Cascading Dialog Modeling with UsiXML

1 IRIT, Université Toulouse 3, France, 118 route de Narbonne, 31062 Toulouse cedex 9 (France), winckler@irit.fr – http://liihs.irit.fr/winckler/
2 Belgian Lab. of Computer-Human Interaction, Louvain School of Management, Université catholique de Louvain, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium) jean.vanderdonckt@uclouvain.be – http://www.isys.ucl.ac.be/bchi
3 Federal University of Rio Grande do Sul (UFRGS), Caixa Postal 15064, 91501970 Porto Alegre (Brazil), fmtrindade@inf.ufrgs.br

Abstract. In the last years User Interface Description Languages (UIDL) appeared as a suitable solution for developing multi-target user interfaces. By applying appropriate model transformations, specifications of User Interfaces (UI) created with UIDLs can be reused and adapted according to constraints imposed by input/output devices, different contexts of use, or specific user preference. From a technical point of view transformational approaches can be used to adapt any aspect (i.e. structure, dialog and presentation) of user interfaces. In the practice, automatic adaptation of the dialog is not that easy. This paper discusses the role of the dialog modeling for the design of usable user interfaces and it proposes a design method which employs formal description techniques to deal with dialog modeling of user interface created with a UIDL called UsiXML.

Keywords: dialog modeling, multi-target user interfaces, user interface description languages (UIML), UsiXML.

1 Introduction

The large variety of computing systems available nowadays (e.g. low-weight desktop/notebook computers, cell phone, Personal Digital Assistant - PDA, Smartphone) have created a milestone for cost-effective development and fast delivery of multi-target interactive systems [9]. Quite often, multi-target user interfaces should be adapted to device’s constraints such as screen resolution and preferred interaction techniques (e.g. text, graphical, voice-based, gesture) which requires the inclusion of the notion of plasticity in the development process [3, 6].

The development of multi-target user interfaces requires the implementation of multiples versions of the same applications. The large availability of computing devices creates problems for ensuring consistent execution of the software along different platforms and it will ultimately increase the costs and time required for software construction and maintenance.

In more recent years, advances on the development of User Interface Description Languages (UIDL) allow designers to specify User Interfaces (UI) variations or a generic UI that is then further tailored for each respective context are a suitable
solution for the development of multi-target user interfaces. For example, UIDLs such as UIML [1], XIML [16], XUL [21], UMLi [7], among many others, have been successfully used for this purpose. In this scenario, the reference framework Cameleon [5] introduced a fresh perspective for the development of User Interface Description Languages (UIDL) by proposing 4 abstraction levels for the specification of user interface (i.e. task models, abstract UI, concrete UI and final UI). Such as multi-layer specification aims at giving more flexibility for specifying variations of the UI design, which is often required to generate the best solution according different contexts of the use. By successive transformations of abstract models, the specification of the UIs is completed and refined to more concrete specifications until it features executable device-platform-modality dependent specifications.

Typically, a UIDL must cover three different aspects of the UI: describe the static structure of the user interfaces (i.e. presentation part which ultimately include the description of user interface elements, e.g. widgets, and their composition), dynamic behavior (i.e. the dialog part, describing the dynamic relationships between components including event, actions, and behavioral constraints) and define the presentation attributes (i.e. look & feel properties for rendering the UI elements). Among the models involved in User Interface (UI) development, the dialog model is one of the most misunderstood and one of the most difficult to exploit [12]. Dialog models play a major role on UI design by capturing the dynamic aspects of the user interaction with the system which includes the specification of: relationship between presentation units (e.g. transitions between windows) as well as between UI elements (e.g. activate/deactivate buttons), events chain (i.e. including fusion/fission of events when multimodal interaction is involved) and integration with the functional core which requires mapping of events to actions according to predefined constraints enabling/disabling actions at runtime.

In this paper we analyze the specification of the dialog part when using a multi-layer description language. This paper introduces a method called “Cascading Dialog Modeling” that propose the combined use of transformational approaches and interactive (i.e. manual) edition of dialog models. The remainder of this paper is structured as follows: Section 2 defines the concepts that are useful for understand our approach, presented in Section 3, including the dimensions: model, approach, and tool, and illustrate how they have been implemented in a case study (here, a car rental system) in Section 4. Section 5 discusses the related work. Section 6 summarizes the benefits and discusses some future avenues to this work.

2 Basic Concepts

This section describes the basic concepts about modeling the dialog aspect of multi-target applications.
2.2 The Architecture of Dialog Arch

The basic assumption on dialog modeling is that it must describe the behavior of input and output devices, the general dialogue between the user and the application and the logical interaction provided by the interaction technique. These requirements for dialog modeling can be decomposed in layers as proposed by architecture Arch [2] which describes the various architectural components of an interactive application and the relationships between them as show in Fig. 1. For the purpose of this paper, the left hand side of the Arch (which concerns the functional core of the application) is not relevant. The steps that are considered in a complete dialog between the user and the system, from the physical input to the physical output (presentation rendering) are the following:

1) Low-level events (physical events) are generated by the physical devices and received by the Physical Interaction component;
2) Low-level events are transformed into logical events that independent of the employed input device;
3) Logical events are treated by the dialog controller which coordinate the sequence of events and the connection the functional core of the application;
4) Changes in the system state generates abstract rendering events;
5) Rendering events are reified into more concrete events offering a concrete rendering of the physical output.

According to the Arch architecture above the dialog model (step 3) can be isolated from technical details concerning the physical input events and rendering output. So that, changing the input/output devices (ex. mouse x touch screen) would not affect the specification of the dialog itself (this is true when considering the same interaction technique, ex. pointing). Conversely, different dialog models would be applied to different contexts of the use (ex. guided interaction through sequential screens or all-at-one interaction on a single screen) without a major impact on the input and/or output devices. Moreover, the same dialog model would be suited to different modalities with similar results. The dynamic adaptation of the dialog should be flexible enough in order to support any modification of the presentation, however the method allowing the adaptation are out of the scope of this paper.
2.1 Levels of Abstraction of User Interfaces

The framework Cameleon [5] proposes to describe user interfaces according four levels of abstractions: task models, abstract user interfaces (AUI), concrete user interface (CUI) and final user interface (FUI). By appropriate tool support it is possible to refine abstract user interface elements into more concrete specifications. According to the step considered, user interface specifications include more or less details about the user interface behavior, which lead designers to treat different dialog components (ex. state, condition, transitions, actions, etc) as exemplified in Table 1.

Table 1. Abstraction levels on dialog modeling.

<table>
<thead>
<tr>
<th>UI Abstraction level</th>
<th>Concepts</th>
<th>Dialog Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Model (TM)</td>
<td>Interactive tasks carried out by the end user &amp; domain objects</td>
<td>Tasks and dependencies between tasks</td>
</tr>
<tr>
<td>Abstract User Interface (AUI)</td>
<td>UI definition independent of any modality of interaction</td>
<td>Relationship between logical presentation units (e.g. transition between windows), logical events, abstract actions</td>
</tr>
<tr>
<td>Concrete User Interface (CUI)</td>
<td>Concretizes AUI into CIOs (widget sets found in popular graphical and vocal toolkits)</td>
<td>States, (concrete) events, parameters, actions, controls, changes on UI dialog according to events, generic method calls, etc</td>
</tr>
<tr>
<td>Final User Interface (FUI)</td>
<td>Operational UI that runs on a particular platform either by interpretation or by execution</td>
<td>“Physical” signature of events, platform specific method calls, etc</td>
</tr>
</tbody>
</table>

2.2 Specifying User Interface Dialogs

There are a large number of notations and techniques for describing the dialog aspect of the user interface. A review on the advances of dialog notations can be found in [12]. Hereafter we focus on some few, but representative, UIDLs which are presented in [Table 2].

Some notations are devoted to the dialog aspect of the user interface (example ICO [3], SCXML [19] and SWC [22]), while other UIDLs might also cover the structure and the presentation aspects. Is some cases the description of the dialog is supported by an external language (ex. XUL), however, quite often, the dialog is embedded into the UIDL, such as is the case of UsiXML, XUL and UIML.

Currently only UsiXML [11] and TERESA [13] have 4 levels of abstraction as proposed by the framework Cameleon. XUL and UIML’s dialog specification are oriented to implementation, which corresponds to the level CUI in the framework Cameleon.

As UIDLs must capture the intended dialog behavior, the specification of complex relationship between widgets quite often requires some kind of formal description technique such as Lotus, Petri Nets or Statecharts. However, this not avoids having some UIDLs implementing specific notations. It is noteworthy that UIDLs based on Petri Nets (such as ICO [3]) or based on StateCharts (SCXML[19] and SWC [22]) should also be considered as generic languages which can be employed at different levels of abstract of the user interface design.

UIDLs might include many mechanisms for specifying dynamic behavior such as the UI changes (corresponding to the local dialog changing properties of individual user interface components, ex. widgets), method calls (facilitating the integrating with
the application’s functional core), events, explicitly representation of current system state and explicitly representation of transitions changing the state of the system.

Data exchange can be done via passage of parameters along transitions, by reference to objects or both. All notations surveyed consider some kind of control for specifying constraints (i.e. conditions) during the execution of the dialog.

Table 2. Support for Dialog Modeling of some User Interface Description Languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>Aspects described</th>
<th>Specification</th>
<th>Levels of abstraction</th>
<th>Formalism/Notation language</th>
<th>Dynamic behavior described</th>
<th>Data exchange</th>
<th>Control (conditions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIXML</td>
<td>Presentation, Dialog, Structure</td>
<td>Embedded</td>
<td>Task Model, AUI, CUI, FUI</td>
<td>Specific notation for every abstraction level</td>
<td>transition, method call, ui change</td>
<td>parameters</td>
<td>Yes</td>
</tr>
<tr>
<td>XUL</td>
<td>Presentation, Dialog, Structure</td>
<td>Embedded</td>
<td>XBL Xul binding language</td>
<td>CUI</td>
<td>Specific notation</td>
<td>transition, method calls</td>
<td>parameters</td>
</tr>
<tr>
<td>ICO</td>
<td>Dialog</td>
<td>Embedded</td>
<td>Generic</td>
<td>Petri Net</td>
<td>transition, event, ui change, transition, state</td>
<td>reference</td>
<td>Yes</td>
</tr>
<tr>
<td>ICO</td>
<td>Dialog</td>
<td>Embedded</td>
<td>Generic</td>
<td>Statecharts</td>
<td>transition, method call, ui change, event, transition</td>
<td>parameters, reference</td>
<td>Yes</td>
</tr>
<tr>
<td>SCXXML</td>
<td>Presentation, Dialog, Structure</td>
<td>Embedded</td>
<td>Task model, AUI, CUI, FUI</td>
<td>Lotus</td>
<td>event, ui changes, transition</td>
<td>Parameters</td>
<td>Yes</td>
</tr>
<tr>
<td>BCE</td>
<td>Presentation, Dialog, Structure</td>
<td>Embedded</td>
<td>CUI</td>
<td>Specific notation</td>
<td>transition, method call, ui change, event, transition</td>
<td>parameters, reference</td>
<td>Yes</td>
</tr>
<tr>
<td>UML</td>
<td>Dialog</td>
<td>Embedded</td>
<td>Generic</td>
<td>Statecharts</td>
<td>event, ui changes, method call, event, transition, state</td>
<td>Parameters</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3 A Method for dealing with multi-level dialog specification

The proposed method is based on the following shortcomings:

- **Autonomy of the dialog** aspect with respect to the structure and the presentation of the UI; the autonomy implies that for any UI model describing the user interface components there might have one or more dialog models that can be associated to it in order to support different design options. Thus, a separate approach will provide a system that has a high degree of machine execution. Moreover, the separation of the dialog might lead to the reusability of some specifications and improve readability.
• **Use of formal description technique** for reducing the ambiguity of specification; This requirement is also important for implementing tool support;

• **Use of some graphical representation for the dialog.** This is an important requirement for improving the readability of specifications;

• **Combined use of automated and manual transformations** of abstract UI specification into more concrete UI. Automated transformations might improve productivity but designer should be able to modify the dialog afterwards;

• **No imposed start point for dialog specifications.** It is advisable to start by task models. However, some designers would prefer to start with more concrete dialog models and then refine them until the implementation; conversely, abstractions can be defined after deep analysis of existing concrete models.

3.1 **Notations**

The method proposed relies onUIDLs able to cover different level of abstraction and independence of dialog towards the user interface. For the purpose of this paper we employ two notations: UsiXML [11] to describe the structure and the presentation aspects of the user interface, and SWC [22] to describe the dialog.

UsiXML (USer Interface eXtensible Markup Language) is defined in a set of XML schemas. Each schema corresponds to one of the models in the scope of the language. UsiXML consists of a User Interface Description Language (UIDL) that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics. It describes at a high level of abstraction the constituting elements of the UI of an application: widgets, controls, containers, modalities, interaction techniques, etc. Several tools exist for editing specification using UsiXML at different level of abstraction, such as SkechXML, GraphiXML, IdealXML, TransformiXML, among others. The interest on UsiXML is the fact that it supports all fours levels of abstraction considered in this paper. Despite of that, UsiXML do not impose any particular development process so that designers are free to choose the abstract level the most appropriate to start their projects.

StateWebCharts notation (SWC) was originally proposed to specify dynamic behavior of Web applications. SWC is a formal description technique based on Harel’s StateCharts. States in SWC are represented according to their function in the modeling: they can be static, dynamic, transient or external. Additionally, SWC transitions explicitly represent the agent activating it (e.g. user actions are graphically drawn as continuous arrows while transitions triggered by system or completion events are drawn as dashed arrows). The interest on SWC for this paper remains on the full support to describe events and the notion of containers associate to states which can be easily mapped to UsiXML containers. Further information about these notations and the proper mapping between then is given along the case study on section 4.
3.2 Step-wise method

The so called “Cascading Dialog Modeling” presented in this section proposes the combined use of transformational approaches and interactive (i.e. manual) edition of dialog models. The name “cascade” is a reference for the fact that, similar to other user interface models, dialog models can be derive from abstract to more concrete specification. The general reification schema is presented by Fig. 2.

![Fig. 2. Dialog reification schema.](image)

The reification schema presented is composed of the following steps: 1) a task model is produced; 2) an Abstract Dialog Model can be generated automatically from task models using transformation rules. In this case, the dialog at this level is limited to the relationship that can be inferred from task models. Designers must create dialog specifications using external tools. Abstract UI can also be created manually in the absence of task models. Appropriate mapping is required to connect the Abstract UI and the Abstract Dialog. 3) A Concrete Dialog Model will be generated from the Abstract Dialog Model based on transformation rules. More Concrete Dialog Components will be added manually according to design choices. 4) The Final UI Dialog Control is generated from Concrete Dialog Control to copy with the target platform.

Designers could start working the dialog at any step of the abstraction levels presented by Fig. 2 by reusing specifications produced via a transformational approach or creating specification for both UI components and dialog at each level. The mapping of between the dialog specification with SWC and others components of the user interface in UsiXML is ensured by mapping tables as presented in Table 3.
Table 3. Mapping scheme between UsiXML and SWC constructs

<table>
<thead>
<tr>
<th>Abstraction level of UI</th>
<th>UsiXML Construct</th>
<th>SWC Constructs</th>
<th>Description of Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Model (TM)</td>
<td>Task</td>
<td>-</td>
<td>User tasks</td>
</tr>
<tr>
<td>Abstract User Interface</td>
<td>abstractContainer</td>
<td>compound states</td>
<td>High level containers for UI components</td>
</tr>
<tr>
<td></td>
<td>abstractIndividualComponent</td>
<td>basic states</td>
<td>UI containers (e.g. presentation units)</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>transitions</td>
<td>Relationships between containers</td>
</tr>
<tr>
<td>Concrete User Interface (CUI)</td>
<td>window</td>
<td>basic state</td>
<td>UI components featuring containers</td>
</tr>
<tr>
<td></td>
<td>behavior</td>
<td>transition</td>
<td>Definition of relationships between containers</td>
</tr>
<tr>
<td></td>
<td>event</td>
<td>event</td>
<td>Events raising</td>
</tr>
<tr>
<td></td>
<td>action</td>
<td>action</td>
<td>Behavior associated to events</td>
</tr>
<tr>
<td></td>
<td>methodCall / transition / uichange</td>
<td>action type</td>
<td>Action executed when event is triggered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>condition</td>
<td>Pre-condition associated to actions</td>
</tr>
<tr>
<td></td>
<td>parameters</td>
<td>parameters</td>
<td>Data exchange format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user transitions</td>
<td>User initiated actions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system transitions</td>
<td>System initiated actions (e.g. timed transitions)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transient states</td>
<td>Non-deterministic behavior of functional core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>history states</td>
<td>Memory for recent states</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end states</td>
<td>Notification of end of system execution</td>
</tr>
</tbody>
</table>

3.3 Tool support

Currently there are some available tools for dealing with different levels of abstraction of UI. In some cases (i.e. TERESA and GraphiXML) the tool supports both the specification and its successive refinement while; in other cases (i.e. IdealXML, SketchXML, FlashiXML and RenderXML) the support for the dialog specification is limited to a certain level but more abstract specifications produced by other tools can be reused. The dialog using SWC notation is support by the SWCEditor [23] that would cover all the levels of specification AUI, CUI and FUI. The automated transformation of models is ensured by the tool TransformiXML [17].

Table 4. Tool support for User Interface Specification.

<table>
<thead>
<tr>
<th>Abstraction level of UI</th>
<th>Tool (examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Model (TM)</td>
<td>IdealXML, TERESA</td>
</tr>
<tr>
<td>Abstract User Interface (AUI)</td>
<td>Ideal XML, GraphiXML, TERESA, SWCEditor</td>
</tr>
<tr>
<td>Concrete User Interface (CUI)</td>
<td>SketchXML, GraphiXML, TERESA, SWCEditor</td>
</tr>
<tr>
<td>Final User Interface (FUI)</td>
<td>RenderXML, FlashiXML</td>
</tr>
</tbody>
</table>
4 Case Study

The case study concerns a simple car rental system allowing users to choose a car, book and pay a reservation and print a receipt. The detailed case study can be found in [17] (pp. 140-164). The next sections present the car rental system featuring 3 levels of abstraction (task model, AUI and CUI); the level FUI is similar to the CUI (refining dialog primitives to target platforms) so, it will not be described hereafter.

4.1 Task Model

The task model considered for the car rental application is presented in Fig. 3.a. The sequence for execution of sub-tasks could follow different orders thus originating different scenarios. We limit our discussion to a single scenario presented in Fig. 3.b.

![Task Model](image)

**Fig. 3.** Specification of task models: a) task model using IdealXML; b) a scenario.

In Fig. 4 we present the task model according to the UsiXML syntax as it is generated by the tool IdealXML. One might notice that all relationships and dependencies among tasks are preserved at this level (see lines 14 and 26 for enabling tasks and 18 and 22 for undetermined choices) so that many scenarios can be extracted.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<taskmodel>
  <!--Tasks-->
  <task id="st0task0" name="RentCar" type="abstraction">
    <task id="st0task2" name="DefinePreferences" type="interaction">
      <task id="st0task3" name="DefineRentalPreferences" type="interaction"/>
      <task id="st0task4" name="DetermineCar" type="interaction"/>
      <task id="st0task5" name="DefinePayment" type="interaction"/>
    </task>
    <task id="st0task6" name="ProcessPayment" type="application"/>
    <task id="st0task7" name="ConfirmRentalInformations" type="application"/>
  </task>
  <!--Tasks relationships-->
  <enabling>
    <source sourceId="st0task2"/>
    <target targetId="st0task6"/>
  </enabling>
  <undeterministicChoice>
    <source sourceId="st0task3"/>
    <target targetId="st0task4"/>
  </undeterministicChoice>
</taskmodel>
```
Fig. 4. UsiXML specification of task models for a car rental system.

4.2 Abstract User Interface (AUI)

Once we have defined the task models, it is possible to generate the abstract model for the user interface. The figure Fig. 5 depicts the corresponding abstract user interface for the task model presented above.

Fig. 5. Abstract User Interface as depicted by IdealXML.

The abstract model provides definitions for user interfaces that are independent of any modality of interaction. By using appropriate transformation rules, it is possible to generate abstract containers from task definitions as presented by Fig. 6. Abstract containers correspond to the static part of the user interface.

Fig. 6. UsiXML specification of abstract models for a car rental system.

At this step one must identify two common dynamic behaviors: transitions between different presentation units, the so called interaction (ex. 7); or the so called application which will be refined to method calls in the concrete user interface (ex. Fig. 6, line 23). The so called interaction behavior corresponds to local dialog control; its implementation is very simple as it just proceeds to the next presentation unit. The
so called *Interaction* behavior has a strong impact on the dialog of the application as
their execution might affect the sequencing of the next task. For example, the
execution of the task *ProcessPayment* might return at least two possible states for the
systems: *successful payment* or *payment fail*. Such as dynamic behavior is described
in the dialog model presented by Fig. 7. In Fig. 7, continuous lines on transitions (i.e.
t4 and t5) correspond to interactive tasks which can be automatically refined by
successive transformation of task models whilst dashed lines (i.e. t6) correspond to a
behavior that should be defined manually by the designer.

![Fig. 7. Abstract Dialog modeling with SWC for a car rental system.](image)

It is noteworthy that the dialog at this step is also independent of the platform.
Further refinement is also required in order to complete the integration with the
functional core of the application. The mapping between states and transitions of
SWC to UsiXML components is made manually by choosing from the UsiXML
specification the components that fits the best to the purpose of the dialog. In the
example presented at Fig. 7, the state *DefinePreferences* is mapped to the
abstractContainer named *DefinePreferences* (see line 4 of Fig. 6).

4.3 Concrete User Interface (CUI)

At this step some modality constraints can be added into the design. There are many
possible scenarios for developing dialog models according to the modality chosen.
Due to space reasons we limited a single scenario but that could have 2 possible
dialog models. The first case considers a dialog model for interactions on a single
presentation unit. For the second case, user interaction is supported along three
different presentation units. The first scenario (i.e. a single presentation unit) would
be suitable for large displays where users can freely choose the order of filling in the
forms whilst the second scenario (i.e. several presentation units) is suitable for small
displays (e.g. PDA) or to context of use where users need to be more guided during
interaction (e.g. vocal interaction on cell phones).
Fig. 8. Concrete User Interface Specification using SketchiXML.

Fig. 9 presents the corresponding CUI specification in UsiXML for the single presentation unit depicted in Fig. 8.a.

Fig. 9. UsiXML Concrete User Interface Specification for a single presentation unit.

In Fig. 10 we propose four design options for the concrete dialog. The option a) (single presentation unit) corresponds to the dialog modeling for the single presentation depicted in Fig. 8.a. The mappings for connecting the SWC specification with the other components of the UsiXML description are in bold face. The operational execution of the model Fig. 10.a is the following: once the state DefinePreferences is reached, all user interface components in the mapping are shown in a single presentation unit. The transitions in SWC are implemented according to events, actions and method calls mapped from UsiXML controls (ex. Fig. 9, line 11, 12 and 13).

Fig. 10.b, c and d, propose alternative interaction behavior for the multiple presentation units depicted in Fig. 8.b. In all these examples, the mapping to concrete components also include the sub set of containers named definePreferences, determineCar and definePayment, which were previously identified at the step AUI (see section 4.2). The most important differences concerns how the states are
connected to each other. It noteworthy that these design options only affect the specification of the dialog and the UsiXML remain the same. As a consequence, a dialog model not implies a specific modality as any of the design options are suitable for rendering the user interface via different channels.

Fig. 10. Design option for dialog at the level Concrete Specification of the User Interface.

5 Related Work

Several works have been done on the design and specification of the dialog aspect of the user interfaces. Considering the organization of complex dialog structures, one should mention the hierarchical events proposed by Kosbie [10] which demonstrates how high level events can be identified and reified to low-level events triggered by user interface devices. Important improvements have also been done towards formal description techniques for the specification of complex dialog behavior. In this respect, it is noteworthy the ICO formalism [14], based on Petri Nets, allows more expressive and modular dialog specifications than the earlier attempts on formal methods for describing fusion/fission of complex events as they occurs in multimodal interaction techniques [14]. The organization of dialog models toward independent, modular and self-contained dialog structures have been a main target for developing complex interactive systems [8]. These previous work have mainly address the case of the organization of the dialog according to a single implementation.

As far as multi-target user interfaces is a concerns, only a few work have considered multi-level dialog specification. Book and Gruhn [4] have proposed the use of external dialogs for treating different presentation channels for multimodal Web applications. Their approach is based on a formal description technique called Dialog Flow Notation (DFN) that provides constructs for the design of modular navigation models for multimodal Web applications. Mori, Paterno and Santoro [13] have proposed a design method and tool called TERESA for dealing with the progressive transformation of abstract description of the user interface to final
implementations whilst try to preserve the usability and plasticity of the user interface. Similarly, Luyten et al [12] have proposed a transformational approach for derive final user interface dialog from task models. These solutions are based on top-down approach of development with little flexibility for implementing design options.

6 Conclusion and future work

This paper discussed several issues related to multi-level dialog specifications for multi-target user interface User Interface Description Languages. Additionally it proposes a design method combining two currently available UIDLs: UsiXML and SWC.

This work tried to demonstrate that transformational approaches and manual dialog specification can be combined to promote the reification of abstract user interface into more concrete user interfaces. The approach presented is duly based on the clear separation of the dialog aspect of the other components of the user interface. Such as separation presents several advantages such as it improves the readability of models, it supports reuse of specifications and it might help the management of versions according different design choices. Moreover, this method is proposed as a white box, which means that UI components can be built with different tools (ex. IdealXML, SketchXML, GraphiXML, etc) and coupled to external dialogs. The advantage of such an approach is that one can reuse knowledge and tools for dealing with dialog models and study the limits of dialog specification at different levels of abstraction.

Dialog models created with SWC can be simulated by the SWCEditor [23] so that, the behavior of the application can be inspected at any time.

The current work is limited to dialog specified produced with the SWC notation. However, we suggest that it could be generalized for other dialog description techniques with similar expressive power. Another limitation is the fact no complex multimodal interaction techniques requiring fission/fusion of events, for example, has been taken into account. Such as situation will be investigated in future work.

References