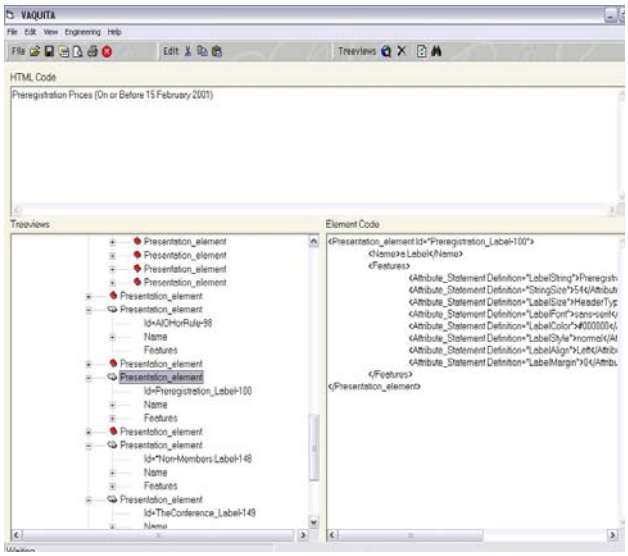

Fig.4 Reverse engineering with Vaquita.

5.3 The Translation Step

After the abstractions steps, the logical translation is then supported by choosing any set of transformations depending on target platforms. These transformations are motivated by two different aims: either the reduction of the size of the original UI, as the main objective of reengineering Web pages is to render them on mobile platforms, or to replace an object by another, because the current object does not exist on the target platform.

Translations from one widget to another are regrouped by target platform or configurations (predefined set of translations), by selecting the most appropriate widget transformation for a specific platform. However, the user can override these mappings by selecting another available translation, so that he can customize the presentation model for a specific case.

6. Automated forward engineering

Forward engineering is then supported to transform the XIML specifications resulting from the reverse engineering to a final UI.

Thanks to Envir3D (fig.5), it is possible to produce VRML code from a XIML presentation model. Originally, Envir3D generates three-dimensional representations of control room. A GUI allows the developer to draw the layout of the control room, by adding AIOs from a library (left part of the screen) on the control room creator (center of the screen). The user can modify the properties of the objects on the right part of the screen, or add its own AIOs by specifying the new XIML definitions in the library. The corresponding XIML presentation model can then be generated from this graphical representation. In a second phase, this presentation model is used to generate VRML code. The developers of control room environments thus have a support in order to evaluate their project, in particular in the field of ergonomics.

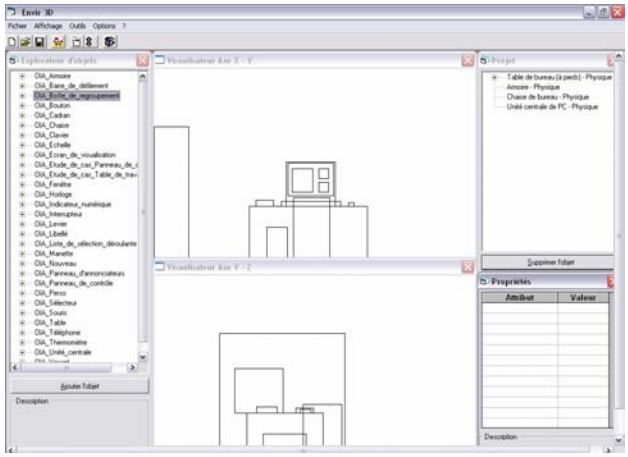

Fig.5 Creation of a control room with Envir3D

Only the second functionality of Envir3D is used in our approach, the generation of a VRML file based on a presentation model. The combination of reverse and forward engineering thus closes the loop to obtain reengineering. In the next section, an example of reengineering towards several computing platforms is shown.

7. Case Study

The Web page selected for this case study is a typical registration form (fig.6). Web Pages are reverse engineered by two successive scans of the code. The first scan detects general information about the page, like the number of logical windows (LW) and where they are situated, the number of radio buttons per group, the number of rows and columns of tables, etc… This form is decomposed during the reverse engineering into 4 logical windows (LW), corresponding to the different parts of the Web page. This subdivision is based on the detection of several tags indicating a "semantic break", like horizontal rules, headers, forms, etc. The construction of the LW is done in an assisted way: Vaquita asks the user for a confirmation before the creation of each LW because these rules are not always true. The resulting presentation model is shown on figure 7.

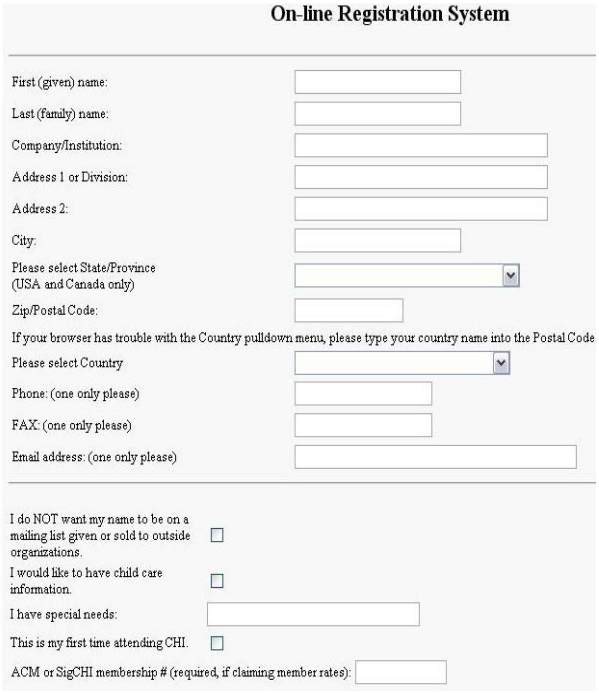

Fig.6 Part of the Registration form

The root element is the Interface element, which has only one child, the window element. This window is the container for the whole UI, and has 4 children, representing the division into 4 LW.

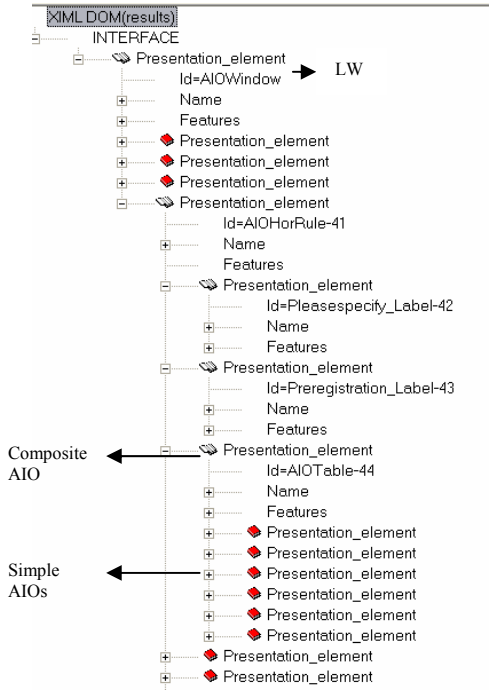

Fig.7 The resulting presentation model

These LW determine the semantic structure of the page. The first contains the titling labels, the second the personal information, the third the conference registration and the last one the paying mode. Each of them contains several other interaction objects, such as labels, textboxes, tables, etc…

The second scan of the code indexes the different interaction objects that the user had selected on the general options page, with their attributes. The layout relations are deduced during this phase. The layout of elements is described in the presentation model in terms of relative positioning. Every element possesses at least two layout-relations: one describing the vertical position of the object and the other the horizontal position relative to other objects on the page. There are at least two relations, because it can happen that an object has several layout relations of the same type, as in the case of a colspan or rowspan for example.

The vertical position can be of four different types: inferiority, containment_top, containment_middle, containment_bottom. Inferiority is deduced by comparing with the preceding element the number of the line on which the interaction object takes place. Every element thus possesses a line attribute, which is incremented when the parser finds one of the following tags :
, <tr>, <h1>,…<h6>, <p>, <blockquote>, <pre>, <hr>, <q>, <table>. The three other relations are used in the case of composite AIO: every simple AIO composing this AIO refers to the composite element (i.e. table cells or tables, fieldset, etc). The horizontal position can be of six different types: succession, justified_right, justified_left, containment_left, containment_right, containment_center. Succession is used for two consecutive elements without any information into the HTML code concerning its positioning. Justification on the left (or on the right) is expressed in relation to the element on the preceding line. It can be deduced for elements in the same column of a table or for the first (or last) elements onto two consecutive lines. The last three horizontal relations are used to describe the layout of elements contained in a table-cell or to describe the first elements on a page (as there is no reference at this point of the process, elements have to be referred to the window).

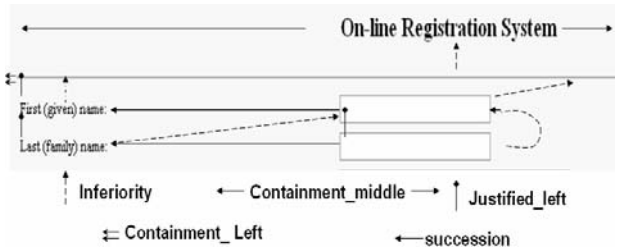

Fig.8 Example of layout relations

In fig.8, layout relations are shown for a small part of the case study. As there is no reference point, the first label is defined with respect to the window, thanks to the containment_middle relation. This is the same case for the horizontal rule, as there is no element on the left of the screen. The position of this rule is thus defined with respect to the window, but now there is a reference for the vertical position and the rule is thus inferior to the first label. The first element of the form, the label asking for the first name, is below (inferiority) the horizontal rule, and justified on the left in relation to the rule.

The first textbox follows the first label, and is inferior to the horizontal rule. The second line of the form is composed of a label, which is left justified with the preceding label and inferior to the last textbox (the inferiority relation is always expressed relative to the last object of the preceding line). The last checkbox is justified on the left with the textbox of the preceding line and is below the same textbox. The different layout relations are shown in table 1.

	Horizontal Relation	Vertical relation
Between Simple AIOs	Succession	Inferiority
	Justified_Left	
	Justified_Right	
Between Simple and Composite AIOs	Containment_Left	Containment_top
	Containment_Right	Containment_middle
	Containment_Center	Containment_bottom

Table 1. Layout Relations in HTML

These layout relations can be used to describe any other UI than HTML, by adding other relations such as horizontal or vertical balancing. The main drawback of this method is that it cannot handle absolute positions for the moment. Absolute positions can be specified in the layer tag or in javascripts, to display elements (sometimes dynamically) at a precise zone of the screen.

After the second scan of the HTML code, the translation step transforms the presentation model into a customized presentation model for Envir3D. The presentation elements are transformed into elements belonging to the widget set of Envir3D, and the relative positions of the objects are transformed into absolute positions, thanks to a constraint-solving algorithm. This transformed presentation model is then used by Envir3D to produce a virtual UI expressed in VRML (see fig.9)

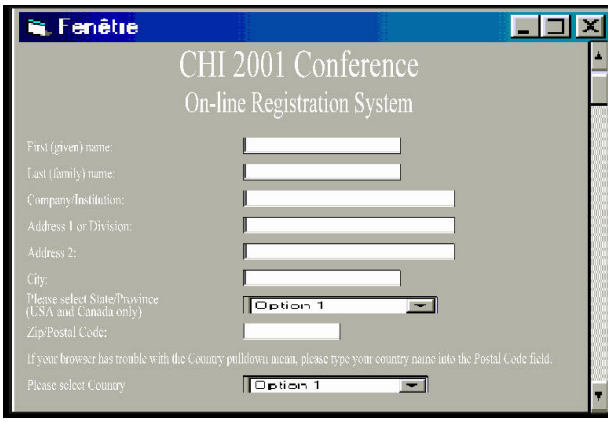

Fig.9 The original interface regenerated in VRML

Fig. 10 shows the registration form resulting from the manual forward engineering for a smartphone, a WAP-enabled mobile phone (fig.11) and Desktop PC (Visual Basic UI – fig 12).

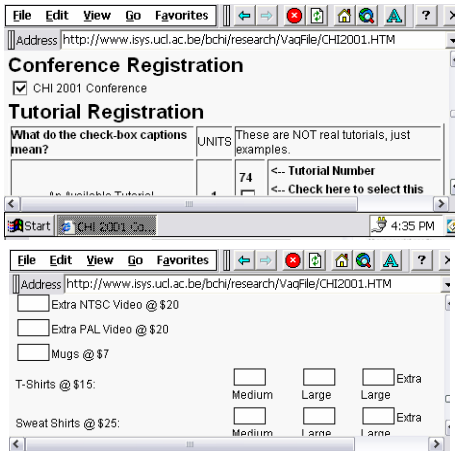

Fig.10 re-engineering on a smartphone.

Fig.11 re-engineering on a mobile phone.

The transformation of the presentation model for a mobile platform moves every input box onto a separate screen, as mobile phone cannot always display two inputs on the same screen. The page has been divided into 4 LW, which are represented on the mobile phone as separate cards, accessible through different links. Several other

transformations are used for the conversion of the presentation model, such as the division of too long tables in a series of paragraphs, the deletion of the style attributes of labels (color, font size …), the transformation of checkboxes in drop down list boxes, etc.

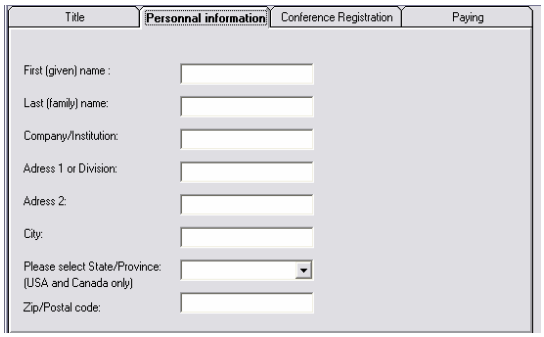

Fig.12 Results from re-engineering (VB UI).

In the last picture (figure 14), the UI has been redistributed onto 4 thumbnails, corresponding to the different logical windows detected by Vaquita. This decomposition into thumbnail is one of the possible mappings of the logical windows (which are physical or semantic parts of a presentation model) on platforms with rich capabilities.

8. Ongoing issues

The state of our research has allowed us to detect some critical points which will be explained in the following subsections.

8.1 Dynamic Uis

More and more Web pages are generated on the fly, because they use data stored in a database which can be queried through the Web site or to adapt automatically to the client platform or user profile. These web pages are described in an imperative language (such as PHP), and are not analysable by Vaquita. We circumvent this problem by making a request to the server for the page, downloading the code, and reverse engineering this page (which can thus be different from another platform on which Vaquita runs), but this way of proceeding does not guarantee us to have reverse engineered all the possible presentations of the generating-script.

8.2 Script Languages

Script languages allow the addition of dynamic effects on Web pages or the carrying out of simple treatments on data (such as, for example, the checking of the answers of a form). The reverse engineering of scripts would require a more complex resolution than for HTML, as scripts languages are imperative languages and the reverse engineering of this kind of languages is completely different from the reverse engineering of declarative languages. A manner of decreasing the complexity of this task would be to identify a series of "standard scripts" recognizable by pattern matching (recognition of sequences of instructions). This manner of proceeding would adapt the complexity of resolution of the problem to the relatively weak importance of these scripts.

8.3 Automated recognition of relations

Semantic relations between elements are currently identified manually in Vaquita. To do this, a special dialogue box pops up and allows the user to define relations between elements in an assisted way. There are two sorts of relations in the reverse engineered presentation model: layout relations and semantic relations.

8.3.1 Layout Relations. The layout relations define the positioning of interaction objects in relation to each other. The layout is not defined explicitly in HTML but it is possible to deduce the position from the objects thanks to their positions in the code of the page. The importance of the relations of placement in the presentation model depends on the platform towards which the user wants to migrate the interface. If this has display capabilities relatively close to those of the PC (as on a smartphone), these relations can appear very useful because it is then possible to regenerate an interface similar to that of the PC. On the contrary, if the capacities differ greatly, all the relations must be altered and therefore their presence brings few advantages.

8.3.2 Semantic Relations. Semantic relations represent principally the links between the labels and the objects that those labels describe. However, this kind of relation can be applied to every object to show that two objects are linked semantically. This relation is of great importance as it makes it possible to redistribute the interaction objects, to change the structure of an interface while preserving the semantics of the UI. The difficulty of the reverse engineering of these relations is very great, since no indication is given in the code to find it. Only the "geographical" localization on the page of these labels compared to the objects can make it possible to deduce these semantic relations. We are currently considering the following semantic relations: hierarchy (given by the fontsize, bold texts, etc…), grouping (functionally or spatially), ordering (ordered lists, unordered lists in rows or columns), emphasizing (shadow, color change, etc.) and differentiating.

In [8], an automatic deduction system of this kind of relations has been implemented. In this approach, all the possible semantic links between labels and their surrounding elements are evaluated thanks to a list of heuristics. According to this evaluation, a score is given to each pair, and the most probable one is selected to establish the relation.

9. Future Work

We are currently implementing a new version of the reverse engineering tool, Vaquita 2.0, in PHP (see fig.13) and thus in a server-side architecture.

Fig. 13 – Vaquita 2.0 in PHP

The advantage of this kind of implementation is that the tool can be used both manually as in the previous version and automatically on a server to adapt the Web pages of the site dynamically. In the second case, Vaquita can be used as a function with three parameters: the input file, the reverse engineering configuration file to use and the output file. When a request to a Web page is sent, the information about the user's platform is used to select the most appropriate configuration file. These configurations are stored in a knowledge base with one separate file for each platform. For the moment, only basic information can be used to make this choice, but in a near future, the cc/pp protocol will provide all the needed information to make an accurate selection. The time saved with this approach has a price: it will give less flexibility and less control over the transformation of the UI. Therefore, the features of Vaquita 1.0 are kept in this new version: the possibility to fine tune all the reverse engineering options, the control of the creation of logical windows and semantic links and the re-edition/correction of the presentation model before the UI code generation.

10. Conclusion

The approach, the model, and the supporting tools presented in this paper are different from existing work in the sense that it does not consider the pair fixed (source platform, target platform). Many models and tools exist that translate one UI to another one or multiple Uis for fixed pairs. When there is a need to consider multiple target computing platforms, the ad hoc approach no longer remains a viable approach. Therefore, manipulating the UI at a higher level of description than merely the code level is to be expected. Moreover, platform to platform tools usually support limited conversion and application of rules and heuristics in a very rigid way.

In contrast, we presented an approach that reverse engineers web pages of web sites to a level where it can be regenerated for itself (redesign of web sites or reformatting) or for other computing platforms. Corresponding Uis have been introduced for mobile phones, SmartPhones, and Pocket PC.

Advantages resulting from composing UI reverse engineering, logical translation, and forward engineering are:

- *Generality*: once a logical UI is obtained, it can be submitted to any set of transformations in the logical translation to accommodate the target platform. This process is no longer specific to any source-target pair.
- *Flexibility*: the reverse engineering process can be parameterized to reverse engineer only those UI elements of interest and rejecting those not concerned.
- *Controllability*: the developer can control the process both in the reverse engineering and in the translation.
- *Reusability*: the reverse engineering needs to be operated once. The resulting UI can then initiate as many transformations as there are needs to port it so as to create a new UI for any newly considered platform.
- *Abstraction*: the UI is logical and can be treated in many ways that are more logical than at the code level, thus allowing designers to explore alternative design options that would be otherwise impossible to cover at the code level.

Acknowledgements

The authors would like to thank Pr.Carlon from ILV-UCL for the correction of this paper. We gratefully acknowledge the support of the EU IST 5 Project « Cameleon » (http://giove.cnuce.cnr.it/ cameleon.html). This Work is based in whole or in part on the XIML Language and Schema. Trademarks and Copyrights for XIML are owned by RedWhale Software Corp., Palo Alto (CA), USA.

References

1. M. Abrams, C. Phanouriou, A. Batongbacal and J. Shuster, UIML: An Appliance-Independent XML User Interface Language, in *Proc. of 8th World Wide Web Conference WWW'8 (Toronto, 11-14 May 1999)*, Computer Networks, Vol. 31, No. 11-16, pp. 1695-1708
2. L. Bouillon, J. Vanderdonckt, Retargeting Web Pages to other Computing Platforms, *Proc. of IEEE 9th Working Conference on Reverse Engineering WCRE'2002* (Richmond, 29 Oct-1 Nov 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.
3. K.H. Britton, R. Case, A. Citron, R. Floyd, Y. Li, C. Seekamp, B. Topol, K. Tracey, "Transcoding: Extending Ebusiness to New Environments", *IBM Systems Journal*, Vol. 40, No. 1, 2001, pp 153-178.
4. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, N. Souchon, L. Bouillon, J. Vanderdonckt, Plasticity of User Interfaces: A Revised Reference Framework, in *Proc. of 1st International Workshop on Task Model and Diagrams for user interface design* Tamodia'2002 (Bucharest, 18-19 Jul 2002), INFOREC Publishing House Bucharest, Romania, 2002
5. L.Csaba, Experience with User Interface Reengineering – Transferring DOS panels to Windows, *1st Euromicro Working Conference on Software Maintenance and Reengineering* CSMR 97, Berlin, Allemagne, 17-19 Mar, IEEE Computer Society Press, Los Alamitos, USA, 1997
6. E.J. Chikofsky and J.H. Cross, *Reverse Engineering and Design Recovery: A Taxonomy,* IEEE Software, Vol. 1, No. 7, Jan 1990, pp. 13-17.
7. M. El-Ramly, P. Ingiski, E.Stroulia, P.Sorenson, B.Matichuk, Modeling the System-User Dialog using Interaction Traces, in *Proc. of the eighth Working Conference on Reverse Engineering* (2-5 Oct 2001, Stuttgart, Germany), IEEE Computer Soc. Press, Los Alamitos, USA, 2001, pp. 208-217
8. Y. Gaeremynck, L. D. Bergman, T. Lau, "MORE for less: model recovery from visual interfaces for multi-device application design", *Proc. of the international conference on Intelligent user interfaces* Jan 2003 (Miami, Florida, USA), ACM Press, New York, USA, 2003, pp 69-76
9. E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski and T. Laakko (2000), Two Approaches to Bringing Internet Services to WAP Devices, *WWW9* / Computer Networks, 33 (1-6):231-246, 2000
10. E. Merlo., P.-Y. Gagné, J.-F.Girard, K. Kontogiannis, L. Hendren, , P. Panagaden, and R. De Mori, , Reengineering User Interfaces, IEEE Software, Vol. 12, No. 1, Jan 1995, pp. 64-73.
11. M.M. Moore, Representation Issues for Reengineering Interactive Systems, *ACM Computing Surveys Special issue: position statements on strategic directions in computing research*, Vol. 28, No. 4, Dec 1996, article # 199, ACM Press, New York, NY, USA
12. M.M. Moore and S. Rugaber, Using Knowledge Representation to Understand Interactive Systems, in *Proc. of the Fifth International Workshop on Program Comprehension IWPC'97* (Dearborn, 28-30 May 1997), IEEE Computer Society Press, Los Alamitos, 1997
13. G. Mori, F. Paternò, C. Santoro, Tool support for designing nomadic applications, *Proc. of the 2003 international conference on Intelligent user interfaces, Jan 2003*, (Miami, USA), ACM Press, New York, USA, pp141-148
14. J.Nichols, B.A. Myers, M.Higgins, J.Hughes, T.K.Harris, R.Rosenfeld, M.Pignol, "Generating Remote Control Interfaces for Complex Appliances", *Proc. of the 15th annual ACM symposium on User interface software and technology*, (Paris, 27-30 oct 2002), ACM press, New York, USA, 2002
15. D.R. Olsen, S. Jefferies, T. Nielsen, W. Moyes, P. Fredrickson, Cross Modal Interaction using XWEB, *Proc. of the 13th annual ACM symposium on User interface software and technology* UIST 2000 (San Diego, United States), ACM Press, New York, USA, pp 191-200
16. L.Paganelli, F.Paterno, Automatic reconstruction of the underlying interaction design of web applications, in *Proc. of the 14th international conference on Software engineering and knowledge engineering,* (July 2002, Ischia, Italy), ACM Press, New York, USA, pp 439 - 445
17. A. Puerta, J. Eisenstein (2002), "XIML: A Com mon Representation for Interaction Data", *Proc. of the 7th international conference on Intelligent user interfaces* (San Francisco, 14-16 Jan 2002), ACM Press, New York, NY, USA.
18. F. Ricca, P. Tonella, I.D. Baxter, "Restructuring Web applications via Transformation Rules", *Proc.of IEEE Workshop on Source Code Analysis and Manipulation SCAM'2001* (Florence, 5-9 Nov 2001), IEEE Computer Soc. Press, Los Alamitos, 2001, pp. 150-160.
19. E. Stroulia, J. Thomson, Q. Situ: Constructing XML-speaking wrappers for WEB Applications: Towards an Interoperating WEB. *In the Proc. of the 7th Working Conference on Reverse Engineering* (WCRE'2000). 23-25 November, 2000. Brisbane, Queensland, Australia, IEEE Computer Society Press, Los Alamitos, 2000
20. J. Sussman, G. Banavar and L. Bergman. Generalization: A Key Concept in the Creation of Platform Independent User Interfaces. In *UbiTools 2001, Workshop on Application Models and Programming Tools for Ubiquitous Computing*, Atlanta, GA, Sept 2001
21. J. Vanderdonckt, L. Bouillon, N. Souchon, Flexible Reverse Engineering of Web Pages with VAQUITA, *Proc. of IEEE 8th Working Conference on Reverse Engineering WCRE'2001* (Stuttgart, 2-5 Oct 2001), IEEE Computer Society Press, Los Alamitos, 2001, pp. 241-248.