A Tool for Creating Design Models from Web Site Code

L.Paganelli, F.Paternò
C.N.R., Pisa
Via G.Moruzzi 1
{laila.paganelli, fabio.paterno}@cnuce.cnr.it

ABSTRACT
In this paper we present a method and the related tool for analysing Web site code in order to automatically reconstruct the underlying logical interaction design. Such design is represented through task models that describe how activities should be performed to reach users' goals. The models also include a specification of the objects that should be manipulated to accomplish such tasks. We also discuss how the result of this reverse engineering process can be provided as input to a number of tools for various purposes (model analysis, usability evaluation, user interface redesign).

Keywords
Human-Computer Interaction, Reverse Engineering, Web Applications, Task Models.

INTRODUCTION
Nowadays, creating a Web site takes little effort. There are many tools that support automatic generation of Web pages from many possible formats. However, when we navigate the Web we often encounter problems finding the sought for information or accomplishing the desired tasks. This reveals how difficult it still really is to obtain usable Web sites.

Task models describe how activities should be performed in order to reach users' goals. They are the link between user interface designers and developers as they provide an abstract description able to highlight important information for both that can be used to analyse and discuss design solutions. They can also be used to support usability evaluation via various techniques (for example, to predict task performance, to compare predicted use with actual use, represented by logged sessions).

However, developing task models, as many modelling activities, can require some effort especially when large applications are considered. In addition, designers often have to analyse and evaluate applications developed by others without any logical representation of the design choices.

Often, when considering design of user interfaces researchers have paid attention to top-down approaches, whereby the designer first thinks about an abstract design and then moves on to the concrete design and lastly to the implementation. As already mentioned, in some cases, particularly Web applications, there is also a need for an inverse process: starting with an existing application developed by somebody else, designers have to reconstruct the underlying design decisions in order to better analyse them and propose an improved design.

After some preliminary results [9] we are now able to present a systematic method and the associated tool able to support the possibilities of bottom-up approaches. This can be a useful complement to methods and tools able to support user interface generation from task models.

We also discuss how the result of this reverse engineering process can be provided as input to a number of tools for various purposes (model analysis, usability evaluation, user interface redesign).

RELATED WORKS
So far, little attention has been paid to support the development of models representing the design of interactive systems. Some approaches have addressed the issues related to how to derive such models from informal material used in the design phase, such as textual descriptions of scenarios of use. An example in this area is U-Tel [4], a tool supporting automatic identification of nouns and verbs that are then used as input for the domain and the task model, respectively, on the assumption that nouns indicate the objects that compose the domain model and verbs the activities considered in the task model. A similar approach has been pursued in [2], where shallow natural language parsing is supported to automatically extract task information from narratives. In our case we want to address a different issue: how to identify the task model starting with the Web pages composing the site considered.

A tool with more similar goals is Critique [6], which aims to support the development of KLM models [3] from user session logs. KLM models have a hierarchical description of sequential activities where the basic elements are the actions. Such models are also used to predict task performance on the basis of some cognitive studies that indicate estimated time to perform the types of actions

considered. The rules for identifying the types of actions from elements of user interaction logs are straightforward. Those for chunking the actions in order to identify the corresponding higher-level task are more elaborate. To this end, the rules proposed create a new chunk when users begin working with a new interactive objects or start to provide a different form of input to the current object (e.g. switch from clicking to typing in a text box). While this approach can provide useful information, it seems rather limited because the resulting model will reflect the actual use made by the user and moreover has no rule able to identify general temporal relations among tasks (apart from sequentiality) and, in any event, does not address the specific aspects of Web applications.

Vaquita [1] addresses Web applications but is limited to identifying presentation models for single pages, which mainly means the abstract description of the interaction techniques used in the implementation. This tool is therefore unable to reconstruct the task model associated with the Web application considered.

Mathaino [11] provides support for reverse engineering of interaction plans for legacy interface migration. It is obtained by starting with an analysis of a collection of traces of the interactions between the system and the user. The result is a model of the underlying interaction plan that can be used to drive the redesign of the user interface for a different platform. In this approach one potential limitation is that it mainly focuses on sequential activities, whereas in Web applications often there is the possibility of choosing from various options to accomplish the tasks.

This review of some of the previous approaches to supporting reconstruction of relevant models for interactive systems design highlights the current lack that our work aims to fill: supporting reconstruction of task models of existing Web applications.

In the paper we first introduce the method that we have developed, next we provide a description of the rules that we have identified and implemented in the associated tool. Then, we move on to illustrate an example of applications of the method in order to clarify the approach. Lastly, we discuss how the resulting models can be used to support analysis of the current design, potential redesign for other platforms, and usability evaluation through another tool that we have developed and provide some concluding remarks.

THE APPROACH

The purpose of this work is to automatically reconstruct the task model of the Web application considered. In particular, we consider task models represented in ConcurTaskTrees. It is not the purpose of this paper to describe this notation, which is presented elsewhere [10]. Thus, we will briefly illustrate its features in order to allow readers unfamiliar with it to follow what is presented here. In this notation activities are represented in a hierarchical manner (as is the case for several task model notations). It uses different icons to represent task allocation (whether a task should be performed by the system or the user or their interaction) and has a rich set of temporal relationships that can be used to describe flexible and dynamic behaviours. These operators will be introduced during the explanation of the rules for the reverse engineering transformation. In addition, the notation has some features such as the possibility of describing the objects manipulated during task performance or task attributes (such as frequency, pre-condition and so on).

The tool (see Figure 1) receives as input the Web pages of the site. The site can be either in the local system or remote. The tool is able to automatically identify the pages composing it. Using the classes provided by Tidy [12] it first check that the HTML code is well-formed and if not it corrects it and then creates the corresponding DOM [5] which describes the structure of the page (all the elements contained in it). Then, following the rules that we have developed and are illustrated in the next section it creates the task model associated with each page. The final part of the underlying algorithm is dedicated to describe higher-level tasks that involve multiple pages. The resulting task model can be saved either in XML format or in a format that can still be subjected to modifications or adjustments by the designer using tools available for this purpose.

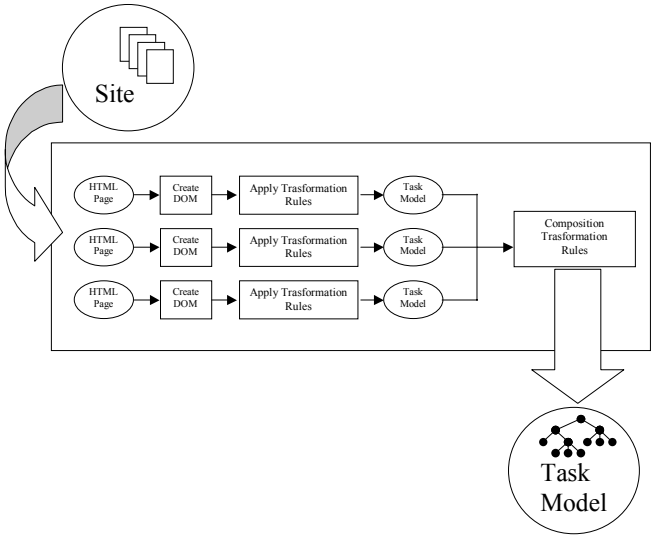


Figure 1: The structure of the tool.

RULES for BUILDING TASK MODELS of SINGLE PAGES

In this section we describe the rules developed to reconstruct the underlying task model. We also aimed at identifying meaningful names for the tasks identified.

In HTML documents we can find many interaction elements (Links, Buttons, CheckBoxes, Radio Buttons, …) and elements for their logical grouping (Forms, Fieldsets..). Our rules analyse all the main elements that can be used in HTML Web pages and aim to identify the corresponding tasks, their mutual relationships in order to build the task model corresponding to the Web site considered.

Creation of initial task structure associated with a web page

When a new page is considered the tool starts to create a structure such as that in Figure 2. It represents the typical interaction pattern that is supported by all Web pages. There is first the loading of the page, followed by some interaction that can be interrupted by the selection of a link to another page. The task name of the root is *"Access"* + *Title* of the page (in case the title is not included we use the URL to identify the page). The first subtask is a basic application task (application tasks are represented by the computer icon), associated with the activity of loading the page in the browser and the name *"Load"* + *Title* of the page. It is followed by an enabling operator (>> operator) and an abstract task named *"Handle"* + *Title* of the page (abstract tasks are represented by the cloud icon and indicate tasks that can be decomposed into more basic elements).

In particular, this task will be decomposed later on with the description of the activities that can be performed once the page has been loaded.

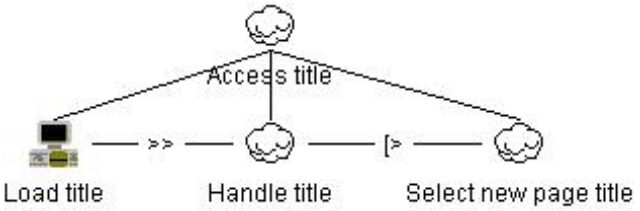


Figure 2: Initial structure of the single page task model.

The second subtask is followed by the disabling operator ([> symbol) indicating that it can be interrupted. In particular, the disabling activity will be the selection of a new page. The third subtask is an abstract task named *"Select new page"* + *T*itle of the page. This last task will contain all the tasks associated to selection of links in the current page leading to other pages, either within or external to the site.

Creation of task structure associated with links

Once we have created the initial structure of the task model of a page, the next rules are dedicated to its refinement according to the actual content of the page. In the analysis we considered the three possible types of links:

- links to anchors internal to the current page;
- links to other pages still internal to the current Web site;
- links to pages external to the current Web site.

The links to internal anchors generate tasks that can still be included as activity related to the current page and thus are inserted as subtasks of *Handle title*, whereas the other two types of links interrupt any activity in the current page and are inserted as subtasks of *Select new page title*.

An example of internal anchors is the selection of a link to the beginning of the same page. Such activity does not interfere with others that can be performed on the same page. For example, considering a page containing a form, it is possible to fill in only some fields, select the internal anchor to the top of the page and then continue filling in other elements in the form.

In order to model such a situation, if there are links to anchors then the node *"Handle"* + *Title* of the page is expanded in the following manner (see Figure 3). The first subtask is an abstract task followed by a suspend/resume operator (|> symbol) indicating that the left-hand activity can be interrupted by the right activity, but when the right activity is terminated then the first activity can be picked up from where it was left off. The second subtask is an abstract task named *"Select Anchors"* + *Title* of the page, which is decomposed in such a way that for each link to an anchor we have a basic interactive subtask called *"Select Anchor"* + *name* of internal anchor. The name corresponds to the name of the tag <A> set for the anchor.

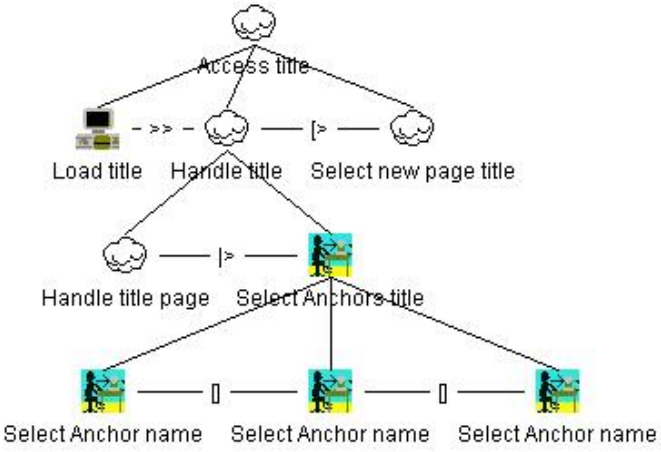


Figure 3: Decomposition for selection of links to internal anchors.

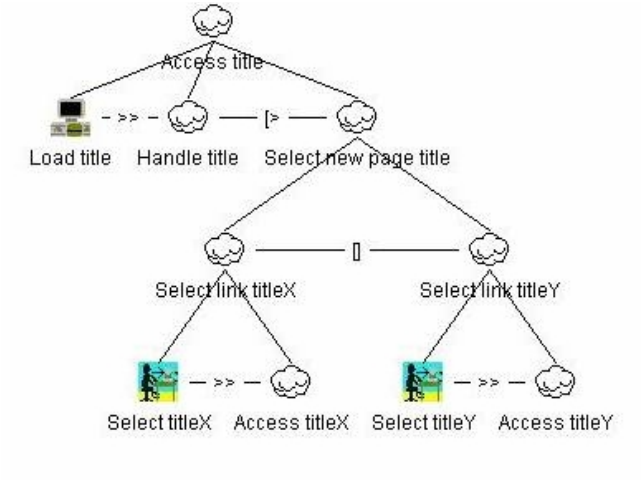


Figure 4: Decomposition for selection of links internal to the current Web site.

Also mailto tags, such as <A href= mailto:fabio.paterno@cnuce.cnr.it>, are associated to internal tasks. Indeed, when the writing of the email message is terminated (through an application external to the web application under consideration) the user will get back to the activities supported by the page under consideration. In this case the name of the task associated with anchor selection is once again defined by *"Select Anchor"* + *name* where the name refers to the mailto string.

Similarly, there is a decomposition describing selection of links internal to the current site. For each internal link we consider two tasks: one representing the actual selection of a link by a user followed by a task representing all the possible activities on the target page (see Figure 4). The first subtask is a basic interactive task named *"Select"* + *Title* followed by the enabling operator. The second subtask is an abstract task called *"Access"* + *Title* of the page. The last one will be expanded when the task model of the entire site is built. These tasks are grouped into an abstract task. The name of the abstract task is given by *"Select link"* + *Title* of the link's target page. All the generated subtrees are included as subtasks of the task named *"Select new page"* + *Title* and composed by the choice operator, which simply indicates that only one link can be selected before changing pages.

When a page contains at least one link to a page external to the current site, in the task model an interactive task, called *"Select" External Link*, is added as subtask of the node grouping all the subtasks defined through the previous rule (*"Select new page"* + *title*). This indicates that at this point the user leaves the site.

Creation of task structure associated with interaction tasks

When navigating in a web application, link selection is not the only basic interactive task. A number of other interaction techniques may be available. In this section we describe the rules indicating how they are considered in the reconstruction process.

Each **button** defined through the tag <INPUT type="button"> or through the tag <BUTTON> is associated with the creation of a selection task in the model. Submit and reset buttons are considered when the Form structure is identified.

For each **Text** or **Text area** we build a subtree, with interactive root, composed of two basic interactive subtasks linked by an enabling operator. The first subtask represents the text field selection while the second one the actual writing (see Figure 5). The name attribute of the HTML element is used to define the task names.

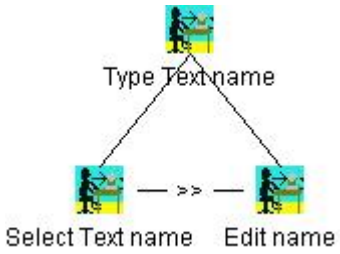


Figure 5: Decomposition for Text area elements.

For each set of **radio** element with the same attribute name, a basic interactive task named *"Select Radio"* + *name* is created. If the name attribute is not defined then for each element an interactive task *"Select Radio"* + *value* is added.

For each set of **checkboxes** with the same name, an iterative interactive basic task called *"Select Checkbox "* + *name* is created (Figure 6). An interactive and optional subtask will be added to this task for each checkbox. Such checkboxes will be composed through the interleaving operator, as the order in which the checkboxes are selected is irrelevant. If the name attribute is not defined, then a interactive task *"Select Checkbox"* + *value* is added for each element.

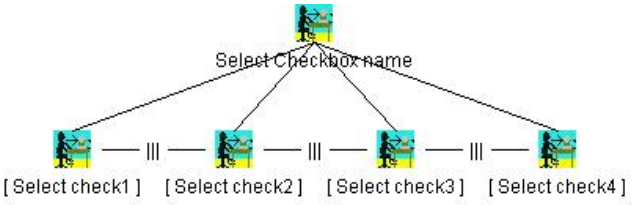


Figure 6: Decomposition for Checkbox

Another user interface element is the **menu** that can be created through the tag SELECT. For each menu, the multiple attribute is examined to determine if it is possible or not a multiple selection.
If the multiple attribute is false then the user interface element corresponds to a set of radio buttons with the same name attribute for a mutually exclusive choice and so it is possible to apply the previous rule.
If the multiple attribute is true then the element corresponds to the set of checkboxes with the same name.

Identification of the objects associated with the tasks

By analysing the HTML elements, the tool is able to identify the objects that each basic task should manipulate during its performance. Objects are classified as user interface or application objects. For example, Table 1 shows the objects and the associated attributes that are identified when a radio button is detected. In this case, one user interface and one application objects are created. While the user interface element is static, the application object can be manipulated. The cardinality attribute of the user interface object depends on the number of choice elements.

Task	Name	Class	Type	Access Mode	Cardinality
Select Radio name	Name	InteractiveRadio	Perceivable	Access	Depends on radio values number (<5, 5-15, >15)
Select Radio name	Radio + name	Single-choice	Application	Modification	Single

Table 1 Objects identified when a radio button is detected

Creation of a task hierarchy associated to a single page

In the model, tasks are structured hierarchically, lower levels indicate how higher levels can be decomposed. This logical structure is quite intuitive and reflects the type of model that often designers have of the activities to support. HTML provides some techniques to logically group related user interface elements, for example, the element **fieldsets**. To make this logical grouping explicit in the task model, all the tasks related to elements included in a tag **FIELDSET** are grouped as subtasks of a new higher-level task. The name of the new task is derived from the tag **LEGEND**.

Figure 7: Example of fieldset.

In HTML **forms** contain the interactive part of a Web page. For each form we have a task pattern (see Figure 8) in which all the tasks related to each of the form's input elements or fieldsets are inserted as subtasks of the *Compile Form name* task. Non-mandatory fields are associated with optional tasks. Input elements are composed by the order independence operator (|=| symbol) to indicate that they have to be performed but the order of performance is not predetermined. If the Form contains both Submit and Reset buttons then the resulting model has the structure indicated in Figure 8. The task "*Send Form*" is an interactive task that represents the selection of the submit button whereas the actual data transmission is represented by the application task "*Submit*". Likewise the task "*Select Reset*" e "*Reset Form*" are created. If the Reset

button is missing then the subtree "*Reset Form name*" will not be included.

The nesting among **Form** and **Fieldset** elements is reflected in the hierarchical structure of the task model. The root of these elements is a subtask of the Handle element created through the first rule explained in this paper. All the tasks associated with input elements not included in a form or fieldset are included as subtasks of the Handle element.

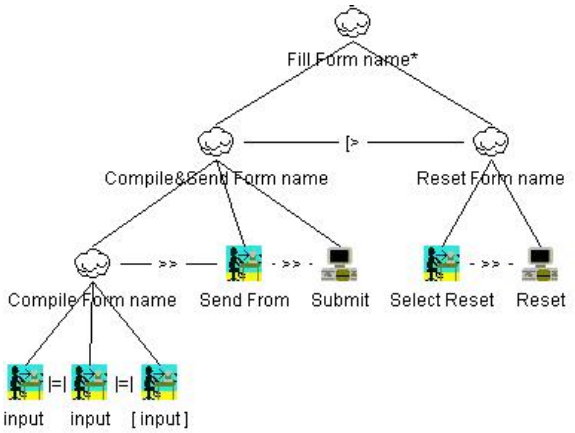


Figure 8: Decomposition for Form elements.

Creation of a new task structure associated with Frames

A HTML page can contain the definition of various **frames**. During the construction of the task model we can consider that interactions with the various frames can occur concurrently (||| operator). For example:

```
<FRAMESET cols="30%,70%">
  <FRAME src="index.html">
  <FRAME src="page.html">
</FRAMESET>
```

The web pages associated with the frames (index.html and page.html in the example) will be analaysed as pages of the site according to the rules described above. Regarding the page containing the structure of the frames, for each frame an abstract node *"Access" + title* of the page referred to the tag Frame is inserted in the *"Handle" + title* node. All these nodes will be composed through the interleaving operator (see Figure 9) and will be expanded when the overall structure of the site will be recomposed.

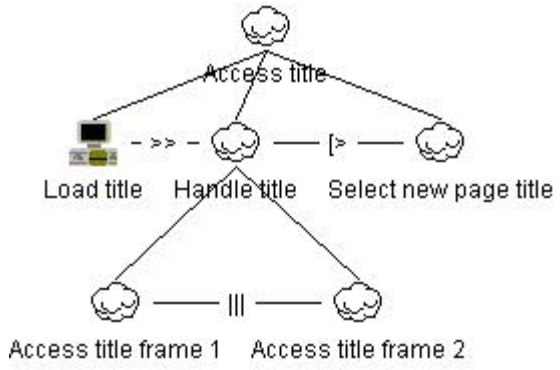


Figure 9: Decomposition for Frames.

CREATION of a TASK STRUCTURE ASSOCIATED with an ENTIRE SITE

Once we have created the task model associated with the single pages, then there is the issue of creating the model's higher levels describing how the various parts are made up. As we saw with internal links, the task model of each page contains an abstract node *"Access" + page* title to which the link refers to indicate the activities possible on the new page.

At this point, starting from the home page the models of the various single pages are connected to create the model of the entire site. In this process it is possible to check whether there are any pages that cannot be reached from the home page.
A brief example can serve to clarify how the various parts of the task model can be composed in order to obtain the entire model. Let us therefore consider two pages that can be accessed sequentially:

http://giove.cnuce.cnr.it/~fabio/index.html
http://giove.cnuce.cnr.it/ConcurTaskTrees.html

The reverse Engineering tool provides the following result.

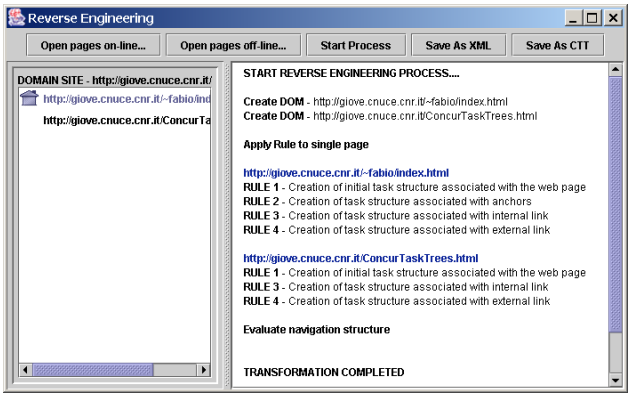


Figure 10: Decomposition for Form elements.

The index page contains anchors (Rule 2) and internal and external links (Rule 3 and 4), whereas the second page has no internal anchor. The next figure shows the task model of the two pages. In the part regarding the index page, the node *Handle Fabio Paternò Page* has not been expanded for the sake of brevity. This node contains some internal anchors and a mailto element.

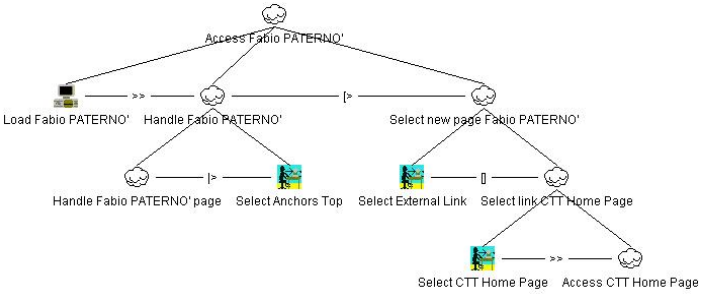


Figure 11: Task model of /~fabio/index.html

Since it is possible to access the second page from the first, the task model of the first includes an abstract node *Access CTT Home Page*, which is the name of the root of the second page task model (bottom-right element in Figure 11).

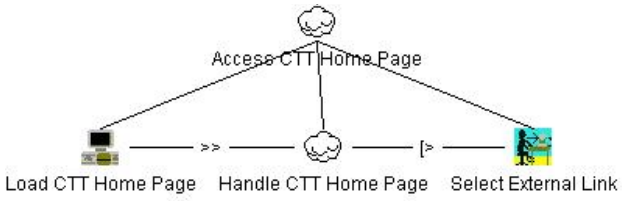


Figure 12: Task model of ConcurTaskTrees.html

The process of constructing the site model starts with the selected home page, which in our example is the page index.html, and analyses the links to pages internal to the site. Then, for each abstract node associated with such internal links we replace them with the associated task model.

In our example, the abstract node *Access CTT Home* in the first model is replaced with the entire model defined for the ConcurTaskTrees.html page (the higher levels are shown in Figure 12) and the final structure is reported below.

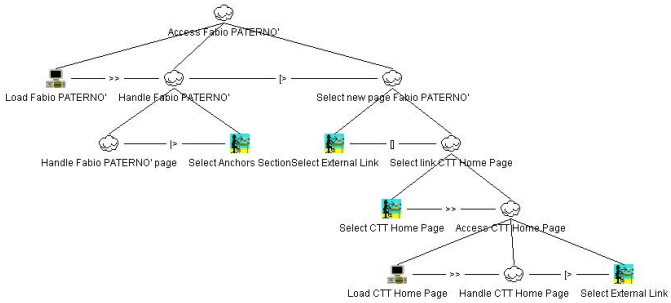


Figure 13: The task model of the two pages.

In the event that the second page contains a link to the home page, then a recursive instantiation of the root of the general model should be included in its task model (bottom-right element in Figure 14). Thus, the resulting structure would be as below.

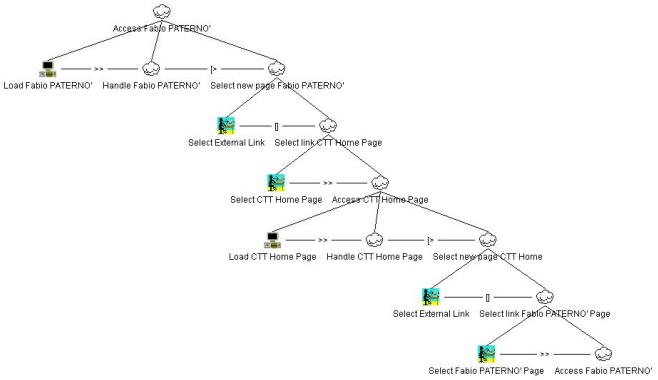


Figure 14: Decomposition for Form elements.

A particularly important case is the presence of structures such as a navigation bar common to all the pages. In this case the subtrees associated with each page are composed by a choice operator ([] symbol) and by a higher level task associated with reaccessing the site. Choosing from among the various pages can be interrupted by the high-level task (see Figure 15) whose recursion indicates that once a branch has been selected, it is then still possible to select any other branch at any time.

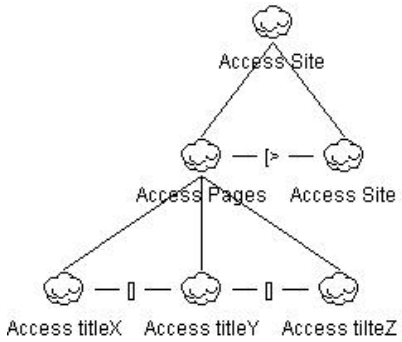


Figure 15: Overall structure for the entire site.

In the process of reconstructing the task model of the web site a number of issues can be detected: links to pages not existing in the web site considered, pages of the site that cannot be reached from the home page, pages with links to themselves.

AN EXAMPLE

In order to illustrate the foregoing rules and their application let us consider the example of the Web site at http://giove.cnuce.cnr.it/ctte.html

All pages of the site display a navigation bar that allows users to reach any part of the site at any time.

The first phase of the reverse engineering process aims to create the task model associated with each page. For example, we can consider the page for the download of the CTTE tool. This page (see Figure 16) contains a form with some input fields, radio, menu, internal anchors, external links and internal links in the navigation bar common to all the pages of the site. Figure 17 shows the sequence of rules applied during creation of the corresponding task model.

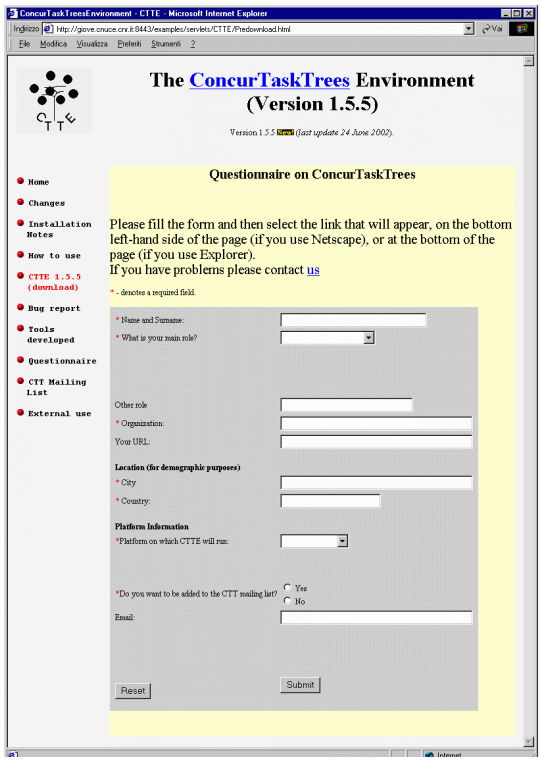


Figure 16:The example considered

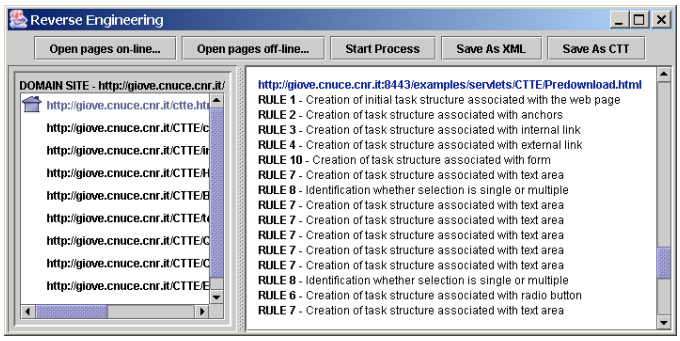


Figure 17: Rules applied to create the task model of the subscription page.

At the beginning the initial structure is created according to Rule 1. Next, all the links in the page are analysed by applying rules 2, 3 and 4. Figure 18 shows the task model created up to this point.

The application of the Rule 2 has determined the creation of one interactive tasks related a mailto element. The task *Select new page CTTE download* contains all the tasks related to external and internal links (in Figure 18 it has not been expanded for reason of space).

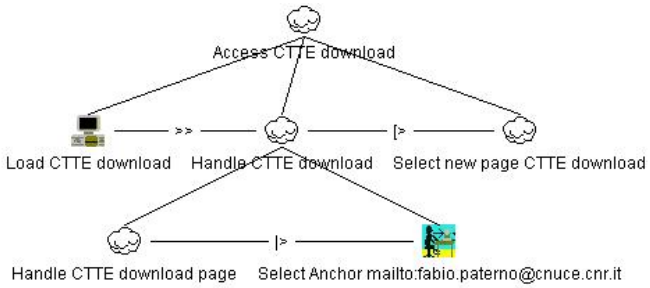
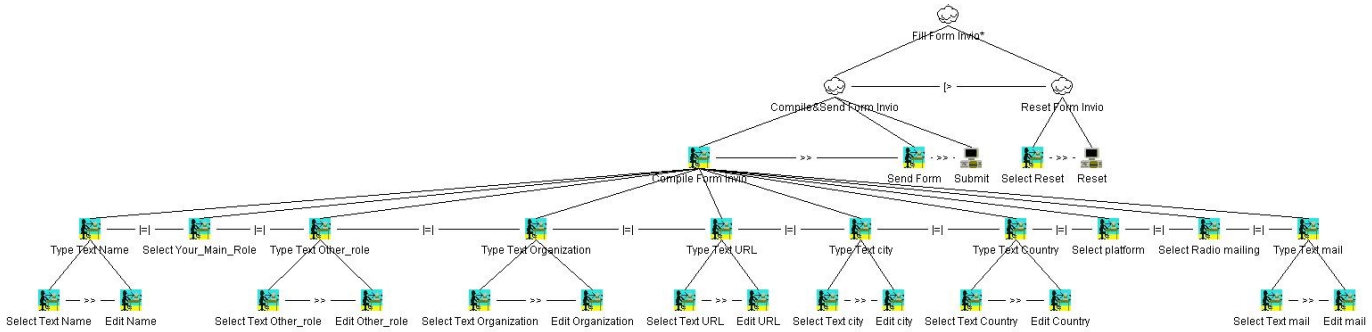


Figure 18: The initial task model for the example

In the next phase a depth-first visit of the DOM of the page is performed and the rules for the elements internal to the page are applied.

Thus, the various subtrees are created as subtasks of the task *Handle CTTE download page*. This visit reveals the presence of a form that requires application of rule 10. While the analysis considers the part of the DOM associated with the form, all the tasks obtained through the application of rule 7, associated with the input elements, are inserted as subtasks of *Compile Form*. At the end of the DOM analysis the task model associated with the form is that shown in Figure 19.

Once the process of creating the task model associated with each page is completed, we move on to consider the overall navigation in the site. Since the site has a navigation bar



that allows the user to access any section from any page, we can apply the related rule.
The resulting model has a recursive structure that describes the possibility of accessing any page at any time. It contains 181 tasks structured into eight levels with 129 basic elements. There are 23 abstract tasks, 142 interaction task, and 16 application tasks.

APPLICATIONS OF THE TASK MODEL

Once a task model of the web site has been obtained, it can be used in different ways:

- as a logical description of the design to analyse and discuss with all the stakeholders,
- as an input element for usability evaluation. We have developed another tool, WebRemUSINE, specifically for such cases.
- as the starting point for redesign of the application for another target platform.

Analysis of the logical description

Once a logical description of the web site considered has been obtained, it can be used for various types of analysis. Such analysis can be better performed with the support of tools, such as CTTE [7], which is freely available at (http://giove.cnuce.cnr.it/ctte.html). In this analysis some metrics can be applied to determine the number of tasks involved, the number of instances of each type of temporal relations and so on. These quantitative values can be used to compare different designs, for example two web sites supporting the same high-level tasks. Another useful automatic aid is a simulator of the dynamic behaviour: it is able to show the list of enabled tasks and allows the designer to select one of them. Then, it calculates what the next enabled tasks are, according to the temporal relations specified in the model. This allows the designer to better understand what tasks are enabled over time according to the current logical structure of the Web site.

Usability evaluation

The resulting task model can also be useful for usability evaluation. For example, WebRemUsine [8] combines empirical testing with model-based evaluation. In addition, it supports remote evaluation because it does not require users and evaluators to be at the same place, at the same time, thus opening up the possibility of analysing a large number of sessions with users spread over many sites.
In order to conduct the usability evaluation, during a session all the user interactions are recorded through a Java script that is automatically included in all the pages making up the Web site. In addition, the list of tasks supported by the application is presented to the users so that they can explicitly indicate what the current target task is. This information is then used to analyse the user interactions, for example to detect errors, which are useless actions for accomplishing the current task.
The automatic analysis can take place after a preparation phase during which the designer indicates the mappings between user interactions and the basic tasks in the model (the leaves of the task hierarchy). This association allows the tool to use the semantic information contained in the task model to analyse the traces of actions of the actual use.
More generally, WebRemUSINE provides three types of information regarding tasks, pages and session simulation.
Information regarding task performance concerns the performance time, which is indicated for both high-level and basic tasks through diagrams that also highlight the minimum, median, mean and maximum values. In addition, page download times are highlighted in order to distinguish between the time spent actually visiting and downloading.

Regarding the user session, the tool is also able to take a user log session and simulate it with respect to the task model. This means that for each action in the log it identifies the corresponding basic task and looks at what happens in the model when that task is performed. If some precondition is violated, it highlights the error. It is also able to point out whether the task is useful to accomplish the target task and indicate what the enabled basic tasks are

according to the temporal relationships indicated in the model. This type of analysis is useful to understand whether there is a mismatch between the actual use and the predicted use represented by the task model.
With this type of analysis it is possible to reveal whether users are able to achieve their goals, if there are parts that require longer than expected for users to understand, what types of errors they perform, frequently performed task sequences (or frequently accessed pages), never performed tasks or never accessed pages.

Redesign for another target platform

The ever increasing availability of new interaction devices is transforming the nature of many applications. Such technology enables nomadic applications, which allow users to access an application from various different places through different devices. This also happens for Web applications. Now, there are many sites specialised for access through PDAs or mobile phones. However, the set of tasks that can reasonably be supported by each type of platforms can be different: a phone allows quick access to small bits of information, whereas a desktop enables users to browse detailed information. Thus, another opportunity provided by our tool is to first create the task model of a web site dedicated to desktop application, then support its analysis in order to identify the changes that should be performed at the logical task level in order to provide access through another type of platform (such as a PDA or a mobile phone). The modified task model can then be used in order to drive the implementation of those parts dedicated to supporting access through the new target platform.

CONCLUSIONS

We have presented a method and the related tool supporting the reconstruction of the task model of existing Web sites. This allows designers to better understand how current implementation assumes that tasks should be performed in order to reach user's goals.
We also discuss how the result of this reverse engineering process can be provided as input to a number of tools for various purposes (model analysis, usability evaluation, user interface redesign).
The tool is freely downloadable at http://giove.cnuce.cnr.it/webrevenge.html

REFERENCES

[1] Bouillon, L., Vanderdonckt, J., and Souchon, N. Recovering Alternative Presentation Models of a Web Page with VAQUITA, Proceedings of CADUI'02, Valenciennes, pp.311-322, Kluwer, 2002.

[2] Brasser M., Vander Linden K., Automatically Eliciting Task Models from Written Task Narratives, Proceedings of CADUI'02, Valenciennes, pp.83-90, Kluwer, 2002.

[3] Card, S., Moran, T., Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, 1983.

[4] Chung-Man Tam R., Maulsby D., Puerta A., "U-TEL: A Tool for Eliciting User Task Models from Domain Expert", Proceedings ACM IUI'98, pp.77-80, ACM Press, 1998.

[5] Document Object Model (DOM) Level 1 Specification (W3C Recommendation) http://www.w3.org/TR/REC-DOM-Level-1/

[6] Hudson, S., John, B., Knudsen, K., Byrne, M., "A Tool for Creating Predictive Performance Models from User Interface Demonstrations", *Proceedings UIST'99*, pp.93-102, ACM Press, 1999.

[7] Mori G., Paternò F., Santoro C., "CTTE: Support for Developing and Analysing Task Models for Interactive System Design", IEEE Transactions on Software Engineering, pp. 797-813, August 2002 (Vol. 28, No. 8).

[8] Paganelli, L., and Paternò, F. Intelligent Analysis of User Interactions with Web Applications, Proceedings ACM IUI'02, pp.111-118, ACM Press, 2002.

[9] L.Paganelli, F. Paternò, Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings Fourteenth International Conference on Software Engineering and Knowledge Engineering, pp.439-445, ACM Press, Ischia, July 2002.

[10] Paternò F., Model-based design and evaluation of interactive applications, Springer Verlag, 1999. ISBN 1-85233-155-0.

[11] Stroulia E., Kapoor R., Reverse Engineering Interaction Plans for Legacy Interface Migration, Proceedings CADUI 2002, Kluwer Academics pp.295-310.

[12] Tidy http://www.w3. /People/Raggett/tidy/