Flexible Reverse Engineering of Web Pages with VAQUISTA

Jean Vanderdonckt, Laurent Bouillon, and Nathalie Souchon

Université catholique de Louvain, Institut d'Administration et de Gestion
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
E-mail: vanderdonckt@qant.ucl.ac.be, lbouillon@ibelgique.com, souchon@qant.ucl.ac.be

Abstract

VAQUISTA allows developers to reverse engineer a presentation model of a web page according to multiple reverse engineering options. The alternatives offered by these options not only widen the spectrum of possible presentation models but also encourage developers in exploring multiple reverse engineering strategies. The options provide filtering capabilities in a static analysis of HTML code that are targeted either at multiple widgets simultaneously or at single widgets at a time, for their attributes and other manipulations. This flexibility is particularly important when the presentation model is itself used to migrate the presentation of the web page to other types of user interfaces, possibly written in different languages, in different computing platforms.

Keywords: Forward engineering, heuristics, knowledge bases, interface migration, model-based approach, presentation model, reverse engineering, static analysis, web page, WML, XML, XHTML.

1. Introduction

Making a web site accessible from the widest range of computing platforms, including the variation of browsers on each platform, for the widest range of users is of course desirable. The technological push of newly marketed access devices—such as Personal Digital Assistants (PDAs) and Wireless Access Protocol (WAP)-enabled cellular phones—has exacerbated this need to access the same web site, from different access devices. With the demand for Internet access increasing for e-mail, shopping, electronic commerce, current events, and quick information, the need for Internet-capable access devices has extended beyond the professional desktop to the home office, the other rooms of the house, and beyond. This demand is still increasing, as we will surely become more dependent on the web for information.

To address these demands, ad hoc development is no longer believed acceptable in terms of the cost and time required for software engineering, development and maintenance. *Model-based approaches* [13,15,22,25] can produce a user interface (UI) for a web site in a forward or reverse engineering manner by exploiting knowledge [16] captured in various models, such as task [8], domain [5], presentation [15] and dialog [10] models. However, most existing web sites have been developed without any model-based approaches. The growing interest in web site development has increased the demand for engineering methods that are more structured than traditional *ad hoc* development or "rush to code" approaches. If the author wants to make the web site accessible from multiple web appliances without loosing the development effort deployed for the web site, there is a need to reverse engineering existing web pages to generate models which can later be exploited in forward engineering.

The problem of migrating the UI of a web page to another environment is really compounded as many programming (e.g., C++, Java) or markup (e.g., HTML, WML) languages exist for many different types of UI (e.g., character-oriented, graphical, vocal) for computing platforms having all their own specific set of constraints (fig. 1).

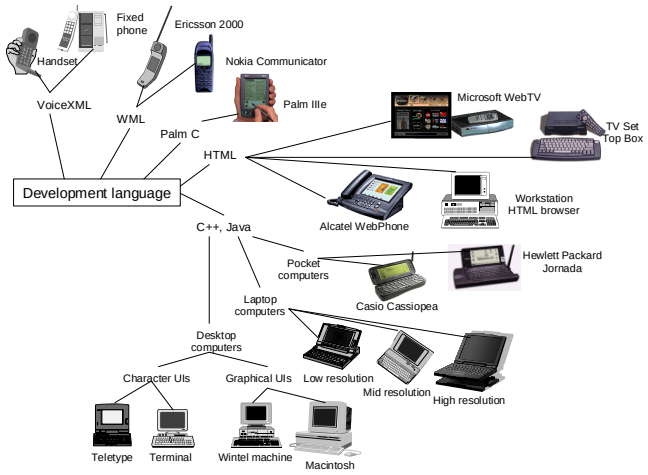

Fig. 1. Different languages for developing different UIs.

The goal of this paper is to address this problem by examining how a presentation model of a web page can be reverse engineered to migrate it to another environment. The remainder of this paper is structured as follows: section 2 provides a state of the art of techniques used to access an HTML page from other devices than a browser; section 3 poses the working hypotheses and specifies the goals and aims assigned to this work; section 4 defines concepts and techniques used in the VAQUISTA software; section 5 discusses implementation issues and section 6 concludes the paper by reporting on progress made with VAQUISTA and some identified shortcomings.

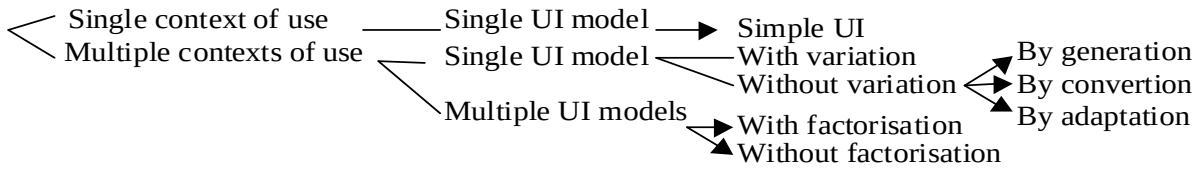

Fig. 2. Some existing techniques for accessing HTML pages from multiple contexts of use

2. Related Work

Many research and development have already been conducted to address the problem of making a web site accessible in multiple contexts of use. By *context of use*, we hereby refer to as a set of environment parameters describing a given context in which a particular user (or a stereotype of user population) is carrying out a particular interactive task. The notion of context of use, sometimes used for environment, namely includes user parameters and software/hardware capabilities. Such parameters as user preferences, native language, computing platform, operating system, screen resolution, screen size, UI language are typical of a context of use.

When a web site only needs to be accessed in a single context of use (fig. 2), say for instance from within a typical Web browser on a Windows-compatible machine, a single conceptual UI model, leading to a single UI, is satisfying since there is no need to support other cases. When supporting multiple contexts of use is requested in the software requirements, two basic approaches can be followed:

1. *Single UI model-based approach*: although multiple contexts are to be supported, the developer still remains with one comprehensive UI model which may or may not include variation in its modeling. For instance, a HTML code may incorporate a Java script detecting the computing platform, the browser name, and the screen resolution to tailor the presentation depending on these parameters (e.g., for compatibility with different browsers). Without variation included, three sub-cases are found:

   - By *generation*: when the starting model initiates many UIs for different contexts of use. For instance, UIML [1] captures presentation and dialog specification in a XML-based vocabulary which can be in turn used to generate Java code for Windows platforms, VoiceXML for telephones, WML for WAP-compliant cellular phones, and PalmOS for PalmPilots with appropriate renderers.
   - By *conversion*: when the starting model initiates a UI to be converted at design-time or run-time to make it compatible with another context of use. For example, Vizzavi (www.vizzavi.com) is a web portal that automatically transforms any HTML page into decks and cards for WAP devices. Similarly, [7] describes a web proxy server that converting HTML into WML by using transformation heuristics of HTML tags and elements into WML objects. These heuristics are hard-coded, but the conversion is transparent for the end-user.
   - By *adaptation*: when the starting model initiates a single UI that is adapted to a specified context of use. For example, [9] restructure HTML, XHTML and XML pages/documents into PalmPilot screens by classification of original elements and adaptation according to a device and a user model. Digestor [2] provides device-independent access to web pages by relying on structural page transformation and sentence elision. This approach produces results that are more suited to a target context than the previous approach, but deletes text. Accordion summarization [3] enables users to browse web pages on small devices using a syntactic page summarization and a browsing mechanism based on structural text units (STUs) to progressively expand or collapse information.

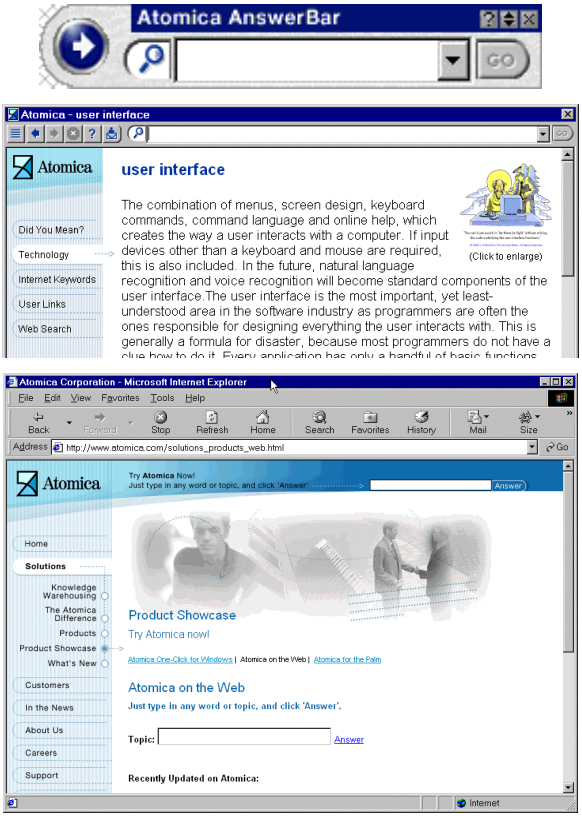

Fig. 3. Multiple versions for three various contexts of use.

2. *Multiple UI model-based approach*: the developer captures into different models the peculiarities of the different contexts of use to consider. The different models may or may not share parts that are common to different contexts of use. For example, Atomica (www.atomica.com) developed 3 separate models to support its service on Windows Personal Computer, Pocket Computer, and HTML browser (fig. 3). In contrast, a context-sensitive task model [25] identifies sub-tasks that are common to all different contexts of use and isolates the rest for other contexts which are selected according to a decision tree. This gives rise to multiple UIs that are tailored at design-time.

3. Hypotheses, Aims and Goals

Transformation techniques outlined in section 2 can be classified according to three orthogonal dimensions:

1. *Consistency* vs *variability*: some techniques are aimed at reproducing the same UI for the different contexts of use, whatever their constraints are. This process is questionable: for instance, HTML pages directly rendered on dramatically varying devices are perhaps cross-platform (or cross-context) consistent, but induce huge scrolling manipulations by the end user. In contrast, variability attempts to meet the contexts' constraints, while preserving some consistency, sometimes with no consistency at all.
2. *Design-time* vs *run-time*: transformation from one context to another can be expressed and operated at design-time (before executing the interactive application as in [6]) or at run-time (when executing the interactive application). The first may be interpreted a smarter approach if local constraints are addressed at a higher-level than simply the syntactic code level of the run-time. However, some on-line converters become more and more sophisticated as [3], while maintaining transparency for the end user who even do not notice the transformation process.
3. *Partial* vs *total*: transformation can be targeted to support only some sub-tasks in the new context (e.g., due to local constraints and limited interaction capabilities, it is impossible to maintain the same level of usability [23]) or all sub-tasks. In particular, plastic UIs attempts to cover multiple contexts of use, while maintaining a predefined threshold of usability at design-time [23].

Our work is situated in circumstances where an organization already developed a web site, but did not consider multiple contexts of use when designing the web site. Such a case occurs more frequently than one think not only in the past when the availability of web appliances was not that wide, but also today because of lack of time, resources, budgets, and skills. Yet, this organization does not want to start again from scratch the next version for another context of use and does not want to loose the development momentum gained in the first design.

Therefore, our goal in this paper is not to reverse engineer a whole web site to migrate it to another context of use or to make it accessible for a non-compatible computing platform. Indeed, interactive applications and web sites only share limited common characteristics. Rather, our goal is to migrate highly-interactive parts (e.g., input forms) of a web page to other context by combining a reverse engineering approach first and a forward engineering approach after (fig. 4).

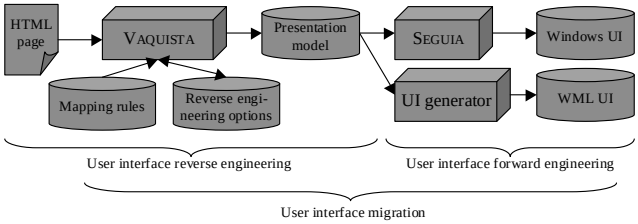

Fig. 4. The complete process envisioned with VAQUISTA.

VAQUISTA (reVerse engineering of Applications by Questions, Information Selection, and Transformation Alternatives) is a software covering the user interface reverse engineering by recovering a presentation model from a single HTML page at a time based on mapping rules (between HTML elements and presentation elements). This process should support transformation with:

- *Variability* to explicitly address local constraints, while maximizing consistency.
- At *design-time* because the process should obtain a model that is high-level enough to warrant any future potential redesigning for migration, even the unforeseen. For this purpose, the presentation model should be stored in an easily manipulable format where applying redesigning options remains as flexible as possible.
- As *total* as possible to factoring out parts of the UI which may be shared across several contexts of use.

The major requirement of VAQUISTA is that it should support the reverse engineering in such an intelligent way that is flexible and developer-controllable enough to maximize the predictibility of the reverse engineered model and to encourage the exploration of multiple reverse engineering options.

After reviewing the state of the art, we observed that allowing only one single transformation technique, even the more elaborate, does not address this requirement. Multiple techniques should be available to the developer to render both the reverse and the forward engineering more flexible. This is not possible with built-in capabilities offered by generation, conversion, or adaptation.

The presentation model is then used in a model editor

where the developer can drag and drop elements of the previous presentation and tailor it to another context. It is for example possible to redistribute presentation elements at the individual or aggregate level by drag and drop. Some software already support the second part of fig. 4. For instance, SEGUIA automatically generates a Windows UI from a presentation model by allowing the developer to explore several layouts [25]. Other generators can be imagined to map any redesigned presentation model into a designated context of use. It is optimistically assumed that developers are powered, trained, and creative enough to apply high-level redesign options that are appropriate in the forward engineering part. In this second part, many redesign options may occur to tailor the presentation to a specific context, such as widget resizing (manual or automated), reallocation of widgets throughout windows [6], change of widget [14], graceful usability degradation [23], and layout rearrangement [25].

4. Reverse Engineering Process in VAQUISTA

The reverse engineering process in VAQUISTA consists in applying a static analysis of HTML elements belonging the to source code of a considered web page (sub-section 4.1) and in mapping them onto elements of a presentation model (sub-section 4.2) using flexible reverse engineering options (sub-section 4.3).

4.1 The Source: a Web Page Written in HTML

Any web page written in HTMl 4.0 is submitted to a static analysis. This technique consists in scanning the HTML code (which is accessible through the Internet) to understand it without executing it [4,17]. This scanning activity results in identifying types of HTML tags, elements, and possible attached values (example of fig. 5).

4.2 The Target: a Presentation Model

A *presentation model* [19,20,22] is a representation of the visual, haptic, and auditory elements provided by a UI to its users. For example, a presentation element may be a window that contains additional elements such as widgets that appear in that window. This model also includes static presentation attributes, such as font styles and orientation of button groups. Presentation of web pages is progressively abstracted using four concepts, the hierarchy of which is depicted in fig. 6:

1. *Concrete Interaction Object* (CIO): this is a real object belonging to the UI world that any user can see (e.g., text, image, animation) or manipulate such as a push button, a list box, a check box. A CIO is said to be *simple* if it cannot be decomposed into smaller CIOs. A CIO is said to be *composite* if it can be decomposed into smaller units. Two categories are distinguished: *presentation CIO*, which is any static CIO allowing no user interaction, and *control CIO*, which support some interaction or UI control by the user.

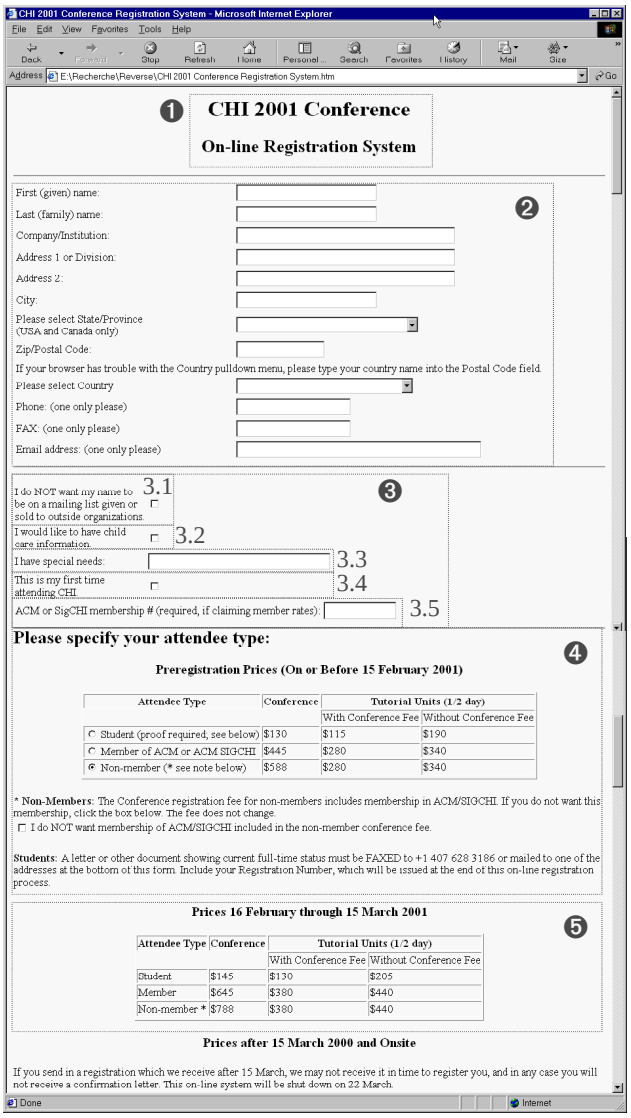

Fig. 5. Example of a web page to reverse engineer. (Rectangles are drawn to facilitate reference in the text – Source: http://www.acm.org/sigchi/chi2001/registration.html).

2. *Abstract Interaction Object* (AIO): this consists of an abstraction of all CIOs from both presentation and behavioral viewpoints that is independent of any given computing platform. AIOs have been used successfully for both forward [24] and reverse engineering [10,14,15]. Each AIO is here identified by a unique generic name (e.g., check box), general and particular abstract attributes (e.g., height, width, color, states), abstract events (e.g., value selection, mouse click), and abstract primitive (e.g., Pr-EditBoxContent). By definition, an AIO does not have any graphical appearance, but each AIO is connected to 0, 1 or many CIOs having different names and presentations in various computing platforms. 32 AIOs were described in a "is-a" hierarchy of classes into a knowledge base [25].

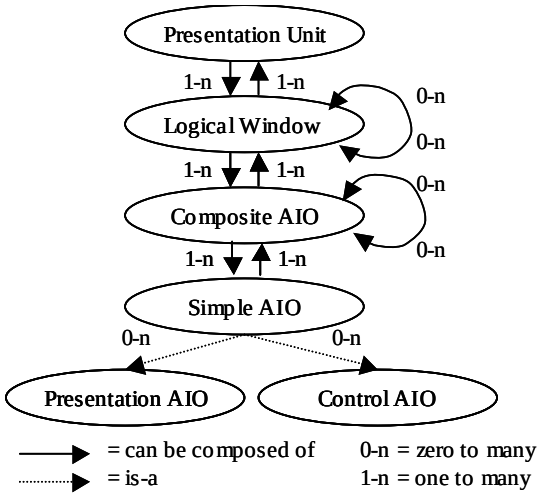

Fig. 6. Hierarchy of presentation concepts.

3. *Logical Window* (LW): this root window can be considered either as a logical container for AIOs or as a physical window, a dialog box or a panel. Every window is itself a composite AIO as it is composed of other simple or composite AIOs. All LWs are supposed to be physically constrained by the user's screen. The three abstractions that have been considered so far are quite related to existing presentation objects. However, none of the presentation abstractions described thus far are closely related to task aspects. The following abstraction is introduced for this purpose.
4. *Presentation Unit* (PU): a PU is assumed to be the complete presentation environment required for carrying out a particular interactive task. Each PU can be decomposed into one or many LWs, which may or may not be all displayed on the screen simultaneously. Each PU is composed of at least one window called the *basic window*, from which it is possible to navigate to the other windows. For instance, a tabbed dialog box is here mapped onto a PU, which is itself decomposed into LWs corresponding to the dialog box appearances depending on the active tab; conversely, a web form can be mapped onto a composite AIO in a particular LW of a given PU.

The concepts of AIO, whether simple of composite, LW, and PU all belong to the presentation model as presentation element. Therefore, any presentation model is organized as a hierarchy of presentation elements, any of – them potentially being itself decomposed into other presentation elements iteratively.

Graphical UIs traditionally incorporate widgets such as controls, windows, dialog boxes, and panels. On the one hand, some of these widgets are also found in web pages, basically the web pages holding some interaction with the end user and, on the other hand, web pages also add new types of presentation elements that are not necessarily found in traditional UIs (e.g., as in [14]). Therefore, AIOs which were pretty classic for these traditional UIs may need to be revisited for web pages in two ways:

1. *New AIO definition*: a presentation element not present in the knowledge base of traditional AIOs needs to be added along with specific attributes, events, and primitives. Such elements include for example links, lists, cells, and forms.
2. *Existing AIO enhancing*: a presentation element which already existed in the knowledge base of traditional AIOs needs to be adapted to model characteristics of interest which were not modeled before. It could be expanding the definition of an attribute with new values, adding new attributes, events, or primitives, and updating the global element definition. Such elements include for example image, table, and separator (with horizontal and vertical lines).

4.3 The Means: Reverse Engineering Options

The transformation of statically analyzed HTML tags, elements, and associated values into instances of presentation elements is established through mapping tables. For example, table 1 shows how elements and the associated values of a text field or password field in HTML are mapped onto an instance of an edit box AIO with corresponding values of its abstract attributes. In particular, Value gives the string by default hold in the edit box.

Edit box attributes	Input type = text OR password
NAME	name= +*TextBox*
AT_EDB_MAX_LENGTH	maxlength=
AT_EDB_FIELD_LENGTH	size=
AT_EDB_STRING	value=

Table 1. Example of a simple mapping table.

Extended edit box	Text area
NAME	name=+*EDM*
AT_EDM_NB_LIN	rows=
AT_EDM_NB_COL	cols
AT_EDM_EDITABLE=no	disabled
AT_EDM_STRING	value=

Table 2. Example of another mapping table.

Table 2 shows another example of a mapping table where a text area is mapped onto an extended edit box AIO (edit box with multiple lines of editing). The HTML name is mapped onto the NAME abstract attribute along with its value. The number of rows and lines are mapped onto the AT_EDM_NB_COL and LIN abstract attributes respectively. When the text area is not editable, the flag AT_EDM_EDITABLE is set to no.

As shown in the related work section, many systems are using more or less similar pattern matching techniques. In

VAQUISTA, the difference relies in its flexible usage of these mapping tables through reverse engineering options, which are classified in the following categories:

- *Single-object options* are dedicated to one and only one presentation element at a time. For example, it can govern the reverse engineering of any occurrences of a given AIO type. These options are subdived into two sub-categories:

    1. *Attributes options*: during the reverse engineering process, some elements and values of HTML tags are considered principal in their corresponding presentation model (such as the name, the AIO type, the length) while others are believed secondary. All principal elements are automatically reverse engineered, while secondary elements are left to the appreciation of the developer as they are optional. The developer can specify that any elements should be taken into account by checking a check box in the option box corresponding to the AIO type.

    2. *Alternative heuristic options*: as many techniques, rules, and heuristics can be used to drive the reverse engineering, the specification of which technique, rule or heuristic based on matching tables is left to the appreciation of the developer. The developer can specify any default alternative heuristic through the alternative option box. Here are some examples:

        - Text folding: in fig. 5, the sub-title of the page is divided in two lines. A static analysis which is not semantic based is not able to infer whether these two lines are semantically related. Therefore, a folding option may be specified to tell VAQUISTA to consider any series of subsequent lines as one presentation element (here, a label). Conversely, the unfolding option specifies that any such series should be reverse engineered into several instances of a label AIO with a relationship between to keep their sequence.

        - Table mapping: fig. 5 hold a table specifying conference tutorial choices. Again, no static analysis can deduce from this structure whether it is more appropriate to consider the table line by line or column by column. The table mapping option allows the developer to drive any reverse engineering of a table by generating a presentation element for each line or each column, possible with a header. Each line or column is then recursively decomposed into a series of presentation elements representing the table cells. Again, each cell can be reverse engineered into many different AIO types, depending on its contents. If each cell is occupied by a widget, then the corresponding AIO may be selected. If not, the cell AIO might be preferred.

        - Cell span: fig. 5 also contains some cells which are spanning over several columns (which can be detected for instance with COLSPAN=2 in this case). Again, an option may be specified to have a cell covering two columns in this line or to keep the line as a sequence of 3 presentation elements if the layout is not considered important.

- *Multiple-objects options* are defined with a scope of several presentations elements simultaneously, possibly widespread in the HTML page or possible of different AIO types. A representative example is the FORM tag in HTML which can hold several other types of presentation elements. HTML forms can be reverse engineered to different instances of different AIO types. When transforming HTML to WML for instance, a multiple-object option may specify which AIO type to use to perform the reverse engineering: a simple table, an indexed sub-tree or a list [7]. HTML forms contain a group of user-input elements all of them being reverse engineered easily. However, the graphical layout and most of the grouping of the user-input elements should not be lost in the process. To preserve this, fifteen layout relationships (described in [25]) are available to capture left, right, center, total justification, top/bottom alignments, horizontal/vertical distribution, horizontal/vertical equilibrium, etc. User-input elements on a form are converted into a single card, for instance, where they appear in the same order as in the original HTML code [7]. Similarly, each HTML frame of a frameset is converted to one or more global presentation elements (here, presentation units or to one or more WML decks). Framesets are converted into decks that provide indices to WML decks that correspond with individual frames

The presentation model is then stored in XIML [21], a XML-based vocabulary for specifying models involved in any model-based approach for designing UI. XIML is the XML-compliant version of MIMIC, a meta-language for UI modeling [18]. XIML holds two types of relationships in the presentation model: a hierarchical relationship between nested presentation elements to reflect the hierarchical decomposition of the whole presentation into presentation elements, and spatial presentation relationships between the elements themselves (e.g., alignment).

Fig. 7 illustrates a typical interactive session with VAQUISTA. Three views are displayed: a tree view in which the developer can see the hierarchical decomposition of presentation elements, a source code –view where the HTML code is displayed, and a XIML view where the specifications of the presentation model are displayed. Clicking on any presentation element in the tree view automatically highlights the corresponding sections in both the HTML code and the XIML reversed presentation model.

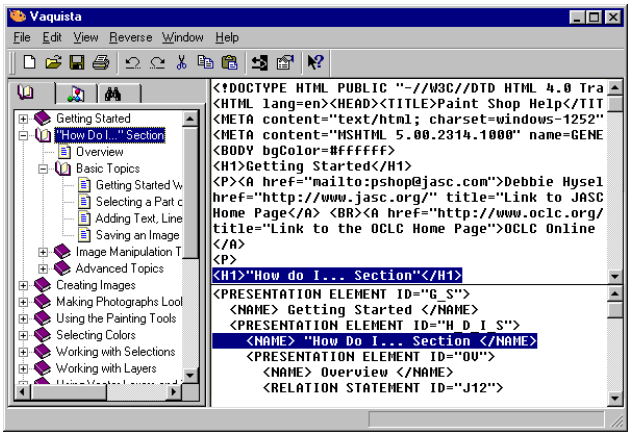

Fig. 7. Three views.

5. Implementation issues

To support the reverse engineering process outlined in section 4, several implementation alternatives were carefully considered: the Tidy tool, The Document Object Model (DOM), The World-Wide-Web Factory (W4F –), and the HTML to XHTML conversion. All of these solutions were not considered flexible enough to meet the requirements defined in the beginning. Thus, a custom application was preferred.

Due to the above reasons, the most important being the lack of flexibility in selecting matching tables (or rules) of these tools, it was decided to implement VAQUISTA in a separate development environment: Microsoft Visual Basic. However, this implementation greatly benefits from the MSXML.DLL dynamic link library (http://msdn. microsoft.com/xml/artciles/msxmlcabfile.asp). This library provides us with a large set of built-in functions to manipulate any XML-compliant document. Therefore, functions such as detection of a tag, of an element, retrieving of an associated value, etc. significantly reduced the development effort.

6. Conclusion

This work only covers the presentation model, not the navigation neither within a web page nor within a same web site. This navigation, which is specific to web, could however be also reverse engineered to come up with a dialogue model [8,18,20] in which links can be exploited. This is a future work. The current state of the work allowed us to identify some benefits and shortcomings:

- The reverse engineering process presented here can be used as a loop for redesigning a web page: an existing web page can be abstracted in its presentation model, which can be edited and used for HTML generation. This loop may improve the usability of a web page namely by considering other models, such as dialogue, user, and platform models.
- Static analysis is easy to manipulate, cheap to use, and quick to conduct, provided that the reverse engineering options are well-founded, well structured, and organized enough to structure the process properly.
- The technique presented here does not support non-HTML code such as JavaScript, Perl, CGI scripts, and embedded objects developed in a foreign environment (e.g., Flash).
- The reverse analysis is limited to a static analysis. A dynamic analysis may improve the redesign of a web page for another context of use. By enlarging the understanding of how end user interact with an interactive web page, it is likely that a redesign may be improved based on this experience.
- The static analysis is also limited in internationalization cases: web pages developed in foreign countries, in other languages and cultures, may be not interpreted correctly. Beyond the technical problem of accessing international character sets (e.g., Japanese, Korean, and Greek), potentially different options may be needed.

A final question may be raised: since the presentation model is itself stored in a XML-compliant markup language, is there a potential risk to come up at the end of the reverse engineering process with another representation of the HTML page which is only a retranscription of the HTML into another language? At first glance, the answer is yes as HTML and XML are both markup languages that can be mapped using XSL transformations. But the final answer is no since the resulting presentation model may hold more than the original HTML file (especially, when adding inferred syntactic relationships between presentation elements) or less than the HTML file (especially for any non-HTML element such as JavaScript, embedded objects, Flash or MacroMind Director multimedia sequences). Moreover, the presentation model is itself used in forward engineering by other tools that are able to recognize the relationships and to interpret them so that the model is manipulated at a higher level than merely the code level.

7. References

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, J. Shuster, "UIML: An Appliance-Independent XML User Interface Language", *Proc. of 8th World-Wide Web Conf. WWW'8* (Toronto, May 1999), Elsevier Science Pub., Amsterdam, 1999. Accessible at http://www8.org/w8-papers/5b-hypertext-medi/uiml/uiml.html

[2] T.W. Bickmore, B.N. Schilit, "Digestor: Device-Independent Access to te World-Wide-Web", *Proc. of 6th World-Wide-Web Conf. WWW'6* (Santa Clara, 7-11 April 1997), Elsevier, Amsterdam, 1997. Accessible at http://www.scope.gmd.de/info/www6/technical/paper177/paper177.html

[3] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, "Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones", *Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'2001* (Seattle, 30 March-5 April 2001), ACM Press, New York, 2001, pp. 213-220.

[4] E.J. Chikofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, Vol. 1, No. 7, January 1990, pp. 13-17.

[5] J.-M. De Baud and S. Rugaber, "A Software Re-engineering Method Using Domain Models", *Proc. of Int. Conf. on Software Maintenance* (October 1995), pp. 204-213.

[6] J. Eisenstein, J. Vanderdonckt, A. Puerta, "Adapting to Mobile Contexts with User-Interface Modeling", *Proc. of 3rd IEEE Workshop on Mobile Computing Systems and Applications WMCSA'2000* (Monterey, December 7-8, 2000), IEEE Press, Los Alamitos, 2000, pp. 83-92.

[7] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, T. Laakko, "Two Approaches to Bringing Internet Services to WAP Devices", *Proc. of 9th World-Wide-Web Conf. WWW'9* (Amsterdam, 15-19 May 2000), pp. 231-246. Accessible at http://www9.org/w9cdrom/228/228.html

[8] L. Kong, E. Stroulia, B. Matichuk, "Legacy Interface Migration: A Task-Centered Approach", *Proc. of 8th Int. Conf. on Human-Computer Interaction HCI International'99* (Munich, 22-27 August 1999), H.-J. Bullinger and J. Ziegler (eds.), Lawrence Erlbaum Associates, Mahwah/London, 1999, pp. 1167-1171. Accessible at http://www.cs.ualberta.ca/~stroulia/Papers/hci99.ps

[9] J. Lopez and P. Szekely, "Automatic Web Page Adaptation", *Proc. of CHI'2001 Workshop on Transforming the UI for Anyone. Anywhere* (Seattle, 1-2 April 2001).

[10] E. Merlo, J.F. Girard, K. Kontogiannis, P. Panangaden, and R. De Mori, "Reverse Engineering of User Interfaces", *Proc. of 1st Working Conference on Reverse Engineering WCRE'93* (Baltimore, 21-23 May 1993), R.C. Waters, E.J. Chikofsky (eds.), IEEE Computer Society Press, Los Alamitos, 1993, pp. 171-179.

[11] E. Merlo, P.-Y. Gagné, and A. Thiboutôt, "Inference of graphical AUIDL specifications for the reverse engineering of user interfaces", *Proc. of Int. Conf. on Software Maintenance* (19-23 September 1994), IEEE Computer Society Press, Los Alamitos, 1994, pp. 80-88.

[12] 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, January 1995, pp. 64-73.

[13] M.M. Moore and S. Rugaber, "Issues in User Interface Migration", *Proc. of 3rd Int. Software Engineering Research Forum* (Orlando, 10 November 1993).

[14] M.M. Moore, "Rule-Based Detection for Reverse Engineering User Interfaces", *Proc. of 3rd Working Conf. on Reverse Engineering WCRE'96* (Monterey, 8-10 November 1996), L. Wills, I. Baxter, E. Chikofsky (eds.), IEEE Computer Society Press, Los Alamitos, 1996, pp. 42-48. Accessible at http://www.cc.gatech.edu/fac/Melody.Moore/papers/WCRE96.ps

[15] M.M. Moore, "Representation Issues for Reengineering Interactive Systems", *ACM Computing Surveys*, Vol. 28, No. 4, December 1996. Article # 199. Accessible at http://www.acm.org/pubs/articles/journals/surveys/1996-28-4es/a199-moore/a199-moore.html

[16] M.M. Moore and S. Rugaber, "Using Knowledge Representation to Understand Interactive Systems," *Proc. of the Fifth International Workshop on Program Comprehension IWPC'97* (Dearborn, 28-30 May 1997), IEEE Computer Society Press, Los Alamitos, 1997. Accessible at http://www.cc.gatech.edu/fac/Melody.Moore/papers/WPC97.ps

[17] M.M. Moore and S. Rugaber, "Domain Analysis for Transformational Reuse", *Proc. of 4th Working Conf. on Reverse Engineering WCRE'97* (6-8 October 1997), IEEE Computer Society Press, Los Alamitos, 1997.

[18] A.R. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development", *Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996), J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996, pp. 19-35.

[19] F. Paternò, *Model-based Design and Evaluation of Interactive Applications*, Springer Verlag, Berlin, 1999.

[20] A.R. Puerta and J. Eisenstein, "Towards a General Computational Framework for Model-Based Interface Development Systems", *Proc. of ACM Conf. on Int. User Interfaces IUI'99*, ACM Press, New York, 1999, pp. 171-178.

[21] A.R. Puerta, J. Eisenstein, "A Representational Basis for User Interface Transformations", *Proc. of ACM CHI'2001 Workshop on Transforming the UI for Anyone. Anywhere* (Seattle, 1-2 April 2001).

[22] P. Szekely, "Retrospective and Challenges for Model-Based Interface Development", *Proc. of 2rd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996), J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996, pp. xxi-xliv.

[23] D. Thevenin and J. Coutaz, "Plasticity of User Interfaces: Framework and Research Agenda", *Proc. of IFIP Conf. on Human-Computer Interaction Interact'99* (Edimburgh, September 1999), IOS Press, 1999, pp. 110-117.

[24] J. Vanderdonckt and F. Bodart, "Encapsulating Knowledge for Intelligent Interaction Objects Selection", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 424-429. Accessible at http://www.qant.ucl.ac.be/membres/jv/publi/ InterCHI93-Encaps.pdf

[25] J. Vanderdonckt and P. Berquin, "Towards a Very Large Model-based Approach for User Interface Development", *Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99*, IEEE Computer Society Press, Los Alamitos, 1999, pp. 76-85.