Chapter 18
A Public Tool Suite for Modelling Interactive Applications

Marco Manca, Fabio Paternò and Carmen Santoro

Abstract Model-based approaches aim to support designers and developers through the use of logical representations able to highlight important aspects. In this chapter, we present a set of tools for task and user interface modelling useful for supporting the design and development of interactive applications. Such tools can be used separately or in an integrated manner within different types of development processes of various types of interactive applications. This tool suite is publicly available and, as such, can be exploited in real-world case studies and university teaching.

18.1 Introduction

Model-based approaches for interactive applications aim to exploit high-level descriptions that allow designers to focus on the main semantic aspects rather than starting immediately to address implementation details. They have been considered also because such features can be exploited in order to obtain better device interoperability through many possible implementation languages. Even the recent HTML5[1] has adopted some model-based concepts by providing some tags that explicitly characterise the semantics of the associated element. However, this language is limited to graphical user interfaces while we will show that model-based languages (such as the ones supported by the tools presented here) can be exploited to support multimodal user interfaces as well.

[1] https://www.w3.org/TR/html5/.

M. Manca (✉) · F. Paternò · C. Santoro
CNR-ISTI, HIIS Laboratory, Pisa, Italy
e-mail: marco.manca@isti.cnr.it

F. Paternò
e-mail: fabio.paterno@isti.cnr.it

C. Santoro
e-mail: carmen.santoro@isti.cnr.it

B. Weyers et al. (eds.), *The Handbook of Formal Methods in Human-Computer Interaction*, Human-Computer Interaction Series,
DOI 10.1007/978-3-319-51838-1_18

Over the years, research in model-based approaches for interactive applications has led to many approaches and related tools. However, while some approaches identified in the academic field remained of interest of just a limited community of scientists and researchers, other approaches have stimulated interest also at an industrial one, resulting in quite a large community. Such interest has also prompted initiatives at standardisation level. One of them has been the W3C Working Group on model-based UI,[2] which produced documents regarding various aspects, primarily task models and abstract user interfaces.

In this chapter, we present a tool suite that covers various types of model-based support to user interface design, particularly focusing on recent evolutions of such tools in order to meet novel requirements. In particular, Sect. 18.2 provides some background information useful to better understand the contribution described in this chapter, while in Sect. 18.3, we provide an overview of the set of tools belonging to the suite. Then, in the next sections, each considered tool will be presented separately. In particular, we highlight some recent contributions concerning how to support the development of task models also through touch-based mobile devices (see Sect. 18.4), how to obtain model-based generation of multimodal user interfaces (see Sect. 18.5) and how to reverse-engineer Web implementations at various abstraction levels (Sects. 18.6 and 18.7). Finally, at the end of the chapter, we summarise and discuss the main current trends identified within the model-based area, also sketching out future directions for research in this field.

18.2 Background

The community working on model-based approaches for human–computer interaction has mainly considered the abstraction levels that are indicated in the CAMELEON Reference Framework (Calvary et al. 2002). It identifies four abstraction levels: Task and Domain, Abstract UI, Concrete UI, Final UI. In this section, for each level, we describe the most relevant concepts, languages and tools of the proposed suite. This will be useful to introduce the tool suite presented in the next sections. In software engineering communities, the model-based approaches have been considered with slightly different concepts because they have a different scope and do not address the specific issues in user interface design and development. For instance, a different conceptual approach is the model-driven architecture (MDA) proposed by OMG (http://www.omg.org/mda/). The OMG's model-driven architecture (MDA) specifies a generic approach for model-driven software engineering and distinguishes four different levels of abstraction: the computation-independent model (CIM), the platform-independent model (PIM), the platform-specific model (PSM) and the implementation (or implementation-specific model—ISM).

[2] http://www.w3.org/2011/mbui/.

According to (Raneburger et al. 2013), the CAMELEON Reference Framework is compliant to the MDA and can be seen as a specialisation in the context of UI development.

The **Task and Domain models** level considers the tasks that need to be performed for achieving users' goals and the corresponding relationships and domain objects. As mentioned before, in our approach, this level is mainly covered by the ConcurTaskTrees (CTT) notation (Paternò 1999).

The **Abstract User Interface** (AUI) level describes a UI through the interaction semantics, without referring to a particular device capability, modality or implementation technology. In addition, at this level, the behaviour and the description of the data types manipulated by the user interface are specified. For this purpose, we use the MARIA language (Paternò et al. 2009), which also includes a data model, defined by using the standard XML Schema Definition constructs. More specifically, the AUI is composed of *presentations* that contain *interactors* and/or *composition operators*. The interactors are the elementary user interface elements, which can be either interactive or output-only, while the composition elements indicate how to put together elements that are logically associated. Examples of abstract interactive elements are *single choice*, *multiple choice*, *text edit*, *numerical edit*, *activator* and *navigator*, while examples of output-only elements are those supporting *descriptions* and *alerts*. *Navigator* elements allow users to move from one presentation to another, while *activators* are those elements that activate some functionalities. The composition operators are *grouping* (a group of elements logically connected to each other), *relation* (when two groups of elements have some type of relation, e.g. a set of input elements and a set of buttons to send their values to the server or clear them) and *repeat* (when a set of elements are grouped together since they share the same similar structure). The MARIAE tool (Paternò et al. 2011) provides automatic support for translating CTT task models into MARIA AUI specifications. This is a type of transformation that cannot be performed through simple mappings because task models and user interfaces involve different concepts and relationships. Since the user interface is structured into presentations, the first step is to identify them from the task model. For this purpose, we developed an algorithm that first identifies the so-called presentation task sets (PTSs): each PTS is a set of tasks enabled in the same period of time and thus it should be associated with a given abstract presentation. This is done by taking into account the formal semantics of the CTT temporal operators. After the identification of the abstract presentations, the interactors and the dialogue models associated with them are generated taking into account the following: (i) temporal relations among tasks (because the user actions should be enabled in such a way to follow the logical flow of the activities to perform); (ii) task hierarchy (because if one task is decomposed into subtasks, it is expected that the interactions associated with the subtasks are logically connected and this should be made perceivable to the user; thus, a corresponding grouping composition operator should be specified in the abstract specification); (iii) the type of task (which provides useful information to identify

the most suitable interaction technique for the type of activity to perform, for instance, if it is a task supporting a selection of one element among several ones, a single choice interactor should be provided).

A **Concrete User Interface** (CUI) provides platform-dependent but implementation language-independent details of a UI. A platform is a set of software and hardware interaction resources that characterises a given set of devices, such as desktop, mobile, vocal and multimodal. Each CUI is a refinement of an AUI, specifying how the abstract concepts can be represented in the current platform taking into account the interaction modality available. MARIA currently supports various platforms: *graphical desktop and mobile*, *vocal*, *gestural*, and *multimodal* (which combines graphical and vocal modalities in various ways). For instance, in the graphical concrete description corresponding to Web implementations, the abstract element *single choice* is refined into either a *radio button*, or a drop-down *list*, or a *list box*, or an *image map*, while the multiple choice is refined into a *check box* or a *list box*, the *navigator* into a *link button* or an *image map*. The other elements are similarly refined to support the semantics of the corresponding abstract elements. In the case of a multimodal concrete language, we have to consider refinements for multiple modalities and indicate how to compose them. In particular, the MARIA concrete language for composing graphical and vocal modalities is based on the two previously defined concrete languages (one for the graphical and one for the vocal modality). It adds the possibility to specify how to compose them through the CARE (Complementarity, Assignment, Redundancy, Equivalence) properties (Coutaz et al. 1995), which can occur between the interaction techniques available in a multimodal user interface. Indeed, as we introduced before, the MARIA abstract language structures a user interface into a set of presentations. Each presentation has composition operators, which contain interactors that can be either interaction or only-output interface basic components. Such interactors can have event handlers associated with them indicating how they react to events. Each of these elements of the language, ranging from presentations to elementary interactors, has different refinements for the graphical and the vocal modality, and in the multimodal concrete language, we indicate how to compose them. Thus, a multimodal presentation has associated both graphical attributes (such as background colour or image or font settings) and vocal attributes (such as speech recogniser or synthesis attributes). For example, a grouping composition in the multimodal concrete language can exploit both visual aspects (using attributes such as position, dimension, border backgrounds) and vocal techniques (i.e. inserting keywords/sounds/pauses or changing synthesis properties). The interactors are enabled to exploit both graphical events (associated with mouse and keyboards) or vocal-specific ones (such as no input or no match input or help request).

The **Final UI** (FUI) corresponds to the UI in terms of some implementation language. From the CUI, different final user interfaces can be derived. A FUI can be represented in any UI programming (e.g. Java UI toolkit) or markup language (e.g. HTML).

18.3 The Proposed Tool Suite

In this section, we briefly introduce the main characteristics of the tools that will be presented in detail in this chapter. The set of tools discussed in this chapter covers the various abstraction levels indicated in the CAMELEON Reference Framework. In particular, the Task and Domain level is covered by the CTT language, which is supported by the desktop ConcurTaskTrees Environment (CTTE) and the ResponsiveCTT tool (see Sect. 18.4). The other abstraction levels can be specified by using the MARIA language and are supported by the MARIAE tool. The tool set supports the possibility of transforming representations at one abstraction level into another, which supports forward engineering transformations, the ones going from the most abstract levels down to the more concrete ones. In addition, it is worth noting that each described tool, apart from supporting such transformations, also enables the user to handle (i.e. create, edit, save) the concerned relevant models. There is another tool that supports transformations going from the most concrete levels to the higher ones (reverse engineering tool). Figure 18.1 provides an overview of the languages and the tools that have been developed. A different colour has been assigned to each of the three tools: light grey to ReverseCTT (see Sect. 18.7), black to MARIAE (see Sect. 18.5) and grey to ReverseMARIA (see Sect. 18.6).

The task model language has been recently adopted as the basis for the task model standardisation within W3C.[3] The language also supports the possibility of specifying task pre- and post-conditions (even combined in complex Boolean expressions), which can be exploited not only within an interactive simulation of the specification but also in the user interface generation process in order to derive meaningful and consistent user interface implementations.

CTT is supported by two main tools: Desktop CTTE and ResponsiveCTT. Desktop CTTE was also integrated with tools for model checking in order to formally reason about task properties (Paternò and Santoro 2000). Both tools allow the user to exploit the cloud to store and share task models remotely, which also facilitates potential sharing and collaboration among designers. ResponsiveCTT can be accessed through touch-based mobile devices such as smartphones and tablets. The tool is responsive as it provides adapted user interfaces to better support task modelling through various types of devices. For this purpose, it also exploits some information visualisation techniques, i.e. representations that make users comfortably analyse/modify task model specifications, which is especially useful when medium-to-large task models are rendered on the limited screen size of mobile devices.

In addition, in this chapter, we also focus on the relationships between task model specifications and models for user interface definition exploited in multi-device contexts. In particular, we focus on how task models can be exploited to derive user interface models at various abstraction levels and for various platforms,

[3] http://www.w3.org/TR/task-models/.

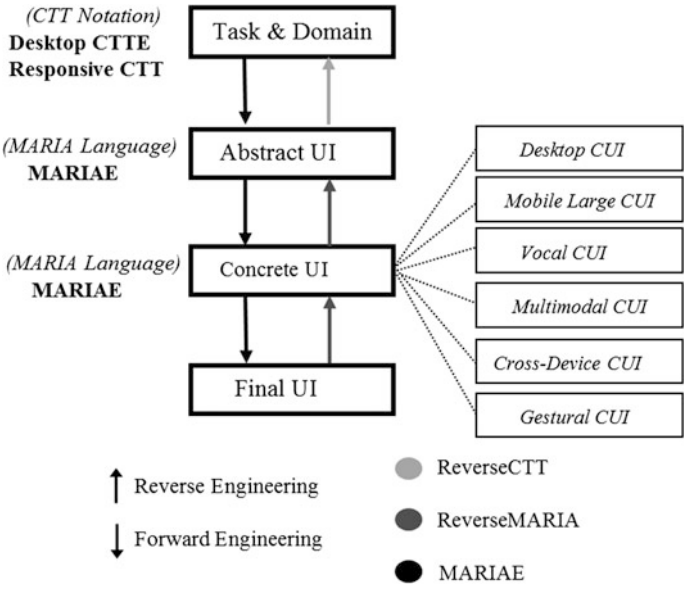

Fig. 18.1 Overview of the languages and the tools proposed

with particular attention to the UI models generated by MARIAE environment. Regarding the latter tool, an aspect addressed is how to provide support for the development of interactive applications able to access and exploit Web services, even from different types of interactive devices. The MARIAE tool is able to aid in the design of new interactive applications that access pre-existing Web services, which may also contain annotations supporting the user interface development. In addition, in MARIAE, various automatic generators are available for a number of platforms (e.g. desktop, mobile multimodal, vocal, distributed), even directly from a CTT task model. Indeed, one of the advantages of using a model-based language such as MARIA over modern languages such as HTML5 is the ability to describe and support even multimodal interfaces by still exploiting a set of core concepts. Another integrated contribution is a reverse engineering tool, which will be discussed to show how to derive interactive systems' descriptions at the various possible abstraction levels starting with Web application implementations.

The development of the tools presented in this chapter has been done according to a number of requirements that were identified and evolved over the years. Regarding the software tools covering Abstract UI, Concrete UI and Final UI, some requirements have been discussed in previous work. For instance, for a predecessor of MARIAE, requirements were identified in Mori et al. (2004): the tool should support mixed initiative, handle multiple logical levels described through XML-based languages and enable different entry points within the multilevel UI specification of CAMELEON conceptual framework. Further requirements for the MARIAE tool were identified in Paternò et al. (2009): the tool should provide

designers with effective control of the user interfaces produced, the transformations for generating the corresponding implementations should not be hard-coded in the tool, the tool should provide support also for creating front ends for applications in which the functionalities are implemented in Web services.

Also for the Task and Domain level and in particular the CTTE Desktop tool, a number of requirements were identified in previous work (Mori et al. 2002): the tool is expected to provide support for modelling and analysis of single-user and cooperative CTT task models, more specifically, visualise/edit/simulate/check the validity of task models, save task models in various formats, support multiple interactive views of task model specifications, support the generation of task models from Web service descriptions (WSDL). ResponsiveCTT, as being more recently developed having in mind mobile devices, posed further requirements which will be discussed in detail in Sect. 18.4.1.

Regarding the tool suite on a more comprehensive level, a main requirement was that it should support all the levels of the CAMELEON framework and associated top-down/bottom-up transformations, which is fully satisfied in our case (see Fig. 18.1).

18.4 Task Modelling

This section introduces the CTT notation for task models. Such notation is supported by two tools: Desktop CTTE is a Java desktop application, and ResponsiveCTT is a responsive Web application, which can be used from various types of devices.

18.4.1 CTT Task Models

Task models indicate the tasks that need to be performed for achieving users' goals by interacting with the UI. Various task model notations are available, e.g. UsiXML (Limbourg et al. 2005), ConcurTaskTrees (Paternò 2000), Hamsters (Martinie et al. 2011), which differ on aspects such as the syntax, the level of formality and/or the operators. A key factor for their adoption is the availability of automatic environments that support model editing, analysis and transformation. However, not all task model notations are supported by (publicly available) tools, and the vast majority of such tools are limited to desktop-based environments. ConcurTaskTrees allows designers to concentrate on the activities that users aim to perform, which are the most relevant aspects when designing interactive applications, and encompasses both user- and system-related aspects. This approach allows designers to avoid dealing with low-level implementation details, which at the design stage would obscure the decisions to make. CTT has a hierarchical structure: this is generally considered as very intuitive since often when people have to solve a problem they tend to decompose it into smaller problems still maintaining the

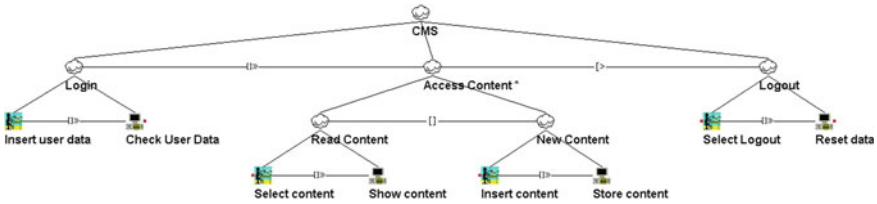

Fig. 18.2 Example of a CTT task model

relationships among the various parts of the solution. Figure 18.2 shows an example of task model related to interacting with a content management system.

The hierarchical structure of this specification has two advantages: it provides a wide range of granularity allowing large and small task structures to be reused, and it enables reusable task structures to be defined at both low and high semantic levels. A rich set of temporal relationships between tasks have also been identified. How the performance of the task is allocated is indicated by the related category and is explicitly represented by using icons. While the category of a task indicates the allocation of its performance, the type of a task allows designers to classify tasks depending on their semantics. Each category has its own types of tasks. In the *interaction* category, examples of task types are as follows: *selection*, when the task allows the user to select some information; *control*, when the task allows the user to trigger a control event that can activate a functionality; *editing*, when the task allows the user to enter a value; *zooming*, the possibility to zoom in/out; *filtering*, the possibility to filter out some irrelevant details. Depending on the type of task to be supported, a suitable interaction or presentation technique will be selected in the development process. Frequency of use is another useful type of information because the interaction techniques associated with more frequent tasks need to be better highlighted to obtain an efficient user interface. The platform attribute (desktop, cellular) allows the designer to indicate the types of devices the task is suitable for. This information is particularly useful in the design of applications that can be accessed through multiple types of platforms, because some tasks could not be available in some platforms. For each task, it is possible to indicate the objects that have to be manipulated to perform it. Since the performance of the same task in different platforms can require the manipulation of different sets of objects, it is possible to indicate for each platform which objects should be considered. It is also possible to exploit such objects to define pre- and post-conditions associated with the tasks. The presence of objects and conditions is indicated in the graphical representation of the model through some cues (see the small rounded icons beside some task icons in Fig. 18.2). Objects can be shared across multiple tasks, and each involved task can have different conditions associated with the object.

The language is also able to model multiuser applications through specific task models for each role involved and an additional model to indicate their relationships. The notation has long been supported by a Java-based desktop tool, the ConcurTaskTrees Environment (CTTE) (Mori et al. 2002), which provides a set of

functionalities for editing, analysis and interactive simulations of the dynamic performance of sequence of tasks. It can be downloaded at http://giove.isti.cnr.it/tools/CTTE/home.

18.4.2 ResponsiveCTT

As mobile devices are now indisputably part of everyday life and widely applied in various domains, we judged it interesting to investigate the possibilities offered by them for task modelling. We focused on truly mobile devices, i.e. those that can be fully and comfortably used even when the user is on the go. As for task modelling tools, some approaches have been put forward, such as CTTE, HAMSTERS and K-MADe (Caffiau et al. 2010), although they all focused on desktop platforms. Attempts to consider modelling tasks on a different platform were carried out mainly in Eggers et al. (2013) and Spano and Fenu (2014). So, apart from a few attempts considering task modelling for mobile use, tools supporting task modelling have been mainly confined to considering desktop platforms. Thus, we have also investigated the possibilities of touch-based modelling on mobile devices through a new tool (ResponsiveCTT[4]). The development of the tool was driven by a number of requirements identified in our experience of several projects and work in which task modelling was exploited. First, in order to widen the impact and possible adoption of the tool, it was developed as a Web application exploiting HTML5, CSS3 (for the presentation) and JavaScript (for the dynamic aspects) accessible by any browser-enabled device. Also, the tool was conceived to support responsive design to effectively adapt the model representations to the screen area of the device, which is particularly important when mobile devices are used. In addition, since the screen size of mobile devices is a key factor for an effective analysis and editing of task models, we judged relevant to exploit information visualisation techniques for dynamically representing task models so as to harness the power of visualization anytime, anywhere, while requiring more limited cognitive effort than in stationary settings. Finally, to store and share task models remotely, the application is cloud-based, which also facilitates collaboration among users.

With the new tool, to edit the task model on a touch-based mobile device, users can touch an empty screen area to create a new task root and perform a tap gesture on a task to edit it, i.e. edit name, type, objects and precondition or add a task as a sibling or as a sub-task, copy/cut and paste a task and/or its children.

In the tool, a focus+context, fisheye-based visualisation technique (Cockburn et al. 2008) has been used as an interactive visual aid to support the visualization, exploration and navigation of task models on touch-based mobile devices, where precise task selections are difficult due to their small screen and limited accuracy of touch commands. In particular, the visualisation of a task model is arranged so that

[4]Available at http://ctt.isti.cnr.it.

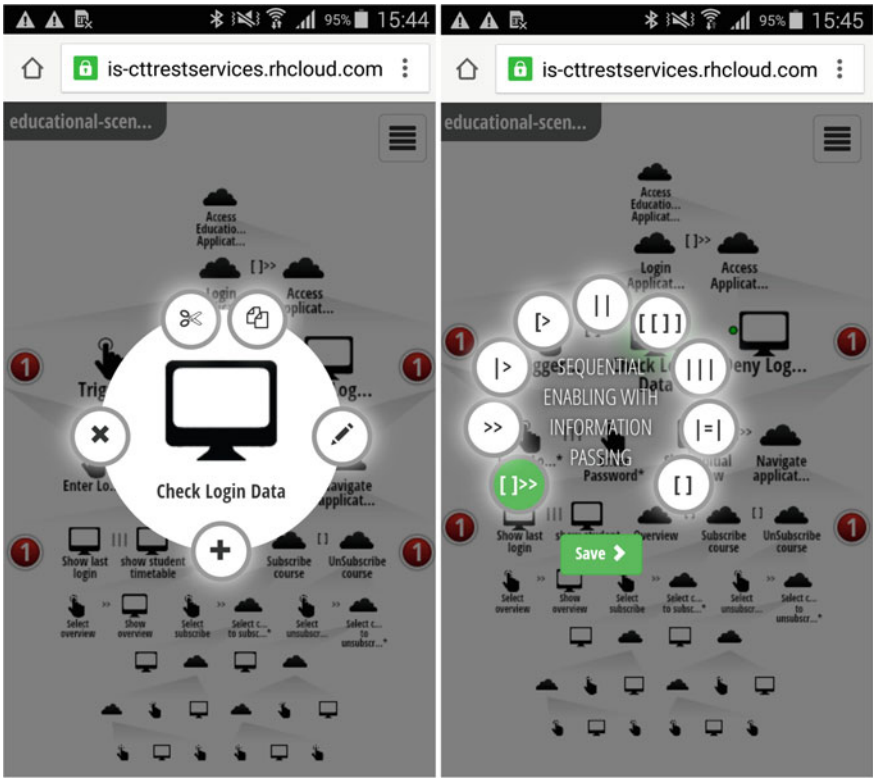

Fig. 18.3 UI for editing task (*left*) and operators (*right*)

the tasks closest to the one that currently has the focus (the task *Check Login Data* in Fig. 18.4) are more emphasised in terms of larger screen space of the associated icons, with a progressive fall-off in magnification towards the upper/bottom model edges. So, in our case, the focus area is determined by the selected task and includes its nearest siblings and children tasks, while the "context" is composed of the remaining tasks of the model. When selecting a task, the application sets the focus to that task and changes the fisheye visualisation of the model accordingly. When users tap on the task currently having the focus, a semi-transparent circular menu appears (see Fig. 18.3), showing the actions that can be executed on that task: change its properties, add new tasks, add objects and pre- and post-conditions associated with it. When users select a new task, the visualisation of the task model is dynamically recalculated to show the new task having the focus on a prominent position of the window and rearrange the model visualisation accordingly. By selecting the icon of a temporal operator, a contextual menu appears, visualising the possible operators (see Fig. 18.3, right part), presented according to their priority. Furthermore, a number of gestures are supported: "pinch to zoom" for zooming on the task model, "swipe down/up" to move up/down in the task model level. It is

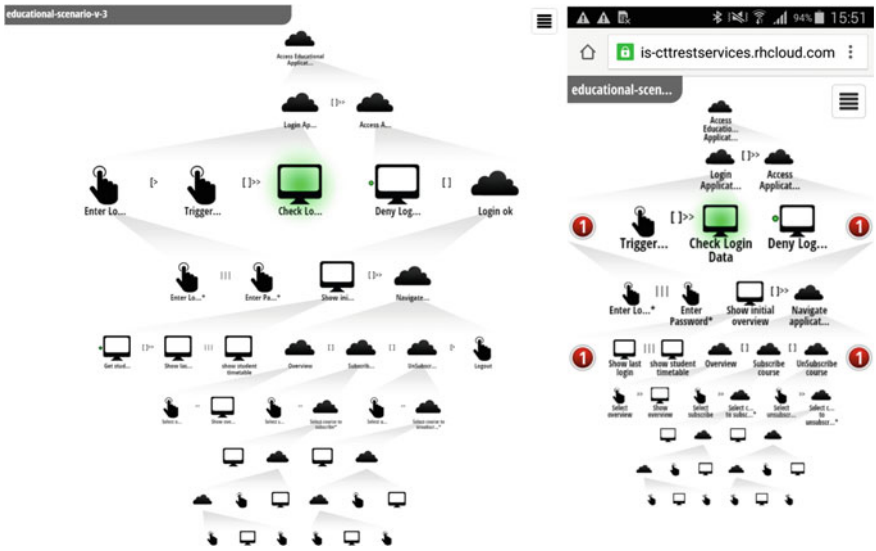

Fig. 18.4 (*left*) Complete task model (desktop); (*right*) Task model on a mobile platform

also possible to change various task attributes (e.g. task name, category) and add the specification of objects manipulated by the task and pre-/post-conditions associated with it. Finally, users can also save models in a dedicated cloud-based service.

As mentioned, the task that has the focus is supposed to be the currently most "important" one; thus, it is always placed in a central position within the working window and highlighted by a specific colour. More generally, every task has a *degree of interest* dynamically calculated, which is inversely proportional to its *distance* from the currently selected task. The dimension of the graphical representation of each task varies according to this distance factor: the further the focused task is, the smaller the icon of the considered task will be, where the "distance" between two tasks is represented by the number of levels that need to be traversed in the model to reach one task from the other. This algorithm is performed whenever a task model is loaded or any editing operation modifies its structure. When it becomes difficult to graphically represent some tasks in a way sufficiently perceivable by the user because of the limited space, they are replaced with a cue-based technique that shows numbers indicating how many tasks are currently hidden (see Fig. 18.4 right part). The numbers are visualised at the same task level and side as the hidden tasks, with the size of the numbered icon proportional to the number itself. By interactively selecting a numbered icon, the previously hidden tasks at the considered level are shown.

Tasks can have some preconditions visualised in the task model through a small coloured rounded icon close to the task icon, whose colour changes during the simulation phase according to the precondition state: if it is true the colour is green,

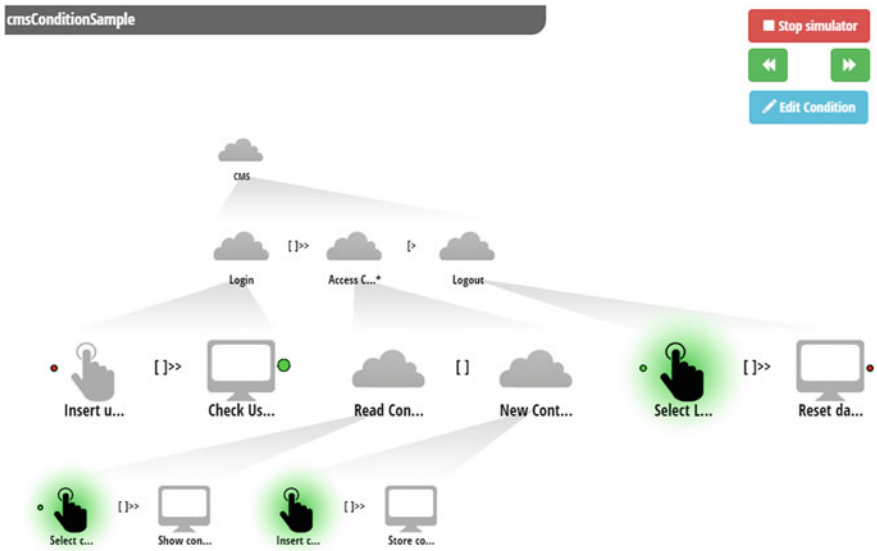

Fig. 18.5 Rendering tasks with preconditions within the ResponsiveCTT simulator

otherwise it is red. Figure 18.5 shows an example while the task model simulator is running.

First-user studies (Anzalone et al. 2015) delivered encouraging results in how the tool supported users in handling task models on mobile devices. They also indicate that tablets are more suitable for supporting task modelling than smartphones since modelling tasks are medium-/long-term, cognitively demanding activities which are better performed when the supporting devices allow for performing them in a comfortable manner.

18.5 Modelling and Generating Multimodal User Interfaces

While the previous section focused on the Task model level, in this section, we describe how to model user interfaces and obtain a corresponding implementation. MARIA has an abstract language and various concrete refinements that depend on the modalities considered. In order to illustrate how to model and generate user interfaces, it can be fruitful to consider the multimodal case, since little work has been dedicated to it and it is an important trend in the HCI area to obtain more natural interfaces. MARIA was developed to address various limitations in previous model-based languages such as TERESA (Berti et al. 2004). One important

contribution of MARIA and the associated environment (MARIAE[5]) is the possibility of generating multimodal interactive applications, which can be executed in any browser-enabled device supporting the Web Speech APIs. This is relevant since most model-based approaches have focused only on graphical user interfaces. A model-based Framework for Adaptive Multimodal Environments (FAME) has been proposed in Duarte and Carrico (2006), but it does not provide support for automatic application development, as in our case. Octavia et al. (2010) describe an approach to design context-aware applications using a model-based design process, but they do not address multimodal adaptation. The model-based approach was also considered in Sottet et al. (2007) for its flexibility, although run-time adaptation is considered only by regenerating the whole UI and multimodal adaptation is not addressed. MyUI (Peissner et al. 2011) provides a framework for run-time adaptations while supporting accessibility. However, its design pattern-based approach can quickly become cumbersome.

Our solution is able to generate multimodal interactive Web applications and does not require using any particular API in addition to standard HTML, CSS and JavaScript. Such implementations are obtained from the concrete language in MARIA addressing multimodal user interfaces. It supports various combinations of graphical and vocal modalities, and it can be easily extended to address other modalities. At various granularities levels, it is possible to indicate the modalities that should be used to support the considered user interface part. There are four possibilities indicated by the CARE properties: *complementarity*, which means that part is associated with one modality and part with another one; *assignment* indicates that only one modality should be considered; *redundancy* is used to indicate that multiple modalities should be used for the same user interface part; *equivalence* is used when there is the possibility to choose one modality or another for the corresponding user interface elements. Depending on the modality indicated, the corresponding concrete attributes are specified.

The multimodal user interface generator produces HTML implementations structured in two parts: one for the graphical user interface and one for the vocal one. The implementation exploits the Web Speech APIs[6] for automatic speech recognition (ASR) and text-to-speech synthesis (TTS). The generator annotates the elements that need vocal support. Such annotations are detected through scripts that are activated when the page is loaded and call the vocal libraries for ASR and TTS. In particular, each UI element is annotated with a specific CSS class in the implementation generation, according to the indications of the CARE properties. If it contains a vocal part and the CARE value is redundancy or vocal assignment, the class *tts* for the *output* elements and the *prompt* part of *interaction element* are added, while the class *asr* is added for the input parts of interaction elements only if the CARE value of this part is *equivalent* or *vocal assignment*. The generated elements are marked with these classes because the multimodal scripts use them to

[5]Available at http://giove.isti.cnr.it/tools/MARIAE/home.

[6]https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html.

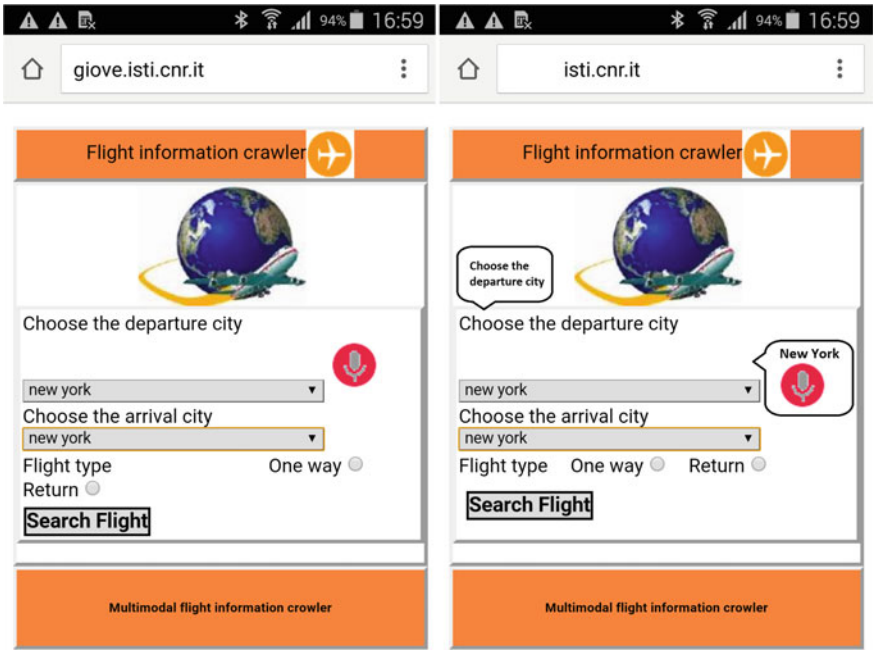

Fig. 18.6 Multimodal generated user interface

identify all the graphical elements having an associated vocal part. Figure 18.6 shows an implementation example for multimodal interaction generated from a concrete user interface (CUI).

In Fig. 18.7, we consider an excerpt of the MARIA multimodal concrete specification related to the definition of the single choice element corresponding to the selection of the departure city in a task aimed to search for a particular flight.

The considered UI element defines the CARE properties for each interaction part (see Fig. 18.7 line 1): for the input element, the CARE value is *equivalent*, which means that input can be entered through either the graphical modality or the vocal one; prompt and feedback phases are both *redundant*; thus, they are visible and vocally synthesised. The single choice element is refined as a drop-down list (line 5) from the graphical point of view and as a single vocal selection (line 10) for the vocal specification. The vocal part defines the vocal prompt, feedback and the events related to the vocal interaction.

Figure 18.8 shows an excerpt of the code generated from the MARIA multimodal specification in Fig. 18.7. Since the single choice element is specified in a multimodal way (see the CARE properties in Fig. 18.7 line 1), the corresponding generated code is composed of a graphical part (Fig. 18.8 from line 1 to 9) and a vocal part (Fig. 18.8 from line 10 to 26) that are not visible but still accessible from the multimodal scripts described before. The label element is annotated with the *tts* class to indicate that there is an associated vocal part with the same id plus the *tts*

```
<single_choice input="equivalence" prompt="redundancy" feedback="redundancy">
    <choice_element vocal_selection="New York" value="New York"/>
    <choice_element vocal_selection="Boston" value="Boston"/>
    <choice_element vocal_selection="Washington" value="Washington"/>
    <drop_down_list>
        <label label_position="left">
            Choose the departure city</string>
        </label>
    </drop_down_list>
    <single_vocal_selection askConfirmation="true" list_values="true">
        <question counter="1" timeout="10">
            Choose the departure city
        </question>
        <feedback>The departure city is</feedback>
        <events>
            <noinput message="No input recived, try again" reprompt="false"/>
            <nomatch message="Your input do not match the possible values"
                reprompt="true"/>
        </events>
    </single_vocal_selection>
</single_choice>
```

Fig. 18.7 Excerpt of the MARIA multimodal concrete specification

```
<!-- Graphical Part -->
<label for="departure_city" id="departure_city_label" class="tts">
    Choose the departure city
</label>
<select id="departure_city" name="departure_city" class="inputElem asr">
    <option value="New York">New York</option>
    <option value="Boston">Boston</option>
    <option value="Washington">Washington</option>
</select>
<!-- Vocal Part -->
<!-- Prompt -->
<span style="display:none" id="departure_city_label_tts">
    <p class="tts_speech" title="{counter:1, timeout:10}">Choose the departure city</p>
    <p class="tts_break">1s</p>
</span>
<span style="display:none" id="departure_city_select">
    <p class="listValues">true</p>
    <p class="askConfirmation">true</p>
    <!-- Feedback -->
    <span  class='vocalFeedback'> The departure city is </span>
    <!--Vocal Events -->
    <span class="vocalEvent">
        <p class="noInput" title="{reprompt:false}">No input recived, try again</p>
        <p class="noMatch" title="{reprompt:true}">Your input do not match the possible values</p>
    </span>
</span>
```

Fig. 18.8 Excerpt of multimodal generated code

suffix (Fig. 18.8 line 2 and line 12). Since the select element is an input element, it is annotated with the *asr* class to indicate that there is an associated vocal part (Fig. 18.8 line 5 and 16).

Figure 18.9 shows another example of multimodal user interfaces for a car rental application. The first page supports a multimodal search through which users can provide the search parameters (also using the vocal modality): for all the interaction

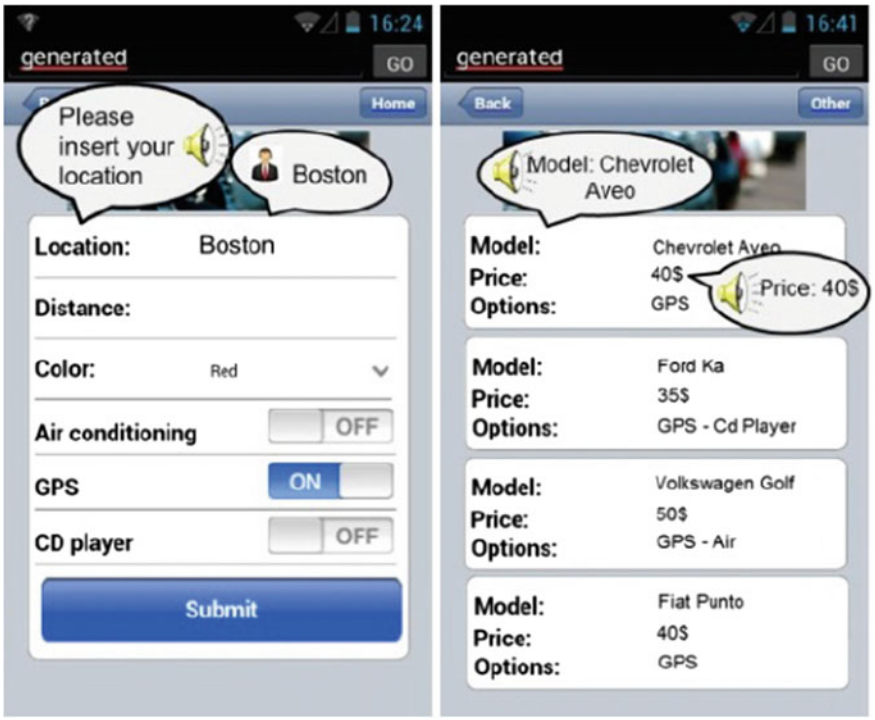

Fig. 18.9 Multimodal car rental application

elements, the prompt and the feedback part are redundant (thus they are rendered both graphically and vocally), while the input part is equivalent (users can choose either modality). The second page is the search result, rendered in a redundant way as well: all the output elements are both graphical and vocal.

18.6 Reverse Engineering User Interface Logical Descriptions

While the previous two sections mainly analysed forward engineering transformations, in this section, we analyse ReverseMARIA, a tool able to reverse any web page (local or remote) to build the corresponding specification in the MARIA graphical Concrete UI language, and then it is also possible to obtain its Abstract UI specification. The tool is available at https://hiis.isti.cnr.it:8443/WebReverse/indexRev.html, and its development was aimed to facilitate building models associated with existing applications. Such models can then be used to obtain the specification of user interfaces more suitable for different contexts of use, for example for devices with different features (Paternò et al. 2008).

In general, there are two main approaches to reverse engineering user interfaces: pixel-based and code-based. Pixel-based approaches analyse the correlations between the display's pixels to find the set of interface components (buttons, forms) and the hierarchy amongst them. Although being implementation language—neutral—see Dixon et al. (2014) as an example—the main problem of pixel-based approaches is that they are able to retrieve only information about the visual presentation of UIs, and not about the behaviour of components, because they depend on the hidden source code implementation. Source-code reverse engineering in turn can be done in two ways: using static analysis through the application code, or analysing the running application in a dynamic way (dynamic analysis). It is also possible to perform hybrid analysis when static and dynamic analyses are used together. There are various tools that exploit the static analysis-based approach (see e.g. Bernardi et al. 2009a; Da Silva 2010). The benefit of static analysis is that all the information is stored in the code and ready to be processed, but it is not able to retrieve the part of the implementation that is not detectable unless the code is executed. Dynamic analysis solves this problem by analysing the behaviour of the user interface during execution. Examples of dynamic analysis are described in Maras et al. (2011) and (Morgado et al. 2010). For Web applications, a risk related to dynamic approaches is leaving many interface states unexplored and thus obtaining an incomplete interface model. For this reason, a hybrid analysis approach (see Silva and Campos 2013; Li and Wohlstadter 2008) can be more thorough with respect to either static or dynamic techniques.

We followed a novel hybrid approach for reverse engineering web pages to enable analysing them both when stored on a server and when loaded client side, also considering the current internal state of the UI and the user inputs. Our tool reverse-engineers web pages by taking as input the DOM representation which can be obtained in two ways: (i) getting the input page through an http request to the server, and generating the DOM from the HTML code using an external library[7]; (ii) serialising the HTML DOM on the client side through either the browser or a proxy. In addition to the extraction of DOM, the tool

- Associates the significant CSS properties with the related HTML elements;
- For each event associated with an HTML element, determines the type of event, the JavaScript functions invoked on its activation and their parameters;
- Transforms the result obtained into an equivalent MARIA element;
- Serialises the MARIA elements in an XML file using JAXB[8] technology.

[7]We used the Java Tidy Parser, available at http://tidy.sourceforge.net/, modified by us to handle HTML5 tags as well.

[8]Java Architecture for XML Binding.

18.6.1 The Reverse Algorithm

The tool has a first pre-processing phase in which it creates the DOM (see Fig. 18.10), moves all the JavaScript nodes to an external file and stores in cache memory all the CSS properties that are in external files or in style nodes. At the beginning, it creates the MARIA *interface* element, which contains information regarding the MARIA version and schema, and a *presentation* element, which contains the default page properties taken from the HEAD part of the page (e.g. page title). Then, it performs a depth-first search visit of the DOM starting from the *body* node. For each node, it first analyses the type and its attributes, then adds the id attribute if it is not set and introduces an attribute used to analyse the nodes' textual properties. Then, it analyses the neighbouring node to retrieve correlations between nodes and classifies single nodes according to their content and visual properties. The result obtained is the logical representation of the DOM structure. The following example (involving the Yahoo home page) shows how the numbered parts of the HTML structure and CSS properties of the page shown in Fig. 18.11 are transformed into the MARIA descriptions shown in Figs. 18.12 and 18.13.

Figure 18.12 shows the MARIA elements (a grouping and a description element) that correspond to the HTML elements indicated by (1) in Fig. 18.11, while MARIA attributes correspond to CSS properties.

In Fig. 18.13, the red part (2) represents the input text element in the considered web page (see Fig. 18.11), while the green part (3) represents the submit button.

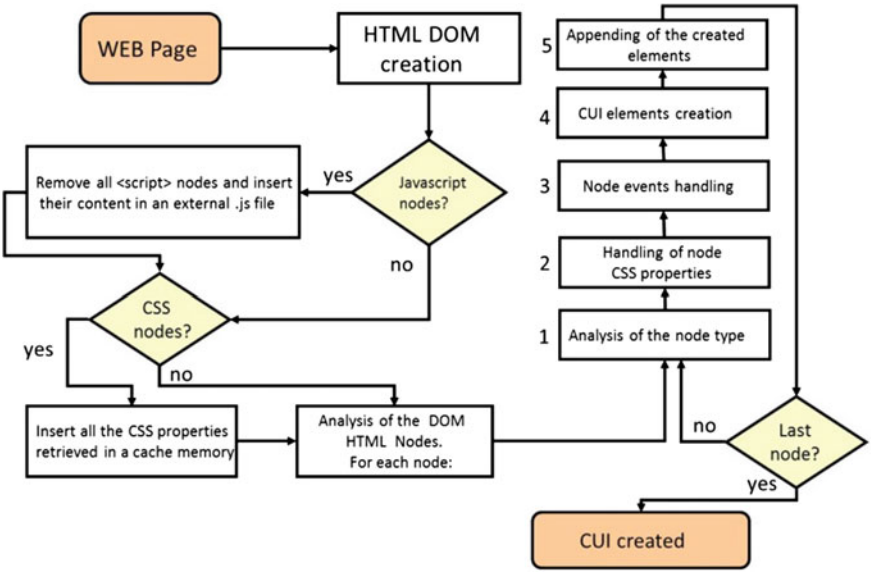

Fig. 18.10 Reverse engineering algorithm

Fig. 18.11 Yahoo Home page

Fig. 18.12 Result corresponding to element 1

Fig. 18.13 Results corresponding to elements 2 and 3

Parts 2 and 3 are contained in a relation MARIA element, which represents an HTML form.

Part 4 in Fig. 18.11 corresponds to a link, with its label composed of a picture and a text. This HTML part corresponds to a MARIA navigator element, along with a connection indicating the target presentation.

18.7 Reverse Engineering Task Models

In this section, we present the ReverseCTT tool, which covers the Abstract UI-to-Task Model transformation of the CAMELEON Reference Framework. While ReverseMARIA is a tool able to reverse any web page (local or remote) and build the corresponding specification in the graphical desktop concrete language of MARIA and then obtain its abstract description, ReverseCTT reverses the MARIA AUI specification into a CTT task model description. Thus, it has a more modular approach than WebRevenge (Paganelli and Paternò 2003b), which aimed to create task models directly from the website code.

The ReverseCTT process is articulated into a number of rules. In order to understand them, it is important to briefly remind the characteristics of the input that this transformation receives, namely the AUI. Every AUI is structured into a set of *presentations*, and each presentation has a *name* attribute. The relationships between the AUI presentations are modelled through *connections,* which can be elementary or complex, and which mainly support navigation between the different presentations belonging to each AUI. In other terms, connections are the logical/abstract counterpart of navigational links in Web UIs. The structure of each presentation is articulated into a set of abstract *interactors* (which can have different *types* ranging from editing, navigation, choice, to description and activators), whose relationships are modelled through *operators* (e.g. grouping, relation) defined in the language. Our assumption is that, after a user selects a presentation, the same pattern repeats, regardless of the specific presentation selected. This pattern is the following (for each presentation): load the (selected) presentation, handle the presentation (for user's goals) and then select a new presentation.

Having said that, the idea is that for each AUI presentation, a new task model is created, whose root name is derived by the name of the AUI presentation (by concatenating "Access" + <AUI presentation name>). The root task just created has three children: one is an application task ("Load" + <AUI presentation name>. This application task is followed (through an enabling operator) by an abstract task ("Handle" + <AUI presentation name>), which in turn is disabled by a second abstract task ("Select new page" + <AUI presentation name>). The resulting CTT task model after this step is shown in Fig. 18.14.

In the second step, the process analyses the elements of type "*elementary connection*" in the AUI, which mainly represent HTML anchors/links. For instance,

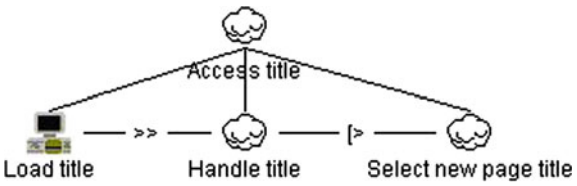

Fig. 18.14 From AUI to CTT: the model after the first step of the transformation

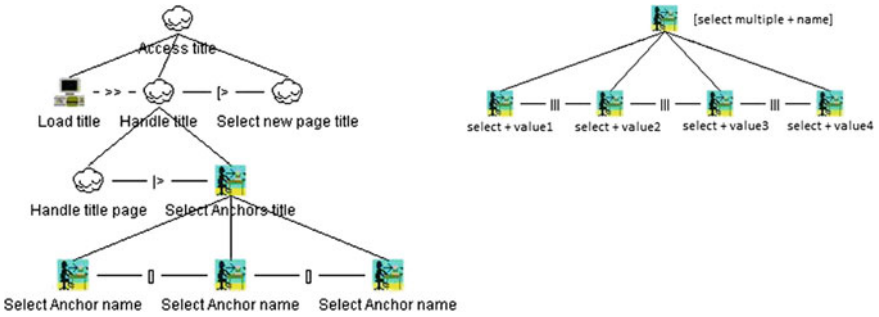

Fig. 18.15 From AUI to CTT: the resulting CTT model after translating elementary connections associated with HTML anchors (*left part*) and multiple choice elements (*right part*)

when they are anchors, this means that the navigation remains within the same page/presentation at hand; therefore, the "Handle title" node created in the first step will be expanded. This will be done by adding two new tasks (an abstract task and an interactive task) combined through a suspend/resume operator (see "Handle title page" |> "Select Anchors title" in Fig. 18.15, left part). Then, for each anchor found, an interactive task is created as child of the lastly created task. All such interactive tasks will be linked together through a choice operator (see Fig. 18.15, left part).

In the next step, the algorithm continues by analysing the type of the AUI interactors included in the AUI presentation and translating each of them into the corresponding CTT task models. For instance, if the AUI contains an element supporting a *multiple choice*, all the choice elements referring to the same multiple choice elements are translated into interactive tasks linked together through an interleaving operator (see Fig. 18.15, right part). In Fig. 18.16, a screenshot of the tool showing an example of translation of an AUI description into a CTT task model specification is shown.

Fig. 18.16 Tool for reverse engineering CTT task models

18.8 Conclusions and Future Work

We have presented an integrated set of model-based languages and associated tools for supporting design and development of interactive applications also using various modalities. They can be applied in various domains such as safety-critical systems (see for example at https://www.eurocontrol.int/ehp/?q=node/1617), ERP, workflow management systems and Ambient Assisted Living.

Some of such tools have been used by a broad community since they are publicly available (see http://giove.isti.cnr.it/tools.php), and over time, external use has provided suggestions for small improvements.

Future work will be dedicated to investigating how they can be exploited in combination with agile methodologies. This represents an interesting challenge since modelling requires some time and this may conflict with the fast pace adopted

in such methodologies. However, the use of model-based tools able to support fast prototyping can be able to address this issue. Another important area is how to exploit task models in analysing user behaviours. Previous tools, such as WebRemUsine (Paganelli and Paternò 2003a), have addressed how to compare client side Web logs representing actual behaviour with desired behaviour represented by the task model. It can be interesting to extend this approach to analyse broader human behaviour detected through various sensors and compare it with that described by task models in order to identify potential issues.

We also plan to continue to carry out studies in order to investigate improvements for the usability of the languages and the associated tools.

References

Anzalone D, Manca M, Paternò F, Santoro C (2015) Responsive task modelling. In: Proceedings of ACM SIGCHI symposium on engineering interactive computing systems, pp 126–131

Bernardi ML, Di Lucca GA, Distante D (2009a) The RE-UWA approach to recover user centered conceptual models from web applications. Int J Softw Tools Technol Transf 11(6):485–501

Berti S, Correani F, Paternò F, Santoro C (2004) The TERESA XML language for the description of interactive systems at multiple abstraction levels. In: Proceedings workshop on developing user interfaces with XML: advances on user interface description languages, pp 103–110

Caffiau S, Scapin D, Girard P, Baron M, Jambon F (2010) Increasing the expressive power of task analysis: systematic comparison and empirical assessment of tool-supported task models. Interact Comput 22(6):569–593 (2010)

Calvary G, Coutaz J, Thevenin D, Bouillon L, Florins M, Limbourg Q, Souchon N, Vanderdonckt J, Marucci L, Paternò F (2002) The CAMELEON reference framework. Deliverable D 1

Cockburn A, Karlson A, Bederson B (2008) A review of overview+detail, zooming, and focus +context interfaces. ACM Comput Surv 41(1):2

Coutaz J, Nigay L, Salber D, Blandford A, May J, Young R (1995) Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. Proc Interact 1995:115–120

da Silva CBE (2010) Reverse engineering of rich internet applications. Master thesis, Minho University, 2010

Dixon M, Laput G, Fogarty J (2014) Pixel-based methods for widget state and style in a runtime implementation of sliding widgets. In: Proceedings of annual conference on human factors in computing systems, pp 2231–2240

Duarte C, Carriço L (2006) A conceptual framework for developing adaptive multimodal applications. In: Proceedings of IUI 2006, pp 132–139

Eggers J, Hülsmann A, Szwillus G (2013) Aufgabenmodellierung am Multi-Touch-Tisch. In: Boll S, Maaß S, Malaka R (eds) Mensch & Computer 2013: Interaktive Vielfalt, pp 325–328

Li P, Wohlstadter E (2008) View-based maintenance of graphical user interfaces. In: Proceedings of 7th international conference on aspect-oriented software development, p 156

Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López-Jaquero V (2005) UsiXML: a language supporting multi-path development of user interfaces. In: Proceedings of engineering human computer interaction and interactive systems, pp 200–220

Maras J, Stula M, Carlson J (2011) Reusing web application user-interface. Lect Notes Comput Sci 6757:228–242

Martinie C, Palanque P, Winckler M (2011) Structuring and composition mechanisms to address scalability issues in task models. In: Proceedings of INTERACT, pp 589–609

Morgado IC, Paiva AC, Pascoal Faria J (2010) Dynamic reverse engineering of graphical user interfaces. Int J Adv Softw 5(3):224–236

Mori G, Paternò F, Santoro C (2002) CTTE: support for developing and analyzing task models for interactive system design. IEEE Trans Softw Eng 28(8):797–813

Mori G, Paternò F, Santoro C (2004) Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Trans Softw Eng 30(8):507–520

Octavia JR, Vanacken L, Raymaekers C, Coninx K, Flerackers E (2010) Facilitating adaptation in virtual environments using a context-aware model-based design process. In: England D, Palanque P, Vanderdonckt J, Wild PJ (eds) Proceedings of TAMODIA 2009, pp 58–71

Paganelli L, Paternò F (2003a) Tools for remote usability evaluation of web applications through browser logs and task models. Behav Res Methods Instrum Comput Psychon Soc Publ 35 (3):369–378

Paganelli L, Paternò F (2003b) A tool for creating design models from web site code. Int J Softw Eng Knowl Eng World Sci Publ 13(2):169–189

Paternò F (1999) Model-based design and evaluation of interactive applications. Springer

Paternò F, Santoro C (2000) Integrating model checking and HCI tools to help designers verify user interfaces properties. In: Proceedings of DSV-IS'2000, pp 135–150

Paternò F, Santoro C, Scorcia A (2008) Automatically adapting web sites for mobile access through logical descriptions and dynamic analysis of interaction resources. Proc AVI 2008:260–267

Paternò F, Santoro C, Spano LDM (2009) A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans Comput Human Interact 16(4):19:1–19:30

Paternò F, Santoro C, Spano LD (2011) Engineering the authoring of usable service front ends. J Syst Softw 84:1806–1822

Raneburger D, Meixner G, Brambilla M (2013) Platform-independence in model-driven development of graphical user interfaces for multiple devices. In: Proceedings of ICSOFT, pp 180–195

Silva CE, Campos JC (2013) Combining static and dynamic analysis for the reverse engineering of web applications. In: Proceedings of 5th ACM SIGCHI symposium on engineering interactive computing systems, p 107

Sottet JS, Ganneau V, Calvary G, Demeure A, Favre JM, Demumieux R (2007) Model-driven adaptation for plastic user interfaces. In: Baranauskas C, Abascal J, Barbosa SDJ (eds) Proceedings of INTERACT 2007, pp 397–410

Spano LD, Fenu G (2014) IceTT: a responsive visualization for task models. In: Proceedings of the 2014 ACM SIGCHI symposium on engineering interactive computing systems. ACM, pp 197–200