Model-Based Approaches to Reengineering Web Pages

Laurent Bouillon, Jean Vanderdonckt
Université catholique de Louvain
Belgian Lab. of Computer-Human Interaction
IAG-ISYS, Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
+32-10/47.{8349, 8525}
{bouillon, vanderdonckt}@isys.ucl.ac.be

Jacob Eisenstein
University of Southern California
Department of Computer Science
941 W. 37th Place
Los Angeles, CA 90089-0781 USA
+1 213 740 4496
jacob@isi.edu

ABSTRACT
Today's web sites are increasingly being accessed through a wide variety of computing platforms ranging from the workstation to a laptop and through multiple access devices such as Internet Screen Phone, TV Set Top box, PDA, and cellular phones. Web sites are rarely de-signed and developed to fit such a large variety of contexts of use as each context (e.g., each computing platform, each device) has its own set of constraints. This pa-per describes a model-based approach for reengineering web pages into a presentation and a dialog model stored with XIML, a model-based user-interface specification language. These models are then further exploited to reengineer other user interfaces either for the same context of use (by changing presentation design options) or for different contexts of use (by changing properties of computing platform model). For this purpose, three key elements of the presentation model (i.e. presentation units, logical windows, and abstract interaction objects) and two key elements of the dialog model (i.e., navigational structure and transition) were defined.

Keywords
Model-based user-interfaces, reengineering, knowledge bases, web sites, presentation model.

INTRODUCTION
The growing interest in web site design, development, and evaluation, as well as the underlying cost, has increased the demand for engineering methods that are more structured than traditional ad hoc development or "rush to code" approaches.

It is obviously desirable that any web site should be made accessible from the widest range of computing platforms, including the variation of browsers on each platform, for the widest range of users. The rapid evolution of the HTML language itself exacerbated the need for backward compatibility, for instance, from version 2.0 to 4.0. Moreover, the technological push of newly marketed access devices—such as PDAs and WAP-enabled telephones—has generated a new 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 can only increase, as we will surely become more dependent on the web for information.

To address these demands, ad hoc development is no longer acceptable in terms of the cost and time required for software engineering and maintenance. Forward engineering [3] has been consequently considered as a good candidate for producing quality web sites. For instance, model-based approaches [22,23,26] can produce a user interface (UI) for a web site by exploiting knowledge captured in various models, such as the presentation and dialog models.

Forward engineering is said to be *simple* when one UI is generated from the initial models; when many UIs are generated, it is said to be *multiple*. However, most existing web sites have been developed without any model-based approaches, thus creating a need for reverse engineering to generate the models, which can later be exploited by future forward engineering, simple or multiple.

In this paper, we first report on what the assumptions are for adopting a model-based approach for forward engineering. Then, we describe a model-based approach for reverse engineering web pages implemented as an automatic or mixed-initiative process in the VAQUITA software, with an eye to applying forward engineering subsequently. Reengineering methods are then considered to produce new UIs for other contexts of use, thus creating a capability to rapidly produce UIs for different computing platforms, various access devices, etc.

As supporting all variations of contexts of use is very ambitious, we will focus on accommodating those features that are most likely to vary across platforms. We will then compare the above model-based approach with respect to related work and present some conclusions in a general framework for UI reverse engineering of web pages. In this framework, other forms of re-engineering will be situated with respect to the approach we adopted. We will also introduce the category of redesigning with respect to previously defined approaches.

FORWARD ENGINEERING OF WEB PAGES

For many years, the research and development community of model-based approaches has shown interest for forward engineering [23,26] with many models (fig. 1):

- A *domain model* [4] defines the objects that a user can view, access, and manipulate through a UI. In nature, it is very similar to a data model but it is also intended to explicitly represent the attributes of objects and the relationships among the various domain objects (e.g., an entity-relationship model or an object-oriented model).
- A *presentation model* [2,9,24,25] is a representation of the visual, haptic, and auditory elements provided by a UI to its users. For example, a presentation element might 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.
- A *dialog model* [8,19] defines the way in which the presentation model interacts with the user. It represents the actions that a user may initiate via the presentation elements and the reactions that the application communicates via those same elements.
- An *application model* [22] captures some abstraction of se-man-tic functions of the semantic core component called by the UI and maintains connections with it. It specifies services applications provided to the UI.

For almost a decade, models have been exploited to automatically generate a UI in two ways (fig. 1) [26]:

1. *At design time by exploiting passive models*: models are parsed to generate the UI code for a given computing platform. The generation process ranges from completely automatic (the designer does not see anything during this process) to full mixed-initiative (where the system prompts the designer for design options to decide, for control parameters). The generated code is integrated with the code of the semantic core component and compiled to obtain the intended application.
2. *At run time by exploiting active models*: the specifications contained in the models are scanned and interpreted at run-time to obtain a running UI connected to the code of the semantic core component. For instance, FUSE [8] uses an attributed grammar to abstract a UI to execute the specification contained in the models. Apart from being executable at run-time, this approach is more suitable for generating UIs depending on parameters that only become known at run-time. The alternative is to attempt to predict all alternatives at design-time and then generate a UI that supports all these circumstances. But the resulting code would be inefficient, more complex to produce, not so compact. In some cases, it may be impossible to have all UI alternatives contained into one single piece of code.

Whether at design-time or at run-time, one or many models are typically exploited to generate one UI for one application. To generate many UIs for the same application and in order to support user-centered design, the traditional model-based approach has been extended to a task-based approach (fig. 2) where two new models are introduced:

- A *task model* is a description of the task to be accomplished by the user of an application through the UI. Individual elements in this model represent specific actions that the user may undertake. Information regarding subtask ordering (e.g., sequence, unordered) as well as conditions on task execution is also included.
- A *user model* represents the different types of users of an application. It defines attributes, roles of users, and may include information describing their preferences.

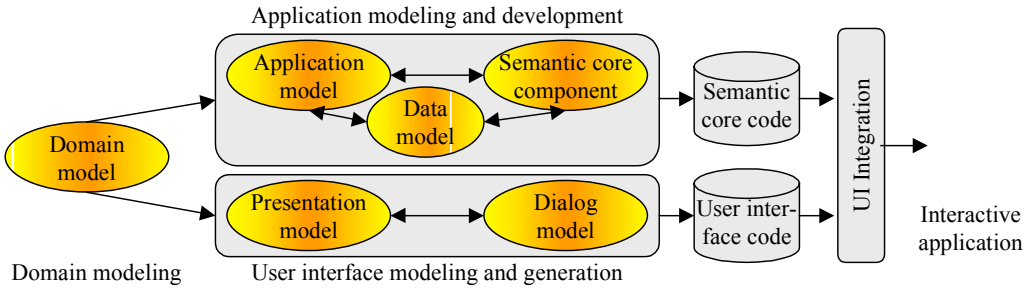

Fig. 1. Traditional model-based approach.

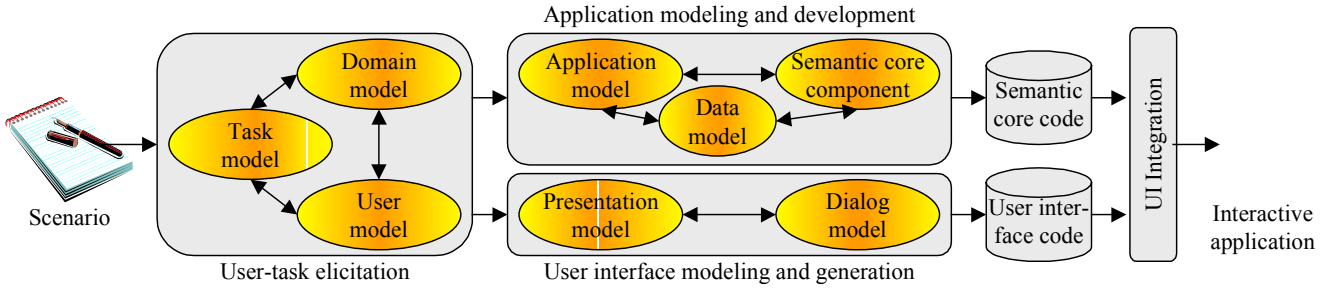

Fig. 2. Task-based approach.

REVERSE ENGINEERING OF WEB PAGES

Several environments today support forward engineering of Web pages as presented in Section 2. For example, Allegro Serve (http://allegroserve.sourceforge.net) is a web server that can dynamically generate Web pages and web-enable existing applications with a browser front-end. However, as most existing Web sites were not built according to such an approach, adopting a reverse engineering of these web pages is a required preliminary step to further benefit from any future forward engineering. The goal of reverse engineering here is to recover both presentation and dialog models of a set of web pages (fig. 3). The UI code should be extracted from the global code and separated from the code of the semantic core component. From this UI code, presentation and dialog models need to be reverse engineered. For this purpose, relevant abstractions need to be defined for each of these two models.

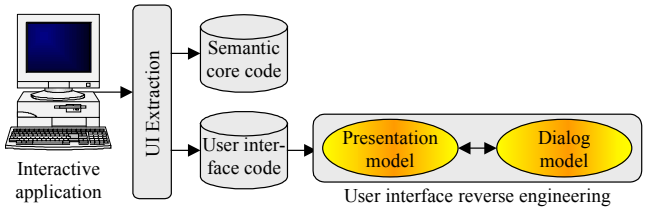

Fig. 3. Reverse engineering.

Abstractions for the Presentation Model

UI presentation of web pages can be abstracted using four concepts, the hierarchy of which is depicted in fig. 4:

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.
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 success-fully for both forward [8,9,20,24] and reverse engineering [10,13,14]. 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].
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 de-composed 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.

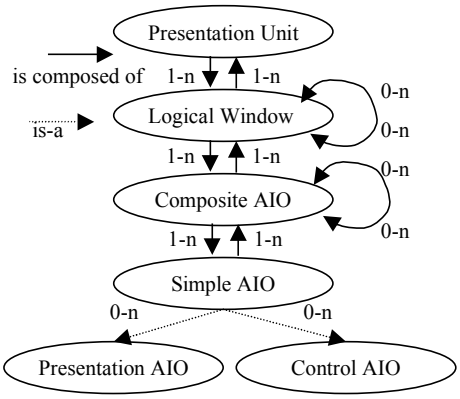

Fig. 4. Hierarchy of presentation concepts.

Abstractions for the Dialog Model

Web sites are typically developed within a graphical or textual editor according to a page & link approach: the presentation of a family of web pages is first designed and the links are added as this design proceeds. The abstraction level of these design concepts can be raised to more encompassing concepts of a transition and navigational structure. A transition is defined as any control capability enabling users to move from a source LW to a target LW. It consequently consists in a control AIO (e.g., an anchor, an icon, a push button) allowing a directional control (e.g., unidirectional or bi-directional) between LWs with operations on these LWs: maximization, titling, minimization, display with tiling technique, display with normal overlapping technique, display with user-defined overlapping technique, display with system-defined overlapping technique, or closing. A navigational structure applies any graph pattern between LWs of a given PU of a given type.

Sequential navigation. This navigation structure enables a user to move between LWs according to a pre-defined sequence. In this structure, unidirectional transition allows a user to move from one LW to another (fig. 5), while bi-directional transition allows a user to switch between a set of LWs, these LWs being closed, restored or redisplayed.

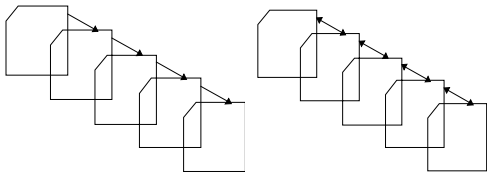

Fig. 5. Unidirectional and bi-directional sequences.

Indexed navigation. This structure is aimed at providing a regular index mechanism in which multiple target LWs are suggested from one single source LW. Having unidirectional transition prevents the user to come back to the source LW, while bi-directional transition will (fig. 6).

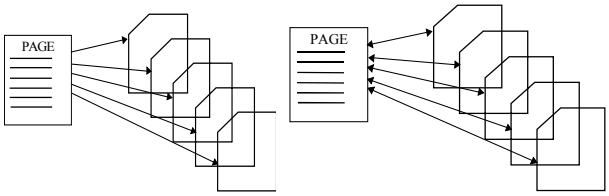

Fig. 6. Unidirectional and bi-directional indexes.

Guided navigation. Guided navigation is similar to a guided tour: a source LW suggests the user a first LW to switch to, this first LW is connected to a second one, and do forth to the last LW suggesting the user to come back to the source LW. In some sense, this navigation is similar to a sequential navigation where the dialog is closed. Unidirectional transition only allows the user to move to the connected LW one way, whereas bi-directional transition allows a round trip (fig. 7).

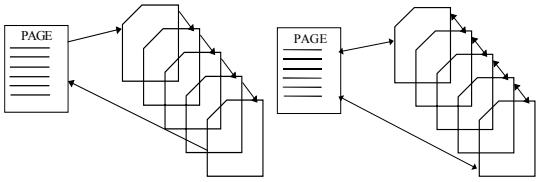

Fig. 7. Unidirectional and bi-directional guided tours.

Mixed navigation. This navigation type groups two previously defined abstractions into a new one: a guided navigation is augmented by an index to combine navigation flexibility provided by an index and guidance provided by a guided tour. This type therefore allows a user to be guided in a guided tour and leave the tour or reinitiate it at every step (fig. 8).

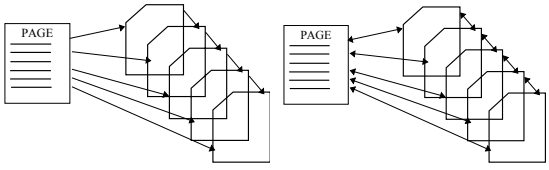

Fig. 8. Unidirectional and bi-directional mixed tours.

Global navigation. This navigation type is considered to be the most complete as it allows the user to move from any LW to any other one. Bi-directional transitions are therefore considered not useful as they reproduce the same transition (fig. 9).

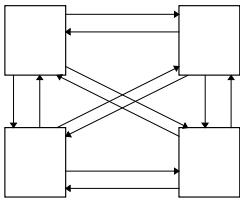

Fig. 9. Global navigation.

Using transition and navigation structures as basic concepts not only raise the abstraction level with respect to the traditional page & link model, but also enables designers to manipulate entire navigation structures and to combine them in a more logical way. Rather than modifying transition controls on each LW, a collection of behaviorally-equivalent transition controls is established and maintained across a series of related LWs. Navigation structures can be defined in isolation or directly combined by referring to a particular LW inside their definition.

In fig. 10 for instance, the designer can first decide to have a sequential navigation A across a family of LWs named A1 through A4. Any LW of this navigational structure can then be a good candidate to form the source LW for any other type of navigational structure. For example, the logical window A3 can be designed as the source LW for an unidirectional indexed navigation. Any combination of the basic navigational structures can be imagined. However, some observed structures do not exactly match these basic or combined structures. In this case, the closest combination is selected instead, thus introducing some modification in the original navigation.

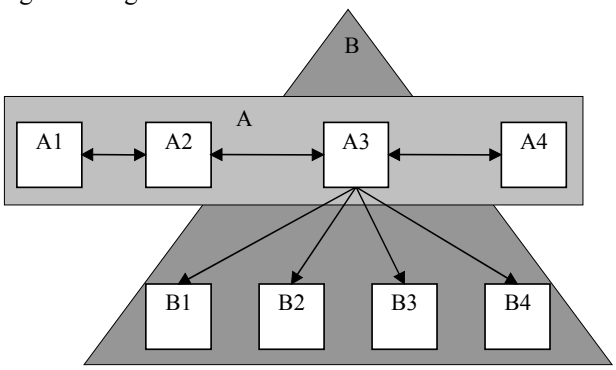

Fig. 10. Combining navigation structures.

REVERSE ENGINEERING ALGORITHM

Having defined abstractions relevant to presentation and dialog models, we can exploit them to reverse engineer these abstractions for a series of web pages. This process basically consists in (i) downloading the HTML code of Web pages of interest, (ii) extracting from their code the UI part and separating it from the part belonging to the seman

tic core component, and (iii) submitting it to the static analysis algorithm for reverse engineering depicted in fig. 11. Thanks to the HTML code accessibility, *remote* reverse engineering of Web pages is permitted; in addition, the design of automated tools for reverse engineering should be much easier for web pages than for compiled program code in C or Pascal. The phases of this algorithm are now detailed in the subsequent subsections as they are considered in VAQUITA (http://www.isys.ucl.ac.be/bchi/research/vaquita.htm), a tool that enables designers to reverse engineer a web page into one or several presentation models. Currently, only the presentation model is reverse engineered.

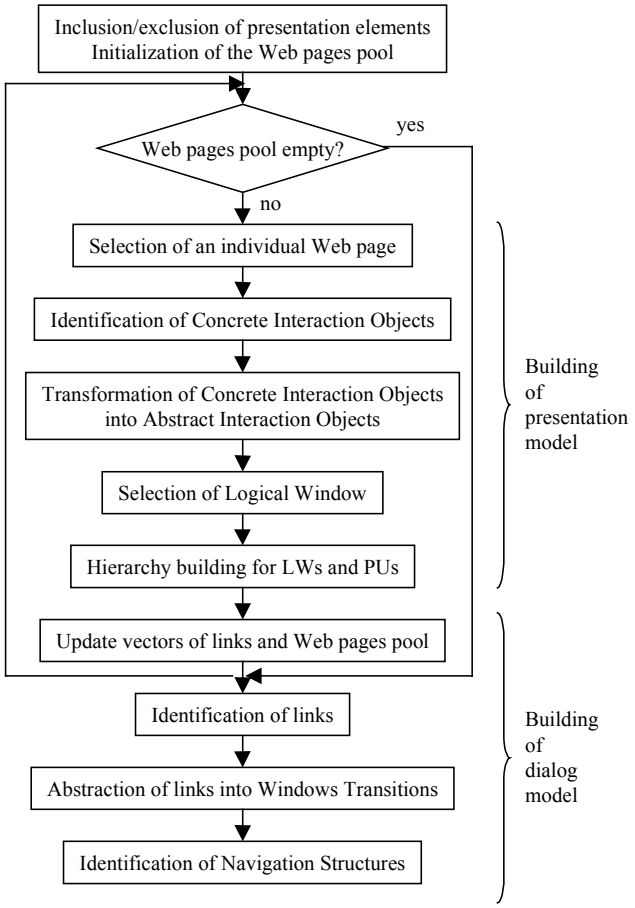

Fig. 11. Reverse engineering algorithm.

Initialization. To initiate the process, the designer specifies whether she wants to reverse engineer a whole web site (by providing the home page's URL), any portion of a site (by providing a starting URL and the depth level up to which pages will be considered) or a series of web pages (by entering their URLs locally and/or from the Web). A first pool of candidate pages' URLs is formed.

To allow more flexibility in the reverse engineering process, the designer can include or exclude not only any HTML tag or element, but also any attribute of each tag. To avoid reparametrization, this setting canbe saved in a configuration file for future usage.

Presentation modeling. An individual web page is first selected in the pages' pool and then submitted to identification of CIOs. The basic idea of the algorithm is to identify individual HTML elements, to map them onto relevant AIO, while building a hierarchy of AIOs. The HTML code of the web page is parsed to identify these HTML elements. The mapping is based on a set of heuristics that depend on the HTML element or tag analyzed. Examples of such heuristics include:

- Each web page is mapped onto a LW in the selection of LW phase. The <TITLE>, if any, is mapped onto the label slot of this AIO. Any HTML element can be mapped to one or many AIOs with one or many properties included.
- Heading levels (e.g., H1, H2, H3) are mapped onto respective composite presentation AIOs and to some structure in the progressively forming hierarchy. The text provided within the <H>...</H> scope is mapped onto the Label abstract attribute of the AIO.
- Each <FORM> tag is mapped onto a composite control AIO. Multiple or embedded forms generate multiple composite AIOs on corresponding hierarchy levels. <TR>, <TD> tags denoting tables are treated similarly.
- Web page's frames are mapped onto separate LWs as they are themselves partial views on other pages.
- Static banners, icons, graphics, illustrations, and GIF files are listed in the "images" category of a "Graphics" simple presentation AIO; animated GIFs or video are listed in its "animation" category, while JPEG files are believed to relate to the "photograph" category.
- Portions of text are equally treated like labels, with their static properties being fed into the abstract attributes of the AIO, e.g., the currently active BGCOLOR is mapped onto BackGroundColor and TEXT is mapped onto ForeGroundColor. When text is subject to an anchor, an additional control AIO is created.
- HTML proprietary CIOs are mapped onto their corresponding simple control AIOs. Examples include the <SELECT> and <TEXTAREA> tags, and the TYPE attribute of the <INPUT> tag. The NAME tag is mapped onto an identification label for the related AIO. The content of the VALUE tag is mapped onto the abstract attribute DefaultValue and so forth.

Dialog modeling. To reverse engineer a dialog model and finding out instances of transitions and navigational structures, a vector of links is created for each page according to the following format:

μ(URL)={list of intra-page links, list of extra-page links}

where each link has the form:

ℓ = (*AIO_name*, *target_URL*, *link_type, occurrence*)

where *AIO_name* = identifier in the presentation model of the AIO holding the link source considered
target_URL= URL of the page linked
link_type = (intra-page, extra-page, request)
occurrence = amount of same links in the page

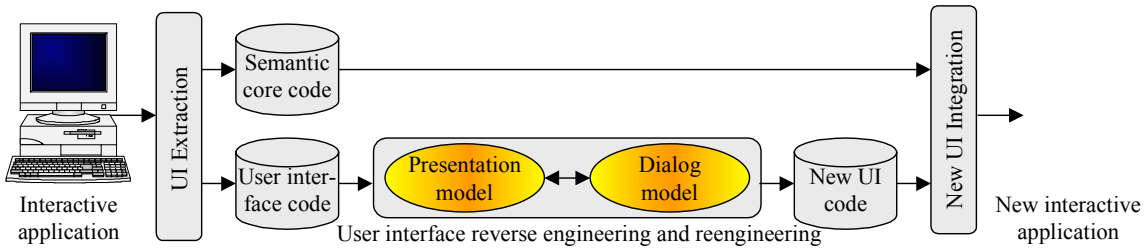

Fig. 12. UI Reengineering

Once any individual page has been processed regarding to its own presentation model and regarding to its own vector of links, this vector is parsed to update the pool of candidate web pages to analyze. In the identification of links, this analysis exploits positive and negative filters defined in the initialization. For example, all external links or all extra-page links up to a certain depth level are omitted. The analyzed web page is then withdrawn from the pages' pool. The pool is examined to find a candidate recursively until no candidate page still exists. After exhausting the whole pages' pool, the remainder of both dialog and presentation models is reverse engineered. First, vectors of links that have been built for the collection of web pages in consideration are examined to identify LW transitions from each pair of LWs. Transformation rules include:

- Any link with multiple occurrences within the same page is reduced to a single link to avoid repetition.
- All links with multiple sources within the same page (e.g., a textual link, a push button, an icon) to the same target page are gathered. Depending on parameters, all instances of such links can be kept or reduced to any particular type. For instance, links can be restricted to only one occurrence of anchor and one occurrence of icon. In this case, the presentation model can combine both AIOs into a single one, of the "icon" type, whose label is the anchor text. Again, positive and negative filters can automatically expand or reduce the scope of AIOs in concern, for example, considering only push buttons and forgetting all other types of AIOs for links.
- Any one-way link is transformed into a unidirectional transition. Depending on the link type, the operations on the source and target LWs are specified:
  1. When a link replaces one page by another, the transition will contain a "close" operation on the source LW and a "maximize" operation on the target LW.
  2. When a link just pops up another page, the transition will contain only a "display with normal overlapping" operation on the target LW.
  3. When a link pops up another page in a constrained window, the transition will contain a "display with system-defined overlapping" on the target LW.
- Any pair of reciprocal links between two pages is transformed into a single bi-directional transition between the two LWs.
- Any intra-page link is transformed into a control AIO branching to the related composite AIO belonging to the same LW. Normally, intra-page links are intended to fasten visualization and avoid scrolling rather than providing LW transitions. This information is still re-corded as, in the future, these AIOs may be replaced by something else, for example another LW. In this design option, parts of a web page that are accessed by intra-page links may be replaced by separate LWs.

Transitions between logical windows abstracting these links are subsequently stored in a matrix. Each matrix column denotes a source LW, while each matrix line denotes a target LW. Thus, any cell denotes a window transition between a pair of windows.

The resulting matrix is then passed to a pattern-matching algorithm that is in charge of identifying navigational structures among the LWs. This algorithm begins by guessing the simplest navigational structures (such as sequential or indexed navigation) and continues with more rich structures (such as guided or mixed navigations). As the window transition matrix can be rather sparse or non regular, finding the exact materialization of these navigational structures can be infrequent.

The next step is to generalize the existing structure to the closest abstract structure type. Of course, each existing structure can be generalized into a global navigation structure. But this network of LWs probably generates too many window transitions. In order to moderate this generalization process, a threshold can be specified. This parameter can, for example, regulate how many transitions can be added in order to match to the closest strict navigational structure. As this algorithm may be incomplete, the designer is allowed to check the resulting navigational structures. The designer may accept, reject or modify the structure produced the algorithm, and she also may declare undiscovered structures. This concludes the building of the dialog model.

RE-ENGINEERING OF WEB PAGES

Now that presentation and dialog models are available from reverse engineering of web pages, these two models can theoretically benefit from forward engineering: any model-based approach should be able to generate a new UI from these two models to be further re-integrated with the semantic core component. Before reengineering web pages into a new UI according to the process depicted in fig. 12, it is important to specify the context in which this new UI will

take place. As indicated in the introduction, technological push and user pull are both increasing the demand for accessing web sites in various contexts of use. These variations may depend on:

1. *User*: several users will access the site, each user belonging to a specific stereotype of the user population. Each stereotype ha its own set of parameters, such as preferences, customized features, interaction history, cognitive profile, typing vs. selection skills, motor-, sight-, or auditive-impairment, and specific needs.
2. *Computing* platform: many different computing platforms can be used to access a site, and each platform brings its own set of constraints, such as screen resolution, number of colors, color palettes, planes, UI language (programming language, script language or tab)based language are very different), network, and interaction devices. These two last constraints possess themselves further constraints such as network bandwidth, interaction devices capabilities and availability.
3. *Environment*: users access a site in different environments. Ambient conditions add a new set of constraints, including physical parameters (e.g., noise, light, working space room, heat) and psychological, social parameters (e.g., stressful time slot, productivity, task criticality, unstable environment, collaborative or cooperative conditions, home vs. work).

Reengineering web pages to support all these variations of the context of use would be an ambitious task. As we assumed in the introduction to mainly address cross-plat-form support, this paper is focused on adding a computing platform model only. We feel that techniques that are found to be useful for addressing platform constraints will also be able to handle constraints that come from the user and the environment. The rapid proliferation of various platforms for web accessibility makes this a particularly timely concern. Users increasingly want to compute while moving between locations, thus moving from one platform (e.g., a Windows PC) to another (e.g., a Windows CE-compatible device). Other users do not want to move from one place to an-other, but want to have multiple access from a single place (e.g., connecting a PDA to a PC).

Many parameters vary across platforms. VAQUITA only support screen resolution variation for one UI language (i.e., HTML) as summarized in table 1. The concepts remain the same without loss of generality.

Context	Resolution	Language	Approaches
❶	Fixed	Fixed	Reengineering
❷	Fixed	Varying	Redocumentation
❸	Varying	Fixed	Restructuring and redocumentation
❹	Varying	Varying	Restructuring

Table 1. Variations of considered parameters.

Context ❶ with no parameter varying

In this context, a new presentation model can be inferred from the existing models by considering at least 5 designs:

1. Redistribution of LWs within a single PU;
2. Redistribution of Composite AIOs within a single LW;
3. Redistribution of Simple AIOs within a single LW;
4. Redistribution of all AIOs within a single LW;
5. Redistribution of all AIOs within a single PU.

Individual AIOs should remain unchanged, but any redistribution of these elements can be imagined provided that the screen constraints are satisfied. To verify these constraints, the presentation model needs to be enriched by two AIO abstract attributes: typical height and length. These attributes are assigned to an average value in pixels for dimension-fixed widgets (e.g., a check box, a radio button).

These attributes are assigned to a value that equals a multiplying coefficient in pixels times the character length for dimension-varying widgets (e.g., an edit box, a list box). Other dimensions are inferred or estimated from the original CIOs themselves (e.g., an image thanks to the HEIGHT=x, WIDTH=y tags.) Analyzing all possible above reconfigurations is beyond the scope of this paper. We refer to [2,25] for analyzing redistribution of LWs within a same PU (i.e., minimal, maximal, input/output, functional, user defined, grouped, ungrouped).

Context ❷ with varying UI language

In this context, only the underlying UI language changes. This often occurs when multiple computing platforms running different operating systems and presentation managers are considered. Some languages are Internet-compatible (e.g., HTML, Java, WML), some are not (e.g., Visual Basic, C++). Beyond redistributions described in Section 4.1, this context poses a new challenge: all these languages do not share similar capabilities in terms of CIOs of the source computing platform.

For instance, an AIO may be mapped onto zero CIOs (the corresponding CIO does not exist in the target language), one (the most frequent and easy case where a one-to-one relationship exists between an AIO and its corresponding CIO), or many CIOs (several CIOs are required to reproduce the same behavior of the intended AIO). To allow these mappings, a platform-specific transformation table allows any AIO instance to be assigned to corresponding CIOs.

When no corresponding CIO exists, an alternative CIO is proposed (ultimately, the edit box is suggested), but can be tailored by the designer. Nevertheless, most access devices or specific computing platforms coming with their own proprietary language also impose some screen resolution change. For instance, Wireless Markup Language (WML) can only be used on cellular phones; Voice XML, only on with a screen reader. A third context is then considered.

Context ❸ with varying screen resolution

It is a property of HTML that many HTML-compliant browsers on many types of screen can display it. Although this display is technically feasible (i.e., the browser adapts the presentation with respect to the platform), it may be *unusable* to the end-user for one or another reason, e.g., loss of information structure, overcrowded screen, too many scroll bars, image reduction). Therefore, it makes sense to consider how to re-engineer the UI for different screen resolutions. This need is exacerbated by multiple HTML browsers

- On different platforms equipped with different monitors (ranging from workstation screen to small laptop)
- On different access devices with built-in browser (ranging from the Internet ScreenPhone with 640x480 resolution to HTML-compatible PDA).

If the screen resolution is increasing, the centered table with three columns can be used to keep a presentation centered on the screen with borders (fig. 13). If the resolution is decreasing, the above redistributions are no longer applicable. Three strategies have been explored so far:

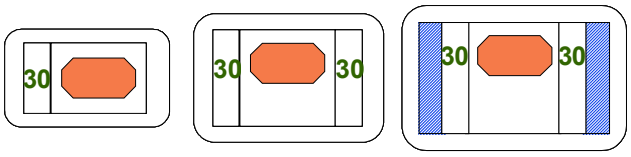

Fig. 13. Presentation centered on the screen.

1. *AIO replacement*: each AIO definition is augmented with a list of potential replacements considered as either behaviorally equivalent, but consuming less space (e.g., a list box to a drop-down list box) or degraded (e.g., an accumulator to a multiple-value list box).
2. *AIO size reduction*: typical height and length of AIOs are submitted to a possible reduction, depending on usability constraints. For example, images and icons can be reduced to a certain limit specified by a threshold, e.g., the minimally usable size of icons.
3. *AIO ungrouping*: when individual AIOs cannot be reduced according to the two first strategies, the distribution of AIOs may be reconfigured. AIOs can be grouped or ungrouped and new LWs can be created to contain a limited amount of AIOs.

Again, the development of these strategies bears further explanation, which we hope to conduct in a later paper.

Context ❹ with varying language and resolution

This context is most likely to occur when a completely different computing platform of access device is considered. For example, a cellular phone only supports WML, WebTV only accepts a subset of HTML V.3, and screen readers only accept VoiceXML. This context can be handled using a combination of the approaches described in previous contexts.

RELATED WORK

Cross-platform development is not new as several environments provide support for this purpose: Galaxy [6] and Open Interface [17] render the same UI on different platforms with their native look & feel, while SUIT [18] employs a unique UI definition that can be processed on multiple platforms. However, not one of these systems truly adopts a model-based approach, although SUIT's common definition holds some presentation abstractions. CT-UIMS [9] pioneered the platform model by supporting some AIO [15] redistribution for OSF/Motif large screen and small Macintosh screens. AUIDL [10,11,12] is probably the first set of abstractions for reverse and re-engineering UIs: from internal hierarchical structures, type and variable declarations, a UI can be recovered in IDL with a presentation model (based on OO paradigm) and a dialog model (based on Milner's process algebra). MORPH exploits production rules [13] to infer AIOs [14] from CIOs and thus produces a graphical UI from a textual UI [15]. This transformational approach is similar in principle to ours. Forward engineering can be executed by model composition [20], binding [21] or derivation, thus proving that the approach is feasible. MORALE [1] is an extensive set of techniques and tools for reverse engineering legacy systems, rather than web pages. CELLEST [7] reverse engineers similar systems, but into DHTML, thus adopting active models, rather than passive models.

CONCLUSION

This paper has introduced a model-based approach supporting both reverse and re-engineering of web pages. The architecture outlined in fig. 14 assumes that all models are stored in a model repository, each model being specified with the eXtensible Interface Markup Language (XIML) promoted by the XIML Consortium [28]. This model textual specification is declarative, analyzable, and editable [22].

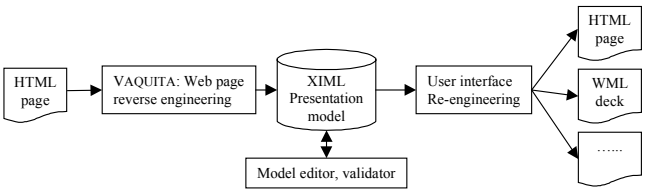

Fig. 14. Model-based approach architecture.

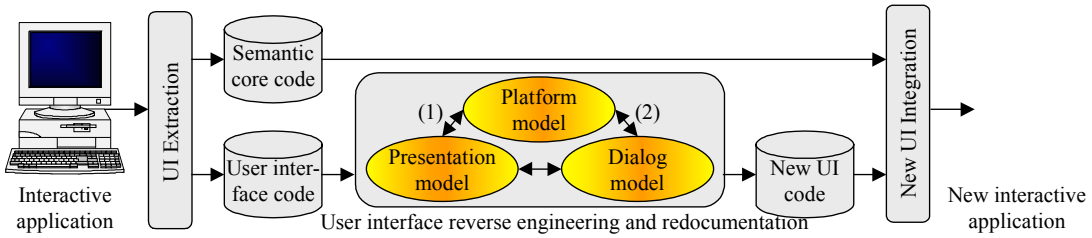

Fig. 15. UI Redocumentation

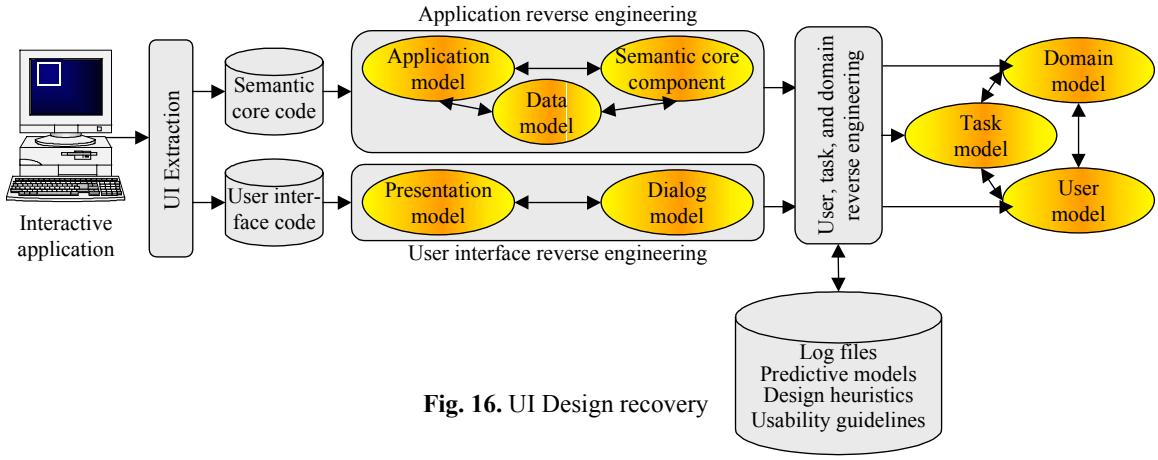

Fig. 16. UI Design recovery

This ongoing work is just at the beginning, as it will require a significant amount of work to create a comprehensive environment with tool-support for model-based reengineering of web sites. The described approach inevitably suffers from many restrictions and lack of support such as:

- Non-standard HTML elements (e.g., embedded objects, browser-specific tags) or languages (e.g., JavaScript functions, ActiveX controls) are ignored as their support would require expensive development efforts.
- Dynamically created web pages are not supported since their HTML code is generated on the fly. The callback routines attached to AIOs producing dynamic pages can be replaced by direct calls to the semantic core. Active models are in this case too expensive to manage re-engineering on the fly. However, we can imagine to partially support applications that use explicit template-based page generation (e.g., XSLT, JSP): in this case, the static analysis, should be able to differentiate variable parts from run-time provided parts.
- Style sheets are unsupported in the outlined algorithm, as are other relevant presentation aspects. For instance, graphical images designed as tabs (e.g. on www. amazon.com) or links placed on top of such graphics are not recognized and, therefore, cannot be mapped onto several LWs of a same PU.

On the other hand, the environment we have described may envision other forms of UI reverse engineering [3] as represented in table 1:

- *Redocumentation* (fig. 15): this flow is similar to re-engineering, except that the platform model is no longer needed, as it remains constant over time.
- *Restructuring* (fig. 15 with arrow (1) used only): the UI remains basically the same for semantics and dialog, but the presentation model changes. The scope of the platform model is restricted to re-engineering.
- *Redesigning* (fig. 15 with arrows (1) *and* (2) used): the UI remains identical regarding its semantics, but both the dialog (e.g., other interaction styles, techniques, or genres) and the presentation (e.g., any redistribution) models can change due to platform constraints. This may consists in a new category of UI re-engineering.
- *Design recovery* (fig. 16): in this process, it is expected that task and domain models could be recovered. Guessing these models from HTML code seems impractical, but other information sources might be investigated, such as log files produced by user traffic, comparison with predictive models, design heuristics to identify usage patterns, and automated usability analysis based on guidelines.

REFERENCES

[1] Barclay, P.J., Griffiths, T., Mc Kirdy, J., Paton, N.W., Cooper, R., Kennedy, J.: "The Teallach Tool : Using Models for Flexible User Interface Design". Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 139–158. Accessible at http://img.cs.man.ac.uk/teallach/publications/Cadui99/CADUI99.ps

[2] G. Abowd, A. Goel, D.F. Jerding, M. McCracken, M.M.

Moore, J.W. Murdock, C. Potts, S. Rugaber, and L. Wills, "MORALE–Mission Oriented Architectural Legacy Evolution", *Proc. of Int. Conf. on Software Maintenance* (1997)

[3] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, and J. Vanderdonckt, "Computer-Aided Window Identification in TRIDENT", *Proc. of the 5th IFIP TC13 Conf. on Human-Computer Interaction Interact'95* (Lillehammer, 25-29 June 1995), Chapman & Hall, London, 1995, pp. 331-336. Accessible at http://www.info.fundp.ac.be/cgi-publi/pub-spec-paper?RP-95-021

[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 and A. Puerta, "Adaptation in Automated User-Interface Design", *Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'2000* (New Orleans, 9-12 January 2000), ACM Press, New York, 2000, pp. 74-81.

[7] "Galaxy Application Environment", Ambiência Information Systems, Inc., Breckenridge, 2000. Description accessible at http://www.ambiencia.com/galaxy/galaxy.htm

[8] L. Kong, E. Strouila, 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] F. Lonczewski and S. Schreiber, "The FUSE-System: an Integrated User Interface Design Environment", *Proc. of 2nd Int. Workshop on Coputer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996)*, Presses Universitaires de Namur, Namur, 1996, pp. 37-56. Accessible at ftp://hpeick7.informatik.tu-muenchen.de/pub/papers/sis/fuse_cadui96.ps.gz

[10] Ch. Märtin, "A UIMS for Knowledge Based Interface Template Generation and Interaction", *Proc. of Interact'90*, Elsevier Science Pub., Amsterdam, 1990, pp. 651-657.

[11] 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.

[12] 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.

[13] 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.

[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] "Open Interface™", Neuron Data, 156 University Avenue, Palo Alto, CA 94301, 1992.

[19] R. Pausch, M. Conway, and R. DeLine, "Lessons Learned from SUIT, the Simple User Interface Toolkit", *ACM Trans. on Office Information Systems*, Vol. 10, No. 4, October 1992, pp. 320-344. Accessible at http://www.cs.virginia.edu/~uigroup/docs/publications/Suit.lessons.paper. ps

[20] A.R. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development", *Proc. of the 2nd Int. W. on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996), Presses Universitaires de Namur, Namur, 1996, pp. 19-36.

[21] R.E.K. Stirewalt and S. Rugaber, "Automating UI Generation by Model Composition", *Journal of Automated Software Engineering*, Vol. 7, No. 2, 1998, pp. 101-124. Accessible at http://www.cc.gatech.edu/reverse/repository/gener.ps

[22] R.E.K. Stirewalt, "MDL: A Language for Binding User-Interface Models", in [26], pp. 159-184.

[23] P. Szekely, P. Luo, and R. Neches, "Beyond Interface Builders: Model-Based Interface Tools", *Proc. of ACM Conf. on Human Aspects in Computing Systems InterCHI'93*, ACM Press, New York, 1993, pp. 383-390.

[24] P. Szekely, "Retrospective and Challenges for Model-Based Interface Development", *Proc. of 3rd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996), Presses Universitaires de Namur, Namur, 1996, pp. xxi-xliv.

[25] J. Vanderdonckt and F. Bodart, "Encapsulating Knowledge for Intelligent Interaction Objects Selection", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 424-429.

[26] 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.

[27] J. Vanderdonckt and A. Puerta (eds.), *Proc. of the 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99*, Kluwer Academics Publishers, Dordrecht, 1999.

[28] The XIML Consortium, http://www.ximl.org, 2002.