CAMELEON Project

R&D Project IST-2000-30104

Title of Document:	Tools for Model-Based Design of Multi-Context Applications
Author(s):	G.Mori, L.Paganelli, F.Paternò, C.Santoro, G.Calvary, J.Coutaz, D.Thevenin
Affiliation(s):	ISTI - CNR /UJF
Date of Document:	September 5, 2002
CAMELEON Document:	Deliverable 2.1
Distribution:	Reviewers
Keyword List:	Tools, transformations, Models, Multi-platform applications
Version:	Completed

CAMELEON Partners:

ISTI - CNR, Pisa, Italy

Université catholique de Louvain, Belgium

University of Grenoble, France

MTCI, Motorola, Turin, Italy

IS3-GICE, Paris, France

Title: Tools for Model-Based Design of Multi-Context Applications	**Id Number: D 2.1**

Abstract

This document provides a description of the tools prototypes, developed by the partners in the project, supporting model-based design of multi-platform applications. A demo of the tools will be given at the review meeting. Thus, the purpose of this deliverable is to introduce the underlying methods.

Title: Tools for Model-Based Design of Multi-Context Applications	**Id Number: D 2.1**

Table of Contents

Introduction

In a recent paper, discussing the future of user interface tools, Myers, Hudson, and Pausch [MHP00] indicate that the wide platform variability *encourages a return to the study of some techniques for device-independent user interface specification, so that developers can describe the input and output needs of their applications, so that vendors can describe the input and output capabilities of their devices, and so that users can specify their preferences. Then, the system might choose appropriate interaction techniques taking all of these into account*. The basic idea is that instead of having separate applications for each device that exchange only basic data, there is some abstract description and then an environment that is able to suggest a design for a specific device that adapts to its features and possible contexts of use. This is also called user interface plasticity [TC99] . Methods for modelling work context [BH98] can provide useful information for this type of approach.

This problem is a novel challenge for model-based design and development of interactive applications. The potentialities of these approaches have been addressed in a limited manner. In the GUITARE Esprit project a user interface generator was developed: it takes ConcurTaskTrees (CTT) task models [P99] and produces user interfaces for ERP applications according to company guidelines. However, automatic generation is not a general solution because of many, varying factors that have to be taken into account within the design process. Semi-automatic support is more general and flexible: Mobi-D [P97] is an example of a semi-automatic approach but it only supports design of traditional graphical desktop applications.

UIML [APB+99] is an appliance-independent XML user interface language. While this language is ostensibly independent of the specific device and medium used for the presentation, it does not take into account the research work carried out in the last decade on model-based approaches for user interfaces: for example, the language provides no notion of task, it mainly aims to define an abstract structure. The W3C consortium has recently delivered the first version of a new standard (XForms) that presents a description of the architecture, concepts, processing model, and terminology underlying the next generation of Web forms, based on the separation between the purpose and the presentation of a form. If it shows the importance of separating conceptual design from concrete presentation, it also highlights the need for meaningful models to support such approaches.

More generally, the issue of applying model-based techniques to the development of UIs for mobile computers has been addressed at a conceptual and research level ([EVP01], [CCT01]) but there are still many issues that should be solved to identify systematic, general solutions that can be supported by automatic tools. We aim to support design and development of nomadic applications providing general solutions that can be tailored to specific cases, whereas current practise is still to develop *ad hoc* solutions with few concepts that can be reused in different contexts.

In the CAMELEON Reference Framework [D1.1 02] the overall structure of the way in which the Project consortium has addressed the design and development process of multi-target UI's has been described.

In this deliverable we consider two kinds of tools which cover the two main phases described in the CAMELEON framework -reification and abstraction. We report on different tools developed by project partners and introduce the underlying methods.

1. Tools for forward engineering (reification)

1.1 TERESA: A tool for designing and developing multi-platform Web applications

1.2.1 The Proposed Method

Various models have been proposed to highlight important aspects in the design of user interfaces ([JWM+93], [P99], [SFG93], [SLN93], [VB93]). In our method we focus on models that can support development of user interfaces while preserving usability, in particular task models specifying the different activities that are supposed to be performed in an interactive system. Such models should be developed involving users so as to represent how they prefer to perform activities. The basic idea is to capture all the relevant requirements at the task level and then be able to use such information to generate effective user interfaces tailored for each type of platform considered.

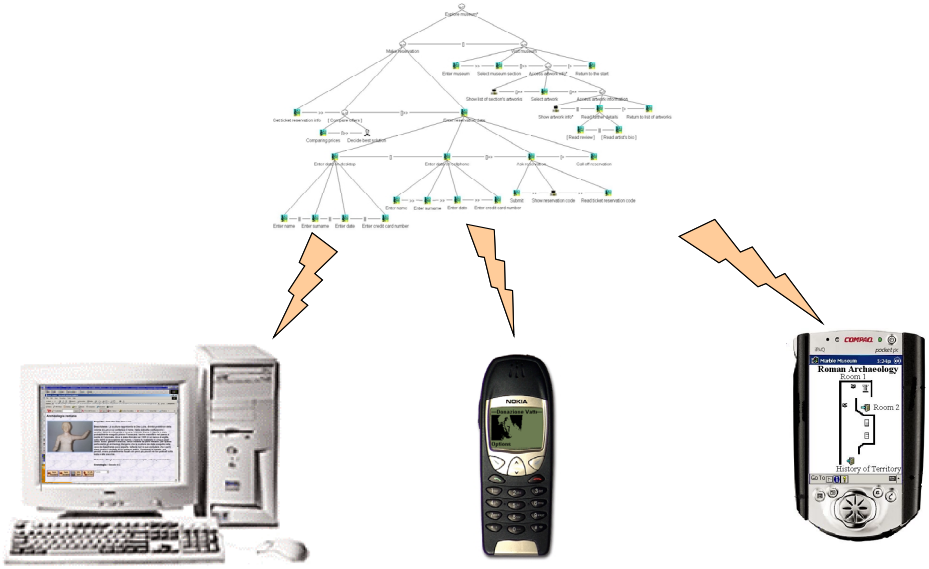


Figure 1: One Model, Many Interfaces

The design of multi-platform applications can follow different approaches. It is possible to support the same type of tasks with different devices. In this case, what has to be changed is the set of interaction and presentation techniques to support information access while taking into account the resources available in the device considered. However, in some cases designers should consider different devices also with regard to the choice of the tasks to support. For example, phones are more likely to be used for quick access to limited information, whereas desktop systems better support browsing through large amounts of information. To

complicate matters, it must be borne in mind that even within the same class of devices there are different presentation models that need to be handled. For example, more and more, cellular phones are being used to access remote applications, and currently access is provided by WAP phones. There are many usability issues that are limiting their spread. While in desktop systems we have mainly two well-known browsers with some compatibility issues (even though such issues often create some problems), in WAP-enabled phones a number of microbrowsers tend to accept slightly different versions of WML, assume to interact with slightly different phones (for examples, phones with a different number of softkeys) and interpret the softkeys interactions differently.

Our method tries to address such problems, and is composed of a number of steps (see Figure 2) that allows designers to start with an overall envisioned task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices:

- *High-level task modelling of a multi-context application*. In this phase designers need to think about the logical activities that have to be supported and the relationships among them. They develop a single model that addresses the various possible contexts of use and the various roles involved and also a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relationships among such objects. Such models are specified using the ConcurTaskTrees (CTT) notation. The CTTE (CTT Environment) tool (publicly available at http://giove.cnuce.cnr.it/ctte.html) supports editing and analysis of task models specified using this notation. The tool allows designers to explicitly indicate the platforms suitable to support performance of each task.
- *Developing the system task model for the different platforms considered*. Here designers have to filter the task model according to the target platform and, if necessary, further refine the task model, depending on the specific device considered. This filter-and-refine process in some cases involves creating task models in which the tasks that cannot be supported in a given platform are removed and the navigational tasks deemed necessary to interact with the considered platform are added. In other cases it forces to add supplementary details on how a task is decomposed when a specific platform is considered. Thus, we obtain the system task model for the platform considered.
- *From system task model to abstract user interface*. Here the goal is to obtain an abstract description of the user interface composed of a set of abstract presentations that are identified through an analysis of the task relationships and structured by means of interactors composed of various operators. Then, still considering the temporal relationships among tasks, we identify the possible transitions among the user interface presentations

considering the temporal relationships that the task model indicates. Analysing task relationships can be useful for structuring the presentation. For example, the hierarchical structure of the task model can be considered to identify interaction techniques to be grouped, for example, those that have the same parent task and are thus logically more related to each other. Likewise, concurrent tasks that exchange information can be better supported by highly integrated interaction techniques (to some extent, merged), as happens when using adjacent techniques, so that users can better follow their mutual dependencies.

- *User interface generation*. In this phase we have the generation of the user interface. This phase is completely platform-dependent and has to consider the specific properties of the target device. For example, if the considered device is a cellular phone, such information is not sufficient, as we also need to know the type of micro-browser supported and the number and the types of soft-keys available.

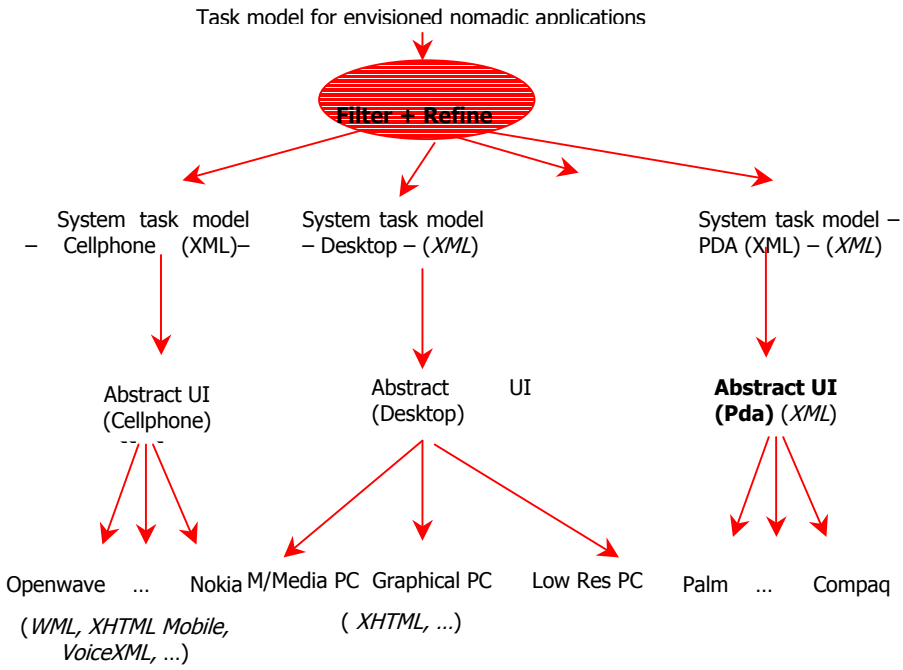


Figure 2 : The transformations supported.

In the following sections we better explain such steps while showing their application to a specific example.

We have defined XML versions of the language for task modelling (ConcurTaskTrees), enabled task sets and the language for modelling abstract interfaces and developed automatic transformations among these representations.

As shown in Figure 3, with regard to the CAMELEON Reference Framework [D1.1 02], the TERESA tool covers the translations from the task model to the abstract user interface and then to the final user interface

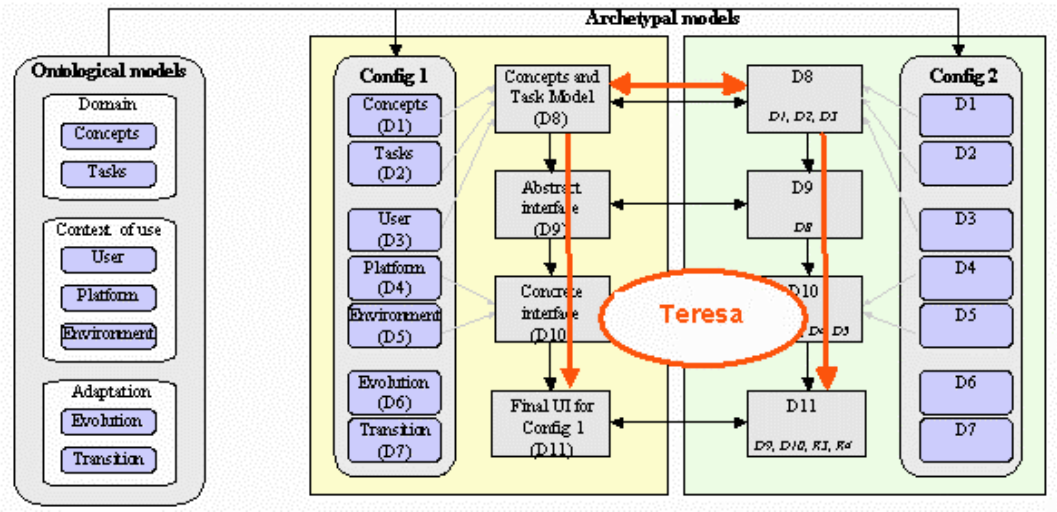


Figure 3: The collocation of TERESA within the CAMELEON Reference Framework

1.2.3. From The Task Model To the Abstract User Interface

Starting with the task model of the system, we aim to identify the specification of the abstract user interface in terms of its static structure (the "presentation" part) and dynamic behaviour (the "dialogue" part): such abstract specification will be used to drive the implementation. By analysing the temporal relationships of a task model, it is possible to identify the sets of tasks that are enabled over the same period of time according to the constraints indicated in the model (enabled task sets). Thus, the interaction techniques supporting the tasks belonging to the same enabled task set are logically candidates to be part of the same presentation, though this criteria should not be interpreted too rigidly in order to avoid excessively modal user interfaces.

This shift from task to abstract interaction objects is performed through three steps:

- *Calculation of enabled task sets;* we have developed an algorithm that takes as input the formal semantics of the temporal operators of the ConcurTaskTrees notation and the specification of a task model and identifies the corresponding enable task sets;
- *Heuristics for optimisation in terms of presentation sets and transitions*; since a direct mapping between enabled task sets and user interface

presentations can generate excessively modal user interfaces or interfaces with a very limited number of elements, these heuristics help designers to group tasks in presentation sets that are better candidates to support the mapping into the user interface presentations.

- *Mapping presentation task sets and their transitions into sets of abstract interaction objects and dialogue.*

1.2.3.1 Identification of Presentation Task Sets

The first step is to calculate the Enabled Task Sets (ETSs) according to the system task model. The CTTE tool automatically performs the identification of these sets. Only application and interaction tasks are considered in ETSs because user tasks (those associated with internal cognitive activities) are not directly relevant to this transformation. The ETSs identify a number of potential presentations and the connections among different ETSs are represented by transition tasks.

Once the ETSs have been defined, we need to specify some rules to reduce their number by merging two or more ETSs into new sets, called Presentation Sets or PSs. The reasons for such step are various: first of all, reducing the initial number of ETSs which —as we previously noted— in some cases can be very high; secondly, keeping and highlighting in the same presentation significant information (as a data exchange is) even when the involved tasks belong to different ETSs so that users can better follow the flow of information; lastly, avoiding repeatedly considering groups of tasks which all share a similar structure. These rules are particularly useful when desktop systems are considered. Up to now, the heuristics that have been identified are the following:

- *H1*: If two (or more) ETSs differ for only one element, and those elements are at the same level connected with an enabling operator, they could be joined together.
- *H2*: If an ETS is composed of just one element, it should be joined with another ETS that shares some semantic feature.
- *H3*: If some ETSs share most elements, they could be unified. For example if the common elements all appear at the right of the disabling operator, they could be joined in only one PS.
- *H4*: If there is an exchange of information between two tasks, they can be put in the same PS in order to highlight such data transfer.

It is worth noting that it is the designer that decides about the heuristics' application, also taking into account the features of the specific platform considered. For example, if we consider graphical user interfaces, it is likely that on devices with small screens the heuristics will be applied less than on other devices with more extended capabilities. The reason is that desktop systems rely on large screen areas, whereas on small displays too many user interface objects in the same presentation would tend to add clutter rather than increase usability.

1.2.3.2 The Language for Abstract User Interfaces

The set of PSs obtained is the initial input for building the abstract user interface specification, which will be composed of interactors [PL94] (abstract interaction objects) associated with the basic tasks. Such interactors are high-level interaction objects that are classified first depending on the type of task supported, then depending on type and cardinality of the associated objects and lastly on presentation aspects.

Figure 4 provides a tree-like representation of the abstract language that has been used for specifying the abstract user interface. As you can see from the picture, an interface is composed of one or more presentations and each presentation is characterised by a structure and 0 or more connections. The basic idea is that the structure describes the static organisation of the user interface, whereas the connections describe the relationships among the various presentations of the user interface. Generally speaking, the set of connections identifies how the user interface evolves over time, namely its dynamic behaviour.

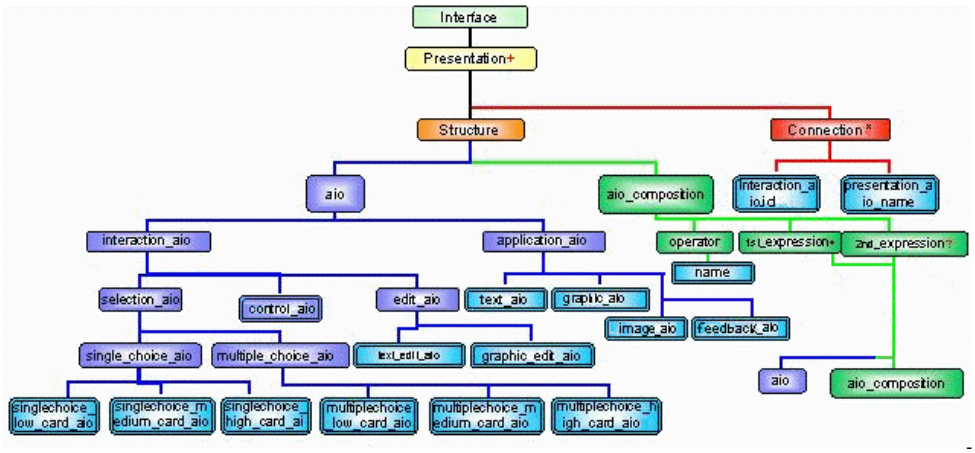


Figure 4: The structure of the abstract user interface language

In fact, each connection has two attributes: an interaction_aio_name, which defines the interaction object whose performance triggers the next presentation, which is identified by the presentation_aio_name.

As far as the structure is concerned, two types of elements can occur: elementary abstract interaction objects (aio), or complex expressions (aio_composition) derived from applying the operators to such objects. Each aio can be either an interaction_aio or an applicaton_aio depending on whether or not an interaction between the user and the application is involved. For interaction_aio we have various categories depending on the type of basic task supported (editing, selection, etc.). For the application_aio we have different classes according to the types of object manipulated.

1.2.3.3 From Presentation Task Sets to Abstract User Interface Presentations

The abstract user interface is mainly defined by a set of interactors and the associated composition operators.

The type of task supported, the type of objects manipulated and their cardinality are useful elements for identifying the interactors. In order to compose such interactors we have identified a number of composition operators that capture typical effects that user interface designers actually aim to achieve [MS95] :

- ***Grouping (G)****:* the idea is to group together two or more elements, so this operator should be applied when the involved tasks share some characteristics. A typical situation is when the tasks have the same parent (they are activities needed to perform a high level task). This is the only operator for which the position of the different operands is unrelevant.
- ***Ordering (O)*** operator: it is applied when some kind of order exists amongst elements. The more intuitive one is the temporal order. The order in which the different elements appear within this operator reflects the order that holds amongst them.
- The ***Relation (R)*** operator should be applied when a relation exists between n elements y_i, i=1,…, n and one element x. Referring to the task model, a typical situation is when we have a leaf task t at the right side of a disabling operator: all the tasks that could be disabled by t (at whatever task tree level) are in relation with t. Again, also this operator is not commutative.
- The ***Hierarchy (H)*** operator means that a hierarchy exists amongst the involved interactors. It is the importance level associated with the operands that identifies the prominence degree that the associated interaction objects should have within the user interface. The importance can be derived from the frequency of access or depend on the application domain. In order to convey this information, various techniques could be used. In graphical user interfaces one example is allotting within the screen a larger area to objects which are hierarchically more ‘important’.

At this point we have to map each task of the presentation set considered into a suitable interactor and build a presentation structure where the relationships among tasks are reflected through the different relationships between such interactors which are expressed by using the composition operators. In order to derive the presentation structure associated to the specific presentation set and deduce the operators that should be applied to them, we have to consider the part of the task model regarding the tasks belonging to a specific presentation set. In this process we have to consider that temporal relationships existing between tasks are inherited also by their subtasks.

1.2.3.4 The Dialogue Part

Once the static arrangement of the abstract user interface is identified, we have to specify its dynamic behaviour. To this aim, an important role is played by the so-called transition tasks. For each presentation set P, we define transition tasks(P) the tasks whose execution makes the abstract user interface pass from the current presentation set P into another presentation set P'. For each presentation set P, a set of rules (transition_task, next_PS) should be provided whose meaning is: when transition_task is executed, the abstract user interface passes from P to next_PS.

1.2.3.5 From the Abstract User Interface to Its Implementation

Once the elements of the abstract user interface have been identified, every interactor has to be mapped into interaction techniques supported by the particular device configuration considered (operating system, toolkit, etc.), and also the abstract operators have to be appropriately implemented by highlighting their logical meaning: a typical example is the set of techniques for conveying grouping relationships in visual interfaces by using presentation patterns like proximity, similarity and continuity [MS95] .

It is worth noting that the composition operators can be iteratively applied on the elements of the domain objects, and we refer to this possibility by putting a * beside the name of the operator (e.g. G*, H*). This means that each interactor involved in the expression actually involves multiple user interface objects, so the operator is iteratively applied to every data element handled by the interactor in the expression.

One example of this can be seen in the following expression, for the desktop system:

R(**H**(**G***(Show_work_info, show_work_image), Section_Description)), **G** (Go_back, Go_museum_map, Go_to_start))

This example shows an expression in which different operators have been used. Reading the expression, we have a grouping operator which involves three elements, namely the interactors associated to *Go_back, Go_museum_map and Go_to_Start* tasks respectively. This grouping is the right member of a Relation operator, which means that each of the elements appearing on the right side is in relation 1:N with the elements appearing on the left side of the R operator.
In turn, the left operand of the R operator is a complex expression which involves a Hierarchy operator. Recalling that H conveys the idea of hierarchy existing amongst the different objects involved, in this case the elements hierarchically arranged are a complex interactor expression (the iterated grouping of *Show_work_info*, *show_work_image*) and a basic interactor (that associated to the *Section Description* task). The latter interactor is considered less important than

the first one and for this reason a smaller space has been allotted to it within the user interface (see Figure 5).

Figure 5: Example of implementation on desktop system

Another example of application can be seen by considering the following expression:

H(Show section list, **G ***(Access to section info, Access to section works))

This expression has, at its outermost level a Hierarchy operator, whose first element ('first' according to the hierarchy as well) is the interactor associated to the Access to sections' list task. The right-hand operand is an iterated grouping of the interactors associates to Access to section's info and to Access to section's works tasks. In figure 6 it is shown how this expression can be rendered on a cellphone user interface, where there are less possibilities due to the more limited capabilities of the device. As you can see the interactor associated to Access to sections' list task has more predominance with respect to the grouped expression and this is conveyed by using a bigger size of the font for displaying the related textual string.

Figure 6: Example of implementation on a Wap device

1.2.4. An Example

The example we considered regards the design of a nomadic application for accessing museum information. Its task model contains tasks that refer to the possible platforms according to the various possibilities discussed in section 4. There are tasks performed through only one platform (such as the possibility of accessing the museum map, namely *Access to map* task,), tasks that are performed differently according to the type of platform (such as *Show artwork info*) and so on.

A possible scenario is a user who comes across the museum while surfing the web at home. S/he accesses the web site and starts to look at the information contained and afterwards s/he proposes a visit to the Marble Museum to some friends. As they appear a bit reluctant, s/he accesses the WAP server trying to convince them by showing some concrete elements of what the museum offers. Finally, they decide to have a visit, so they access again the WAP server to check its location and schedule. When they arrive at the museum, they receive a guide implemented in PDA that provides audio-visual support for their visit.

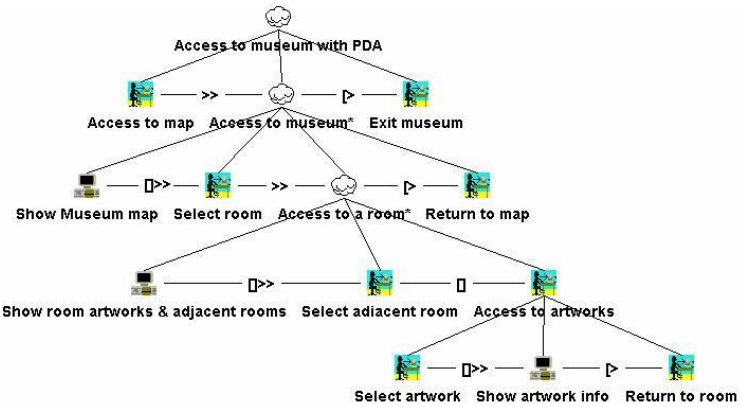


Figure 7: Part of the system task model for the PDA access

For reasons of space we have provided only part of the system task model for the PDA (see Figure 7). Users can access the museum map (Access to map), and then (see the enabling CTT operator ">>"), after such map is displayed (Show museum map), they can select a room (Select room). When they select a room, the presentation shows the different types of artworks in the room, so as to allow visitors to locate them. After the visitor selects a specific artwork (Select artwork), depending on the selection (see the enabling with information passing operator "[]>>") the PDA delivers information about the artwork (Show artwork info), with the possibility of returning to the previous room. Further actions will be made available to the user: either return to the global map (Return to map) or exit the museum (Exit museum), both appearing at the right of a disabling operator ("[>").

For the considered example, the ETSs are:
ETS1: {Access to map}
ETS2: {Show Museum map, Exit museum}
ETS3: {Select room, Exit museum}
ETS4: {Show room artworks&adjacent rooms,Return to map, Exit museum}
ETS5: {Select adjacent room, Select artwork, Return to map, Exit museum}
ETS6: {Show artwork info, Return to room, Return to map, Exit museum}

By applying the heuristics we may find for example that ETS2 and ETS3 differ by only one element: applying H1 they could be unified into PS1={Show museum map, Select room, Exit Museum}. Also, ETS4 and ETS5 share most elements: H3 can be applied to obtain PS2={Select adjacent room, Select artwork, Show Room Artworks & Adjacent Rooms, Return to map, Exit museum}.

For example, the presentation structure then obtained for PS2 is:

O (*Show Room Artworks & Adjacent Rooms*, **G** (*Select adjacent room*, *Select artwork*)) **R** *Return to map* **R** *Exit museum*

This is because the Ordering operator highlights the information transfer (specified by the []>> operator) between the *Show Room Artworks & Adjacent Rooms* and both *Select adjacent room* and *Select artwork* tasks (which are grouped together because of the choice "[]" operator). Each task is in turn put in relation to *Return to map* and *Exit museum* because they both appear at the right of a disabling operator ([>). In the same way, we can identify the abstract presentation associated to PS1 which is: **O**(*Show museum map*, *Select room*) **R** *Exit Museum*

Regarding the transitions, we can see, for example, that PS1 has two transition tasks: Select Room and the task Exit Museum. To express that via the transition task Select room the abstract interface passes from PS1 to PS2 we can use the following rule:

<presentation_set PS1 /presentation_set ><behaviour><rule>
<transition_task Select room /transition_task> <next_PS PS2 /next_PS>
</rule></behavior>

In addition, we have to map each task into a suitable interactor, by considering relevant dimensions specified in the task model. For example with regard to a selection task (as Select adjacent room task) we identify the type of selection (single/multiple), the type of manipulated objects (boolean/quantitative/text, etc) and the cardinality (low/medium/high) of the dataset from which the selection should be performed, as relevant dimensions. Once such attributes have been identified, we define some rules to indicate in which case a widget is more appropriate than another one depending on the different values that each attribute assumes and on the specific platform and/or device considered. For example, as the Select adjacent room task is single selection task managing spatial information and the set of manipulated objects has a low cardinality, the interaction technique to be provided must allow a choice of elements by providing an interactive map.

A possible implementation for the presentation corresponding to PS2 is shown in Figure 8, supposed that the current room is about Roman Archaeology.

Figure 8: One presentation of the PDA user interface

In this part of the document we have described a method for designing multi-device, nomadic applications starting from their task model and using a number of transformations in the design cycle to obtain final applications able to effectively support the users' activities through various devices accessed in different contexts. The application of the method helps maintain a high-level of consistency across the multiple user interfaces developed.

The aim of TERESA (Transformation Environment for inteRactivE Systems representAtions) is to provide a complete semi-automatic environment supporting the presented method and transformations. Particular attention will be paid to design a high-level control panel for designers so that they can focus on main design aspects and choices through effective representations without even knowing the basic underlying mechanisms and concepts (such as enabled task sets) that support the possible transformations.

1.3 ARTstudio: A software environment for the development of multi-platform UI

ARTStudio is a software environment for the development of pre-computed multi-platform UI's. It supports elementary single screen platforms whose resources are known at the design stage. Therefore, distributed user interfaces are not addressed. It complies with the precepts of the Multi-target Reference Framework, but implements a subset of its principles. Before discussing ARTStudio in more detail, we first present a case study: the EDF heating control system.

1.3.1. The EDF Home Heating Control System

The heating control system envisioned by EDF (the French Electricity Company), will be controlled by users situated in diverse contexts of use. These include: At home, through a dedicated wall-mounted device or through a PDA connected to a wireless home-net; In the office, through the Web, using a standard workstation; Anywhere using a WAP-enabled mobile phone. Target users are archetypal adult family members, and the target environments are implicit.

A typical user's task consists of consulting and modifying the temperature of a particular room. Figures 9 and 10 show the final UI's of the system for the target platforms. In 9 a), the system displays the temperature settings for each of the rooms of the house (the bedroom, the bathroom, and the living room). The screen size is comfortable enough to make the entire system state observable. In 9b), the system shows the temperature settings of a single room at a time. A thumbnail allows users to switch between rooms. In contrast with 9a), the system state is not observable, but browsable [GC96] additional navigational tasks, e.g., selecting the appropriate room, must be performed to reach the desired information.

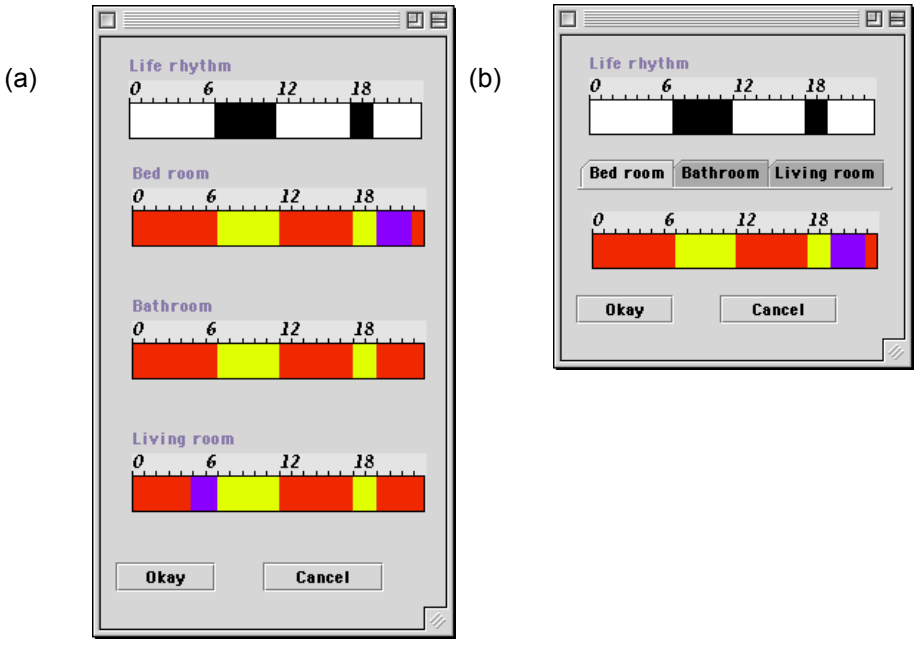


Figure 9: a) Large screen: 24-hour temperature settings are available at a glance for every room of the house. b) Small screen: 24-hour temperature settings are displayed for a single room at a time.

Figure 10 shows the interaction trajectory for setting the temperature of a room with a WAP-enabled mobile phone. In 10a), the user selects the room (e.g., "le salon" – the living room). In 10b), the system shows the current temperature of the

living room. By selecting the editing function ("donner ordre"), users can modify the temperature settings of the selected room (10c).

When comparing with the situation depicted in Figure 9a), two navigation tasks (i.e., selecting the room, then selecting the edit function) must be performed in order to reach the desired state. In addition, a title has been added to every deck (i.e., a WML page) to recall the user with the current location within the interaction space.

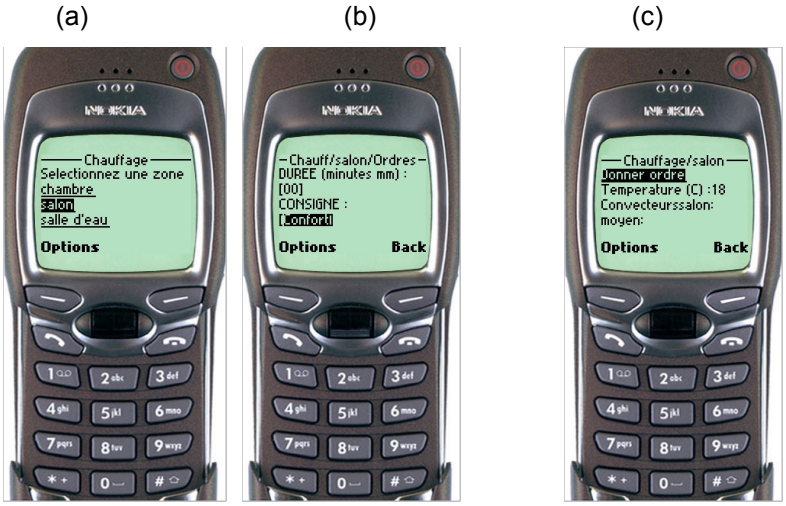


Figure 10: Modifying temperature settings using a WAP-enabled mobile phone.

All of these alternatives have been produced with ARTstudio.

1.3.2. ARTStudio

In its current implementation, ARTStudio supports the four-step reification process of the Multi-target Reference Framework, as well as human intervention and factorization. It supports neither translation nor decorations, and does not include the evolution model. Multi-targeting is limited to Java-enabled single screen elementary platforms. The type of migration is “between sessions” only. ARTStudio is implemented in Java and uses CLIPS, a rule-based language, for generating Concrete UI’s. All of the descriptions used and produced by ARTStudio are saved as XML files.

As shown in Figure 11, the development of a multi-platform UI forms a project. A project includes the Concepts-and-Tasks model, the context model, the Concrete UI and the Final UI. When clicking on the "Contexte" thumbnail, the developer

has access to the platform and interactors models. The "règles" thumbnail gives access to the generation rules used by ARTStudio during the reification process. These rules, which address presentation issues, can be adapted by the designer to the case at hand. In the current implementation, rules are not editable.

Domain concepts are modeled as UML objects using form fill in. In addition to the standard UML specification, a concept description includes the specification by extension of its domain of values. For example, the value of the attribute *name* of type *string* of the *room* concept may be one among *Living room*, *Bed room*, etc. The type and the domain of values of a concept are useful information for identifying the candidate interactors in the Concrete UI. In our case of interest, the *room* concept may be represented as a set of strings (as in Figure 9a), as a thumbnail (as in Figure 9b), or as dedicated icons.

Task modeling is based on ConcurTaskTree ([BPS97] , [P99]). The ARTStudio task editor allows the designer to add, cut and paste tasks by direct manipulation. Additional parameters are specified through form filling. These include:

- Specifying the name and the type of a task. For interaction tasks, the designer has to choose among a predefined set of "universal" interaction tasks such as "selection", "specification", "activation". This set may be extended to fit the work domain (e.g., "Switching to out of frost").
- Specifying a prologue and an epilogue, that is, providing function names whose execution will be launched before and after the execution of the task. At run time, these functions serve as gateways between the Dialogue Controller and the Functional Core Adaptor of the interactive system. For example, for the task "setting the temperature of the bedroom", a prologue function is used to get the current value of the room temperature from the functional core. An epilogue function is specified to notify the functional core of the temperature change.
- Referencing the concepts involved in the task and ranking them according to their level of importance in the task. This ordering is useful for the generation of the abstract and concrete user interfaces: typically, first class objects should be observable whereas second class objects may be browsable if observability cannot be guaranteed due to the lack of physical resources.

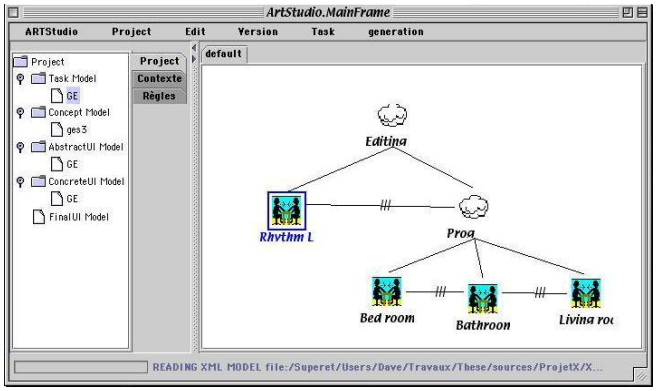


Figure 11: In ARTStudio, a multi-platform UI is developed within a project. The frame of the left end-side gives access to the sets of descriptions. The picture shows the task model for a simplified version of the EDF Heating Control System.

An Abstract UI is modeled as a structured set of workspaces isomorphic to the task model: there is a one-to-one correspondence between a workspace and a task. In addition, a workspace that corresponds to an abstract task includes the workspaces that correspond to the subtasks of the abstract task: it is a compound workspace. Conversely, a leaf workspace is elementary. For example, Figure 12 shows three elementary workspaces (i.e., the Bedroom, the Bathroom and the Living Room) encapsulated in a common compound workspace. This parent workspace results from the Prog task of the task model. In turn, this workspace as well as the Rythm elementary workspace, are parts of the top level workspace.

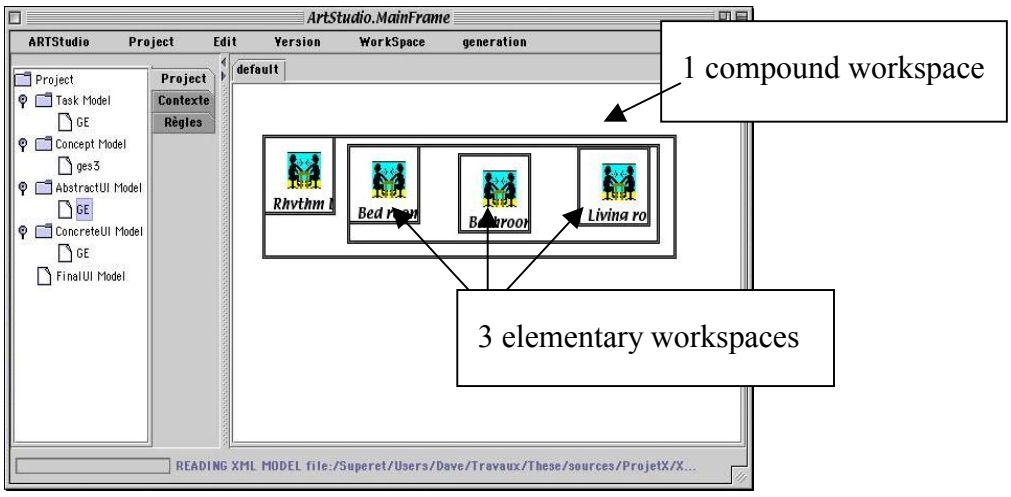


Figure 12: The Abstract User Interface generated by ARTStudio from the Task Model of Figure 11. Thick line rectangles represent compound workspaces whereas thin line rectangles correspond to elementary workspaces.

By direct manipulation, the designer can reconfigure the default arrangements. For example, given the task model shown on the top of figure 13, the designer may decide to group the *Rythm* workspace with the *Room* workspaces (Figure 13a) or, at the other extreme, suppress the intermediate structuring compound workspace (Figure 13b).

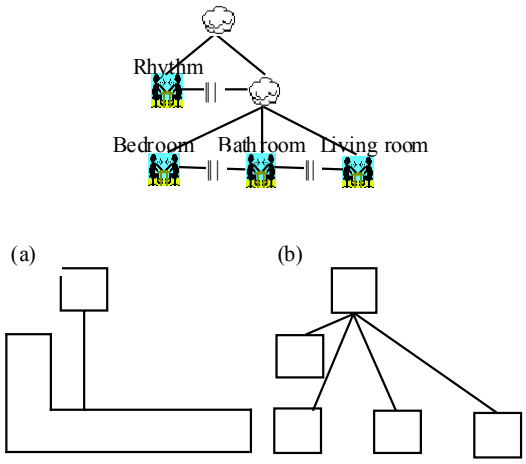


Figure 13: ARTStudio supports human intervention. Workspaces can be reconfigured at will.

ARTStudio completes the workspace structure with a *navigation scheme* based on the logical and temporal relationships between the tasks. The navigation scheme expresses the user's ability to migrate between workspaces at run time.

The platform model is a UML description that captures the following characteristics: The size and the depth of the screen, and the programming language supported by the platform (e.g., Java).

The Interactor Model, for each interactor available on the target platform, includes the following attributes:

- Representational capacity: an interactor can either serve as a mechanism for switching between workspaces (e.g., a button, a thumbnail), or be used to represent domain concepts (e.g., a multi-valued scale as in Figure 9). In the latter case, the interactor model includes the specification of the data type it is able to render.
- Interactional capacity: the tasks that the interactor is able to support (e.g., specification, selection, navigation).

- The usage cost, which measures the system resources as well as the human resources the interactor requires, is expressed as the "x,y" footprint of the interactor on a display screen and its proactivity or reactivity (i.e., whether it avoids users to make mistakes *a priori* or *a posteriori* [C98]).

The generation of the Concrete UI uses the Abstract User Interface, the platform and the interactors models, as well as heuristics. It consists of a set of mapping functions:

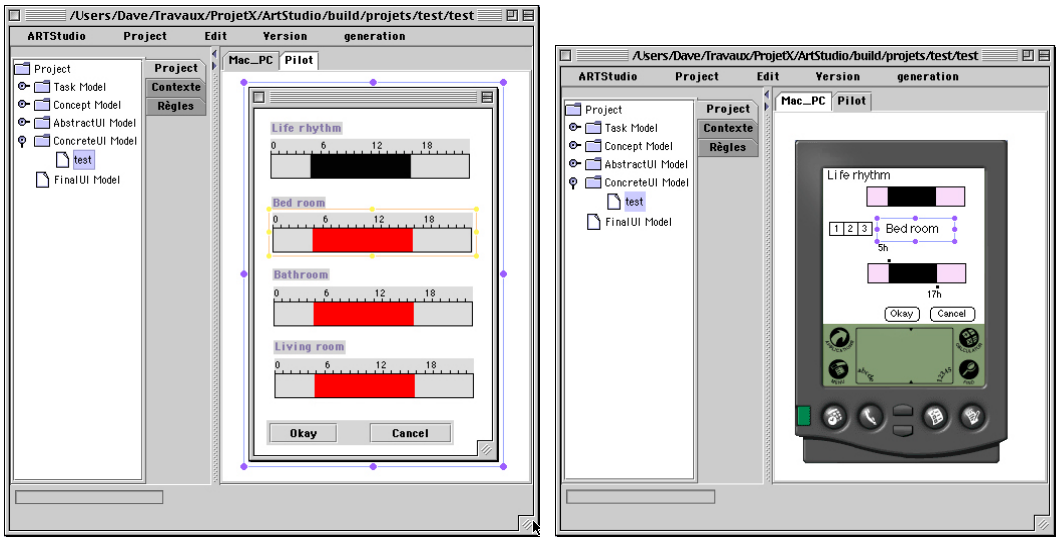
- between workspaces and display surfaces such as windows and canvases,
- between concepts and interactors,
- between the navigation scheme and navigation interactors.

The root workspace of the Abstract User Interface is mapped into a window. Any other workspace is mapped either into a window or into a canvas depending on the navigation interactor used for entering this workspace. Typically, a button navigation interactor opens a new window whereas a thumbnail leads to a new canvas. Mapping concepts to interactors is based on a constraint resolution system. For each of the concepts, ARTStudio matches the type and the domain of values of the concepts with the interactors representational capacity, the interactors interactional capacity and their usage cost.

Figure 14 shows the Concrete UI's that correspond to the Final UI's shown in Figure 9a for large screen targets (e.g., Mac and PC workstations) and in Figure 9b for PDA targets.

Figure14: In a), The Concrete UI generated for the system when running on a workstation. In b), the Concrete UI that corresponds to Palm Pilots.

(a) (b)



As for Abstract UI's, Concrete UI's are editable by the designer. The layout arrangement of the interactors can be modified by direct manipulation. In addition, the designer can override the default navigation scheme.

Pre-computed UI's can now be linked to a run-time environment that supports dynamic multi-targeting.

3. Tools for Reverse engineering (ABSTRACTION)

3.1. Related Work

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 [CMP98] , 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. 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 [HJK+99] , which aims to support the development of KLM models [CMN83] 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 relationships among tasks (apart from sequentiality) and, in any event, does not address the specific aspects of Web applications.

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 this section of the document 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 Web RevEnge 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 usability evaluation through another tool that we have developed and provide some concluding remarks.

3.2. Web RevEnge

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. As shown in Figure 15, with regard to the CAMELEON Reference Framework [D1.1 02], this tool covers the translation from the final user interface to the underlying task model, using HTML code for the final user interface and CTT for specifying task models.

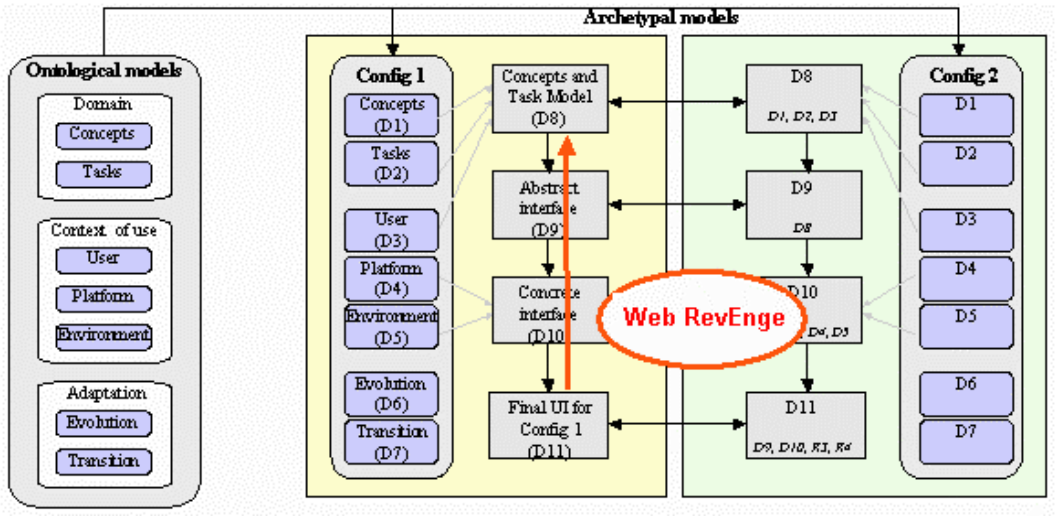


Figure 15 : Collocation of Web RevEnge within the CAMELEON Reference Framework

It is not the purpose of this paper to describe this notation, which is presented elsewhere [P99]. 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 16) 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 [Tidy] it first check that the HTML code is well-formed and if not it corrects it and then create the corresponding DOM [Dom1] 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 subject to modifications or adjustments by the designer using the freely available tool (http://giove.cnuce.cnr.it/ctte.html) for this purpose that can help also to analyse the model with features such as dynamic interactive simulations.

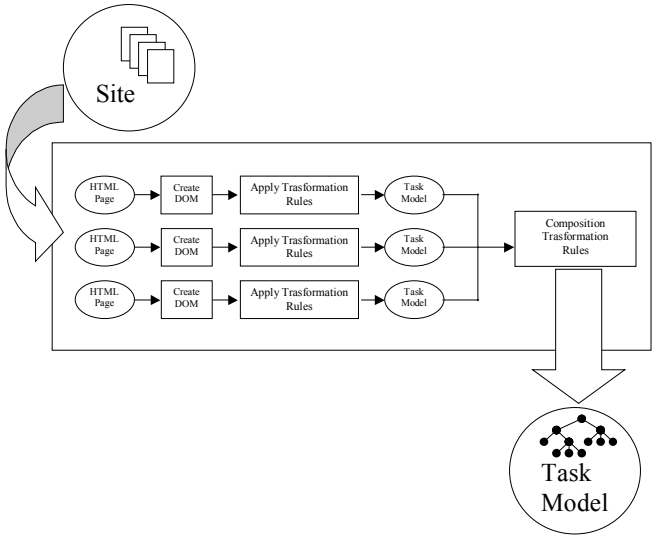


Figure16: The structure of the tool.

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 17. The task name of the root is "Access" + Title of the page (in case 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.

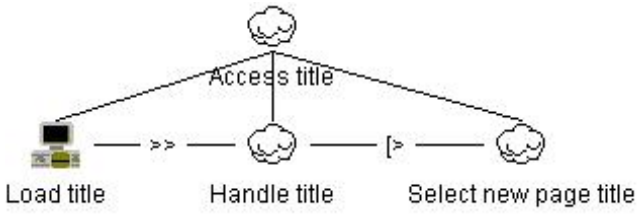


Figure17: Initial structure of the single page task model.

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" + Title of the page.

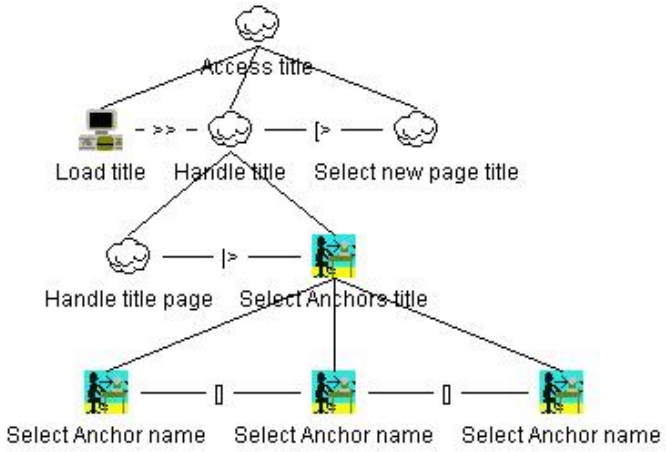
Creation of task structure associated with links

Once we have created the initial structure of the task model, the next rules are dedicated to its refinement. 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 in 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.

If there are links to anchors then the node "Handle" + Title of the page is expanded in the following manner. 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 carried on 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 resulting tree is shown in figure below:



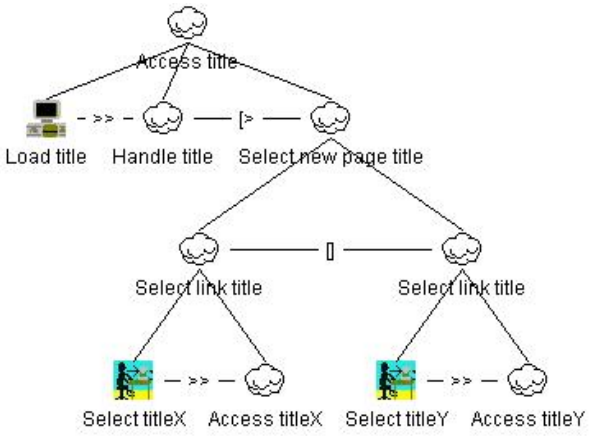
Also mailto tags, such as

<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.

Similarly, there is a decomposition describing selection of internal links. For each internal link there is an abstract task with two subtasks that represent the user link selection and the set of tasks that can be performed on the page associated with the selected link.

The name of the abstract task is given by "Select" + Title . 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 will be built. All the generated subtrees are included as subtasks of the task named "Select new page" + Title and composed by the choice operator (see following figure).



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 . 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. These rules

indicate 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, The name attribute of the HTML element is used to define the task names (see Figure 18).

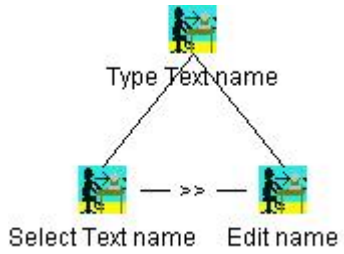


Figure 18: Decomposition for Text area elements.

For each set of **radio** element with the same attribute name, a basic interactive task named "Select Radio" + name is created.

For each set of checkbox with the same name, an iterative interactive basic task called "Select Checkbox " + name is created. To this task, an interactive and optional subtask will be added for each checkbox, such checkboxes will be composed through the interleaving operator. If the name attribute is not defined then for each element a interactive task "Select Checkbox" + value is adedd.

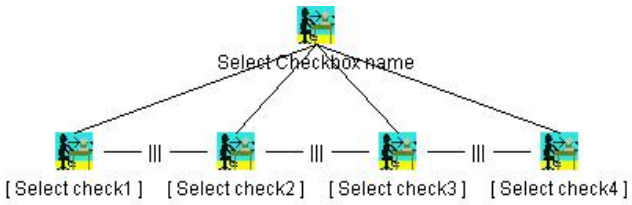


Figure 19: Decomposition for Select elements

Another user interface element are the menus that can be create through the tag SELECT. For each menu the multiple attribute is examined to determine if it is possbile 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 possibile to apply the previous rule.

If the multiple attribute is true then the element corresponds to the set of checkboxes with the same name.

Creation of a task hierarchy associated to a single page

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**.

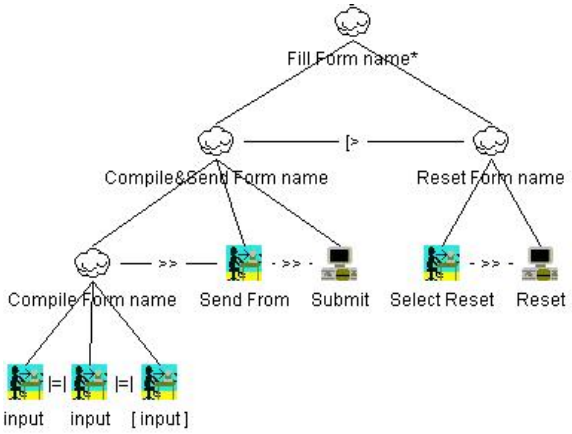


Figure 20: Decomposition for Form elements.

In HTML Forms contain the interactive part of a Web page. For each form we have a task structure 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 20. 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 "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 Rule 1.

All the tasks associated with input elements not included in a form or fieldset are included as subtasks of the Handle element created through Rule 1.

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 nalaysed as pages of the site accordino to the rules described. 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 innserted in the “Handle” + title node. All these nodes will be composed through the interleaving operator and will be expanded when the overall structure of the site will be recomposed.

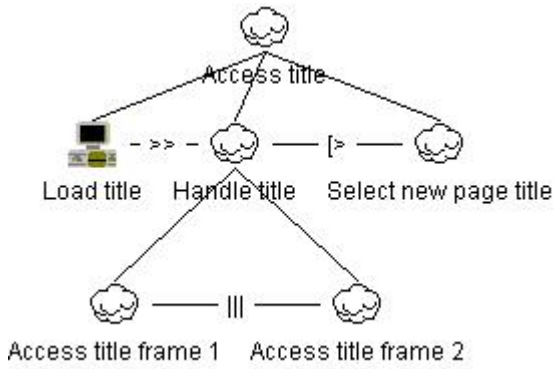


Figure 21: The Decomposition of "Access title" task

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 composed. As we saw with internal links, in the task model of each page there is an abstract node "Access" + title of the page to which the link refer to indicate the possible activities on the new page. At this point, starting from the home page the various model of the single pages are connected to create the model of the entire site. In this process it is possible to determine if there are pages not reachable from the home page.

We can consider a small example to explain how the various parts of the task model can be composed in order to obtain the entire model. As example we can 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.

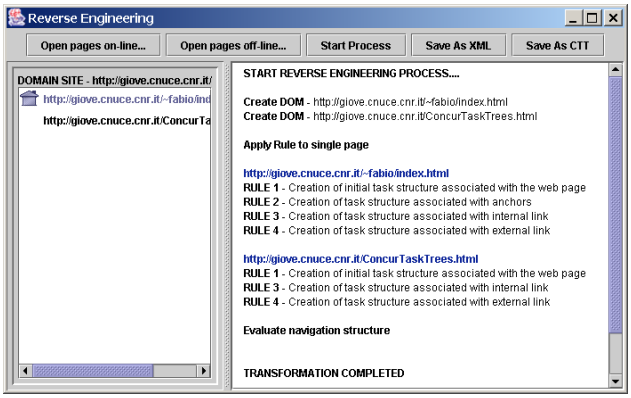


Figure 22: Output of Web RevEnge

In the index page the presence of anchors (Rule 2) and internal and external links (Rule 3 and 4). In the second page, there is no internal anchor. In the next figure the task model of the two pages. In the part related to the index page the node *Select Anchors Section* is not expanded for sake of brevity. This node contains some internal anchors and a mailto element.

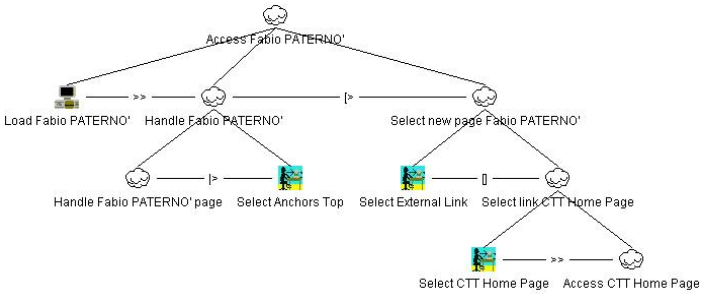


Figure 23: **Task model of /~fabio/index**

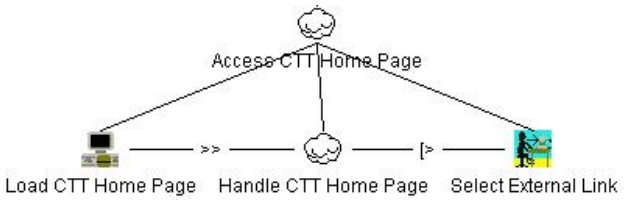
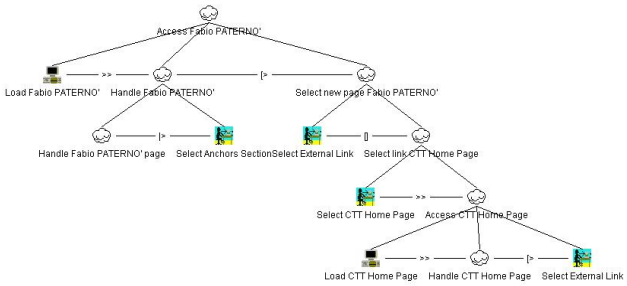


Figure 24: Model of ConcurTaskTrees.html

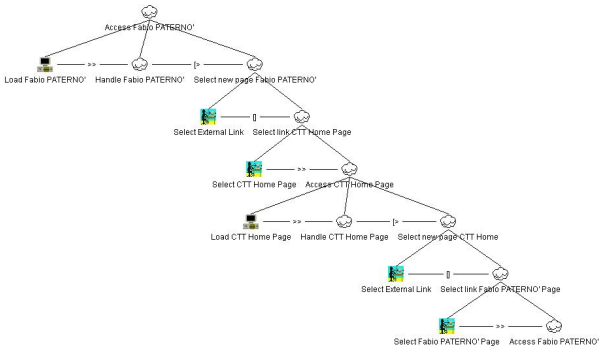
We note that since it is possibile to acces the second page from the first one in the task model of the first one exists an abstract node “Access CTT Home Page” that is the name of the root of the task model of the second page.

The process of construction of the site model starti from the selected home pages, that 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 links internal to the site 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 and the final structure is reported below.



In the case that in the second page there was a link to go to the home page then in its task model a recursive instantiation of the root of the general model should be included. Thus, the resulting structure would be as below.



A case particularly important 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 with a choice operator ([] symbol) and by a higher level task associated with accessing the site. Choosing from among the various pages can be interrupted by the high-level task whose recursion indicates that once a branch has been selected it is then still possible to select any other branch at any time.

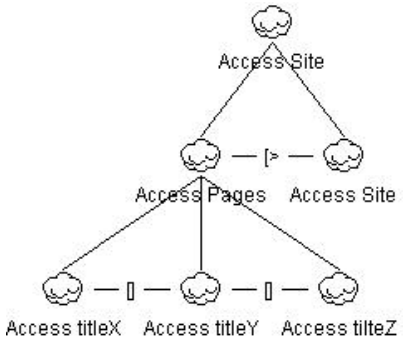


Figure 25: Overall structure for the entire site.

Lastly, the tool is also able to automatically identify the objects manipulated by the tasks from the HTML code.

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.

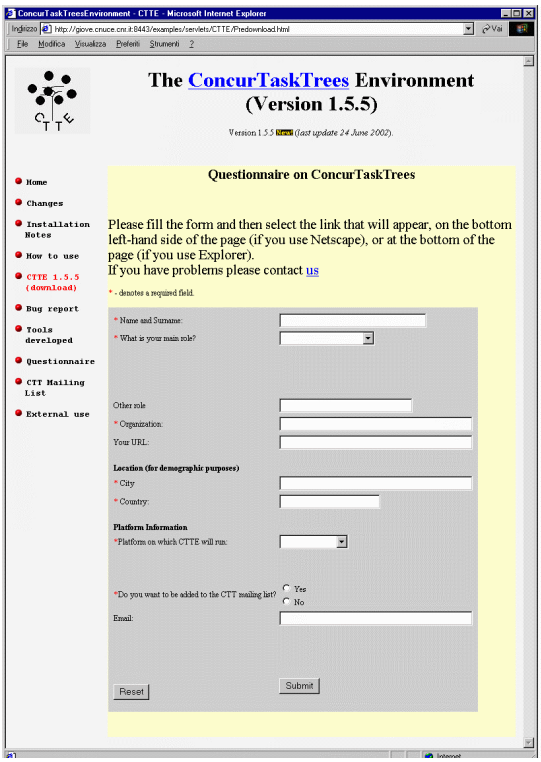


Figura 26:The example considered

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 26) 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 28 shows the sequence of rules applied during creation of the corresponding task model.

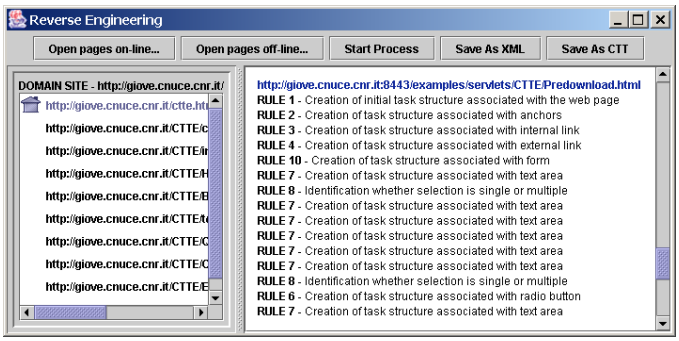


Figure 27: Rules applied to create the task model of the subscription page.

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 28 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 the figure it has not been expanded for reason of space).

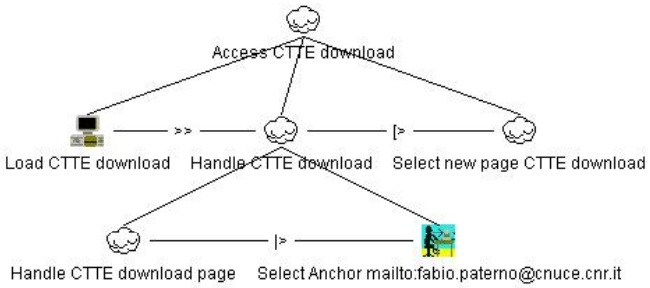


Figure 28: The initial task model for the example

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 sub tasks of *Compile Form*. At the end of the DOM analysis the task model associated with the Form is that shown in Figure 30.

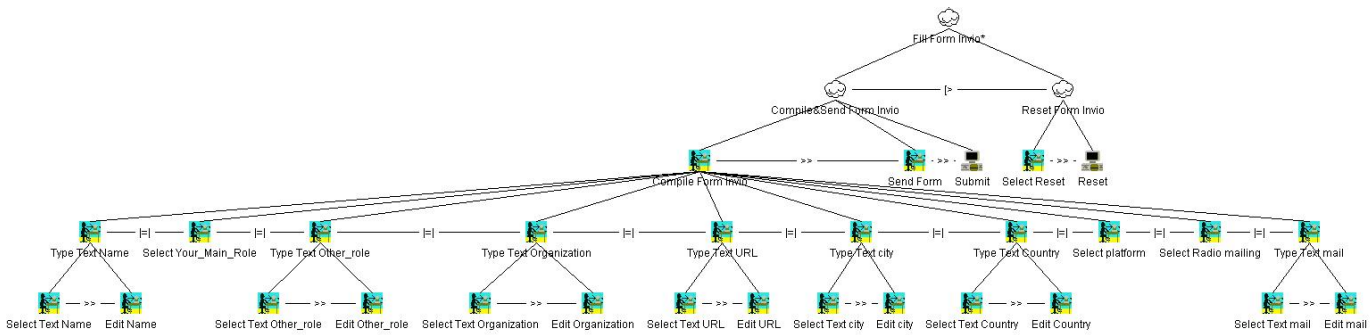


Figure 29: The task model associated with the Form

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.

3.3 VAQUITA

Mobile platforms are becoming an increasingly important alternative for accessing web pages. Many Web pages are not suited to these platforms and need to be adapted, or rewritten from scratch. Adaptation can be made in two ways: either by a dynamic conversion or by a static reengineering. VAQUITA [BVS02] belongs to the second category by applying a model-based approach to reverse engineer web pages at a certain level of abstraction to transfer them to other computing platforms afterwards. Instead of reverse engineering a presentation model and translating it into another model specified for a particular platform, this paper proposes a reverse engineering tailored for any target platform, even one not yet defined. The essence of this reverse engineering approach consists in composing several functions of abstraction, reflection, translation, and reification into two steps: retargeting and regenerating a web page to another platform. Vaquita addresses reverse engineering of Web pages aiming to identifying presentation models for the component pages, which mainly means the abstract description of the interaction techniques used in the implementation. The collocation of VAQUITA within the CAMELEON Reference Framework [D1.1 02], is shown in Figure 31. For further details about VAQUITA see CAMELEON WP2.1 [WP2.1 02]

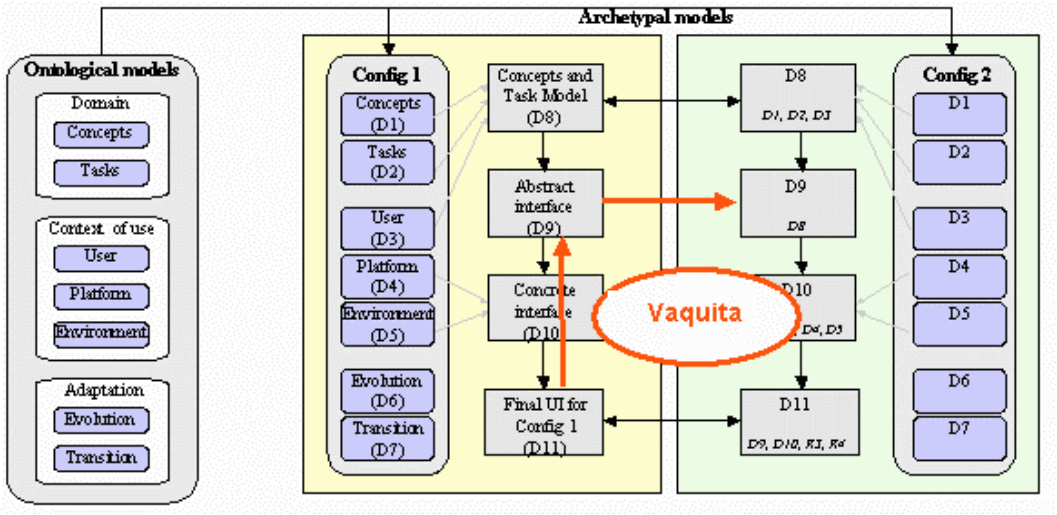


Figure 30: Collocation of VAQUITA within the CAMELEON Reference Framework

4. Comparison between the tools

A set of methods and tools supporting a number of transformations useful when designing multi-platform applications. At ISTI the TERESA tool has been developed currently supporting transformations from task models to desktop and phone user interfaces for Web and WAP access. Another tool supporting automatic reconstruction of task models from HTML code has been developed as well. A different approach to reverse engineering of Web site has been investigated at Univ. of Louvain where the VAQUITA tool has been implemented supporting reconstruction of presentation models from HTML code. Web RevEnge and VAQUITA are already publicly available. TERESA will be publicly available by the end of 2002. Differently from the previous tools, ArtStudio, developed at Univ. of Grenoble, allows development of Java interfaces for multi-platform applications. Discussion on how to integrate such tools through a common XML- based language has started and will be finalized in the second year.

5 References

[APB+99] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.

[BH98] Beyer, H., & Holtzblatt, K. Contextual Design: Defining Customer Centred Systems. San Francisco: Morgan Kaufman, 1998.

[BPS97] Breedvelt-Schouten, I.M., Paterno, F.D., Severijns, C.A.: Reusable structure in task models. In: Proceedings of DSVIS'97, Design, Specification and Verification of Interactive System, Horrison, M.D., Torres, J.C. (Eds) (1997), 225–240

[BVS02] 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.

[C98] Calvary, G.: Proactivité et réactivité: de l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme-Machine, Phd of the University Joseph-Fourier-Grenoble I, Speciality Computer Science, (1998)

[CCT01] Calvary, G., Coutaz, J., Thevenin, D. A Unifying Reference Framework for the Development of Plastic User Interfaces, Proceedings EHCI 2001, Springer Verlag, 2001.

[CMN83] Card, S., Moran, T., Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, 1983.

[CMP98] 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.

[Dom1] Document Object Model (DOM) Level 1 Specification (W3C Recommendation) http://www.w3.org/TR/REC-DOM-Level-1/

[D1.1 02] Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C., *Reference Framework, Models and Criteria for Usability in Multi-Context Applications, Deliverable 1.1, CAMELEON Project,* August 2002

[EVP01] Einsenstein, J., Vanderdonckt, J., Puerta, A. Applying Model-Based Techniques to the Development of UIs for Mobile Computers, Proceedings IUI'01: International Conference on Intelligent User Interfaces, ACM Press, 2001.

[GC96] Gram, C., Cockton, G. Ed.: Design Principles for Interactive Software. Chapman & Hall, (1996)

[HJK+99] 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.

[JWM+93] Johnson, P., Wilson, S., Markopoulos, P., Pycok, J. ADEPT - Advanced Design Environment for Prototyping with Task Models, Proceedings of InterCHI'93, Amsterdam, The Netherlands, 24-29 April 1993, pp 56-57.

[MS95] Mullet, K., Sano, D., Designing Visual Interfaces. Prentice Hall, 1995.

[MHP00] Myers, B., Hudson, S., Pausch, R. Past, Present, Future of User Interface Tools. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.

[O98] Olsen, D. Interacting in Chaos, Keynote address. IUI'98, pp.97-100, San Francisco.

[P97] Puerta, A.R. A Model-Based Interface Development Environment, IEEE Software, July/August 1997, pp 40-47.

[P99] Paternò F., Model-based design and evaluation of interactive applications, Springer Verlag, 1999. ISBN 1-85233-155-0.

[PL94] Paternò, F., Leonardi, A. A Semantics-based Approach to the Design and Implementation of Interaction Objects, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204, 1994.

[PP02] Paganelli, L., and Paternò, F. Intelligent Analysis of User Interactions with Web Applications, Proceedings ACM IUI'02, pp.111-118, ACM Press, 2002.

[SFG93] Sukaviriya, P.N., Foley, J.D., and Griffith, T. 1993. A second generation user interface design environment: The model and the runtime architecture. In Proceedings of INTERCHI '93, Amsterdam. ACM Press, New York, NY, pp.375–382.

[SLN93] Szekely, P., Luo, P., and Neches, R. 1993. Beyond interface builders: Model-based interface tools. In Proceedings of INTERCHI '93. ACM Press, New York, NY, 383–390.

[TC99] Thevenin, D., Coutaz, J.. Plasticity of User Interfaces: Framework and Research Agenda. In Proc. Interact99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., 1999, pp.110-117.

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

[VB93] Vanderdonckt, J., Bodart, F. Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection, InterCHI'93, 24-29April 1993, pp 424-429.

[WP 2.1 02] L. Bouillon, J. Vanderdonckt, Description of the latest version of VAQUITA, *WP 2.1, CAMELEON Project,* September 2002