Cascading Dialog Modeling with UsiXML

Marco Winckler[1,2], Jean Vanderdonckt[2], Adrian Stanciulescu[2], and Francisco Trindade[3]

[1] IRIT, Université Toulouse 3, France, 118 route de Narbonne, F-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. This paper discusses multi-level dialog specifications for user interfaces of multi-target interactive systems and it proposes a step-wise method that combines a transformational approach for model-to-model derivation and an interactive editing of dialog models for tailoring the derived models. This method provides a synthesis of existing solutions for dialog modeling using a XML-based User Interface Description Language, UsiXML, along with StateWebCharts notation for expressing the dialog at a high level of abstraction. Our aim is to push forward the design and reuse of dialog specifications throughout several levels of abstraction ranging from task and domain models until the final user interface thanks to a mechanism based on cascading style sheets. In this way, it is expected that the dialog properties are not only inherited from one level to another but also are made much more reusable than in the past.

Keywords: cascading style sheet, dialog modeling, multi-target user interfaces, StateWebCharts, user interface description language, 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 [21]. 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]. Quite often, it is required the development multiples versions of the same applications. The availability of many computing devices creates problems for ensuring cross-consistent execution of the software along different platforms and it will ultimately increase the costs and time required for software construction and maintenance.

T.C.N. Graham and P. Palanque (Eds.): DSVIS 2008, LNCS 5136, pp. 121–135, 2008.

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. For example, UIDLs such as UIML [1], XIML [15], XUL [20], UMLi [7], among many others, have been successfully used for this purpose. In this scenario, the Cameleon reference framework [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.

We assume that an UIDL must cover three different aspects of the UI: the static structure of the user interfaces (i.e. including the description of user interface elements - e.g., widgets - and their composition), the dynamic behavior (i.e., the dialog part, describing the dynamic relationships between components including event, actions, and behavioral constraints) and the presentation attributes (i.e., look & feel properties of UI elements). However, this is not always the case as many UIDLs do not provide full modeling support for all theses aspects. In particular, dialog model is one of the most difficult to exploit and it is often misunderstood [11].

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. In particular, it presents a method that combines 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 which is presented in Section 3, 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.

2.1 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.

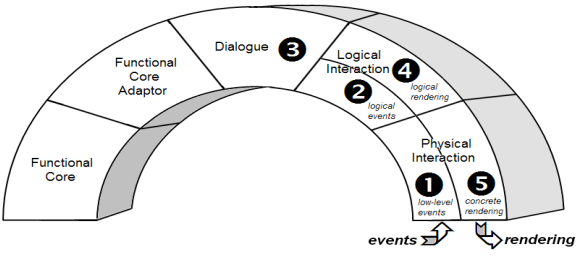


Fig. 1. The architecture Arch

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 (e.g., 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.

2.2 Levels of Abstraction of User Interfaces

The Cameleon Reference Framework [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

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

2.3 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 [11]. 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 (for example, ICO [3], SCXML [18] and SWC [21]), 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 (e.g., XUL), however, quite often, the dialog is embedded into the UIDL, such as is the case of UsiXML, XUL and UIML.

Currently only UsiXML [10] and TERESA XML [12] have 4 levels of abstraction as proposed by the Cameleon Reference Framework. XUL and UIML's dialog specification are oriented to implementation, which corresponds to the level CUI and FUI 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[18] and SWC [21]) 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

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

Language	Aspects described	Specification	Levels of abstraction	Formalism/ Notation language	Dynamic behavior described	Data exchange	Control (conditions)
USIXML	Presentation, Dialog, Structure	Embedded	Task Model, AUI, CUI, FUI	Specific notation for every abstraction level	transition, method call, ui change	parameters	Yes
XUL	Presentation, Dialog, Structure	XBL Xul binding language	CUI, FUI	Specific notation	transition, method calls	parameters	Yes
ICO	Dialog	Embedded	Generic	Petri Net	ui changes method call, event, transition	reference	Yes
SCXML	Dialog	Embedded	Generic	Statecharts	event, method call, transition, state	parameter, reference	Yes
TERESA-XML	Presentation, Dialog, Structure	Embedded	Task model, AUI, CUI, FUI	Lotus	event, ui changes, transition	Parameters	Yes
UIML	Presentation, Dialog, Structure	Embedded	CUI, FUI	Specific notation	ui changes method call, event, transition	parameters, reference	Yes
SWC	Dialog	Embedded	Generic	Statecharts	ui changes method call, event, transition, state	Parameters	Yes

state and explicitly representation of *transitions* changing the state of the system. *Date 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.

3 A Method for Dealing with Multi-level Dialog Specification

The proposed method is based on the following shortcomings:

- *Autonomy of the dialog* with respect to the structure and the presentation of the UI which implies that for any UI model describing the user interface components must have at least one dialog model supporting each design options. 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 on UIDLs 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 [10] to describe the structure and the presentation aspects of the user interface, and SWC [21] 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. 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 method 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.

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.

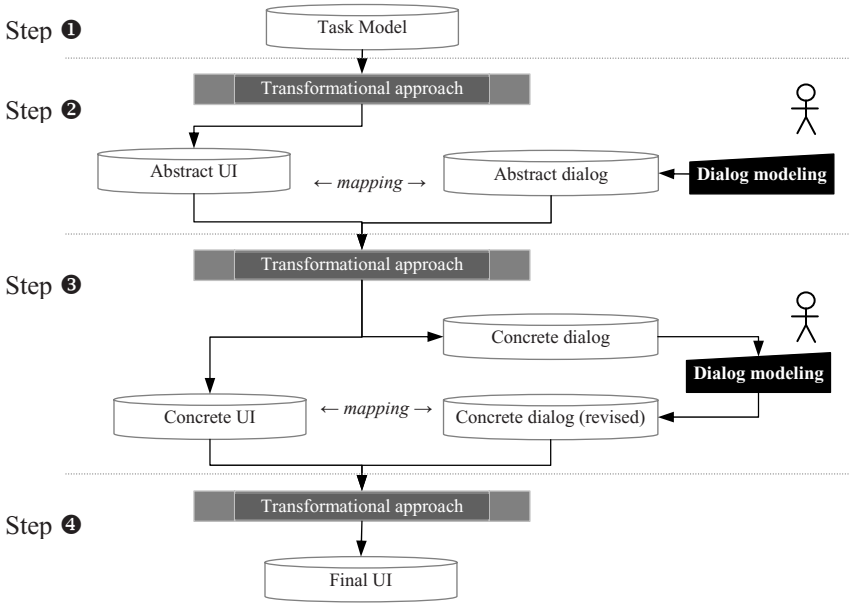


Fig. 2. Dialog reification schema

Table 3. Mapping scheme between UsiXML and SWC constructs

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

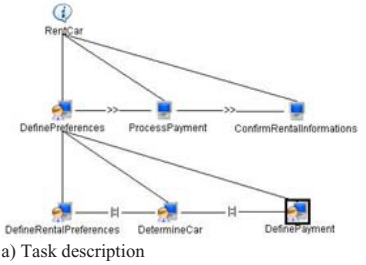
Designers could start working the dialog at any step of the abstraction levels presented by Fig. 2 by reusing specifications produced via a transformational approach or creating specification for both UI components and dialog at each level. The mapping of between the dialog specification with SWC and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

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 [16] (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.



a) Task description

Scenario 1
DefineRentalPreferences
DetermineCar
DefinePayment
ProcessPayment
ConfirmRentalInformations

b) Scenario for task model

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.

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. Fig. 5 presents the corresponding abstract user interface (only abstract containers - e.g. abstract windows – are shown) for the task model. The abstract model provides definitions for user interfaces that are independent of any modality of interaction. By using appropriate transformation rules, it possible to generate *abstract containers* from task definitions as presented in Fig. 6. Abstract containers correspond to the static part of the user interface.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--Tasks-->
   <taskmodel>
     <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>
   <undeterministicChoice>
     <source sourceId="st0task4"/>
     <target targetId="st0task5"/>
    </undeterministicChoice>
   <enabling>
     <source sourceId="st0task6"/>
     <target targetId="st0task7"/>
   </enabling>
</taskmodel>
```

Fig. 4. UsiXML specification of task models for a car rental system

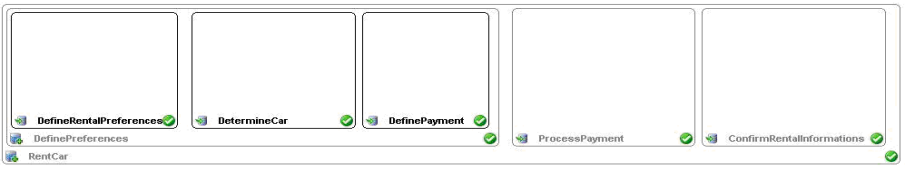


Fig. 5. Abstract User Interface as depicted by IdealXML

```
<?xml version="1.0" encoding="UTF-8"?>
  <auimodel>
    <abstractContainer id="idaio00" name="RentCar">
      <abstractContainer id="idaio01" name="DefinePreferences">
        <abstractIndividualComponent id="idaio02" name="DefineRentalPreferences">
          <abstractIndividualComponent id="idaio03" name="idaio03">
            <control id="idaio04" name="idaio04" actionType="interaction" action="dialog.defineRentalPreferences" />
          </abstractIndividualComponent>
        </abstractIndividualComponent>
        <abstractIndividualComponent id="idaio05" name="DetermineCar">
          <abstractIndividualComponent id="idaio06" name=" idaio06">
            <control id="idaio07" name="idaio07" actionType="interaction" action="dialog.determineCar" />
          </abstractIndividualComponent>
        </abstractIndividualComponent>
        <abstractIndividualComponent id="idaio08" name="DefinePayment">
          <abstractIndividualComponent id="idaio09" name="idaio09">
            <control id="idaio10" name="idaio10" actionType="interaction" action="dialog.definePayment" />
          </abstractIndividualComponent>
        </abstractIndividualComponent>
      </abstractContainer>
      <abstractIndividualComponent id="idaio11" name="ProcessPayment">
        <abstractIndividualComponent id="idaio12" name="idaio12">
          <control id="idaio13" name="idaio13" actionType="application" action="dialog.processPayment" />
        </abstractIndividualComponent>
      </abstractIndividualComponent>
      <abstractIndividualComponent id="idaio14" name="ConfirmRentalInformations">
        <abstractIndividualComponent id="idaio15" name="idaio15">
          <control id="idaio16" name="idaio16" actionType="application" action="dialog.confirmRentalInformations" />
        </abstractIndividualComponent>
      </abstractIndividualComponent>
    </abstractContainer>
  </auimodel>
```

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* (Fig. 6, line 7); or the so called *application* which will be refined to *method calls* in the concrete user interface (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.

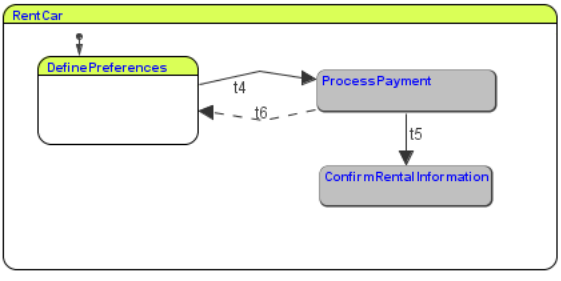
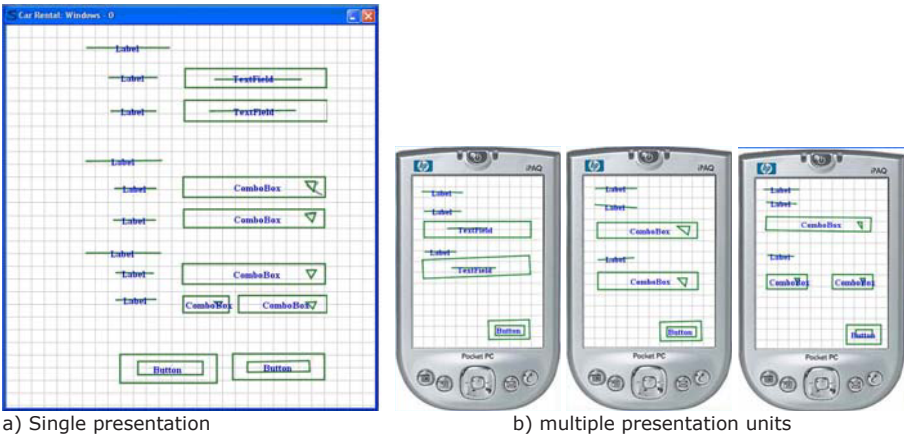


Fig. 7. Abstract Dialog modeling with SWC for a car rental system

It is noteworthy that the dialog at this step is also independent of the platform. Further refinement is 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).



a) Single presentation b) multiple presentation units

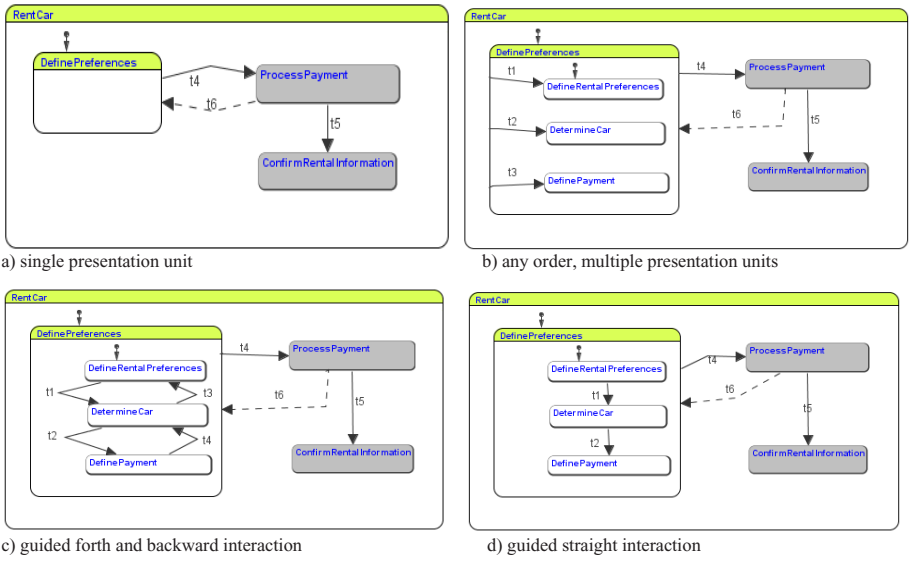
Fig. 8. Concrete User Interface Specification using SketchiXML

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

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.      <uiModel id="Car_Rental" … >
3.        <head>
4.          <version modifDate="2007-12-19T15:45:21.031-02:00"/>
5.          <authorName>SketchiXML</authorName>
6.        </head>
7.        <cuiModel id="Car_Rental-cui" name="Car_Rental-cui">
8.            <window id="window_0" name="window_0" … >
…
9.                <comboBox id="ComboBox_0" name="ComboBox_0"…>
10.               <behavior>
11.                 <event id="evt_0" eventType="change" eventContext="Button_0"/>
12.                 <action id="act_0" name="act_0">
13.                     <methodCall methodName="dialog.carTypeChange">
14.                 <action>
15.               </behavior>
16.             </comboBox>
…
17.             <button id="Button_1" name="Button_1">
18.             <behavior>
19.                 <event id="evt_1" eventType="click" eventContext="Button_1"/>
20.                   <action id="act_1" name="act_1">
21.                     <methodCall methodName="dialog.defineRentalPreferences">
22.                     <methodCall methodName="dialog.determineCar">
23.                     <methodCall methodName="dialog.definePayment">
24.                   <action>
25.               </behavior>
26.             </button>
…
27.           </window>
28.   </cuiModel>
```

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).



a) single presentation unit

b) any order, multiple presentation units

c) guided forth and backward interaction

d) guided straight interaction

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

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 does not imply a specific modality as any of the design options are suitable for rendering the user interface via different channels.

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 [9] 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 [3], 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 [13]. 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 [12] 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. [11] 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. This method is clearly based on open standards like UsiXML which make possible to assemble UI elements built with different tools (for instance, IdealXML, SketchiXML, GrafiXML, see www.usixml.org) and couple them with external dialog specifications (for example, SWC). The advantage of such as 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

1. Ali, M.F., Pérez-Quiñones, M.A., Abrams, M.: Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.) Multiple User Interfaces: Engineering and Application Framework. John Wiley and Sons, New York (2003)
2. Bass, L., Pellegrino, R., Reed, S., Seacord, R., Sheppard, R., Szezur, M.R.: The Arch model: Seeheim revisited. In: User Interface Developer's workshop version 1.0 (1991)

3. Bastide, R., Palanque, P.: A Visual and Formal Glue Between Application and Interaction. Journal of Visual Language and Computing 10(5), 481–507 (1999)
4. Book, M., Gruhn, V.: Efficient Modeling of Hierarchical Dialog Flows for Multi-Channel Web Applications. In: Proc. of 30th Annual Int. Computer Software and Applications Conference COMPSAC 2006, Chicago, September 17-21, 2006, pp. 161–168. IEEE Computer Society, Los Alamitos (2006)
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting With Computers 15(3), 289–308 (2003)
6. Collignon, B., Vanderdonckt, J., Calvary, G.: An Intelligent Editor for Multi-Presentation User Interfaces. In: Proc. of 23rd Annual ACM Symposium on Applied Computing SAC 2008, March 16-20, 2008, pp. 1634–1641. ACM Press, New York (2008)
7. da Silva, P.P., Paton, N.W.: User Interface Modeling in UMLi. IEEE Software 20(4), 62–69 (2003)
8. Conversy, S., Eric, B., Navarre, D., Philippe, P.: Improving modularity of interactive software with the MDPC architecture. In: Proc. of Engineering Interactive Systems 2007 (IFIP WG2.7/13.4 10th Conference on Engineering Human Computer Interaction jointly organized with IFIP WG 13.2 1st Conference on Human Centred Software Engineering and DSVIS - 14th Conference on Design Specification and Verification of Interactive Systems) EIS 2007, Salamanca, March 22-24, 2007. Springer, Heidelberg (2007)
9. Kosbie, D.S.: Hierarchical events in graphical user interfaces. In: Proc. of ACM Conf. on Human factors in computing systems CHI 1994, Boston, April 1994, pp. 131–132. ACM Press, New York (2004)
10. Limbourg, Q., Vanderdonckt, J.: UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In: Matera, M., Comai, S. (eds.) Engineering Advanced Web Applications, pp. 325–338. Rinton Press, Paramus (2004)
11. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 203–217. Springer, Heidelberg (2003)
12. Mori, G., Paternò, F., Santoro, C.: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering 30(8), 507–520 (2004)
13. Navarre, D., Palanque, P., Bastide, R., Schyn, A., Winckler, M., Nedel, L., Freitas, C.M.D.S.: A Formal Description of Multimodal Interaction Techniques for Immersive Virtual Reality Applications. In: Costabile, M.F., Paternó, F. (eds.) INTERACT 2005. LNCS, vol. 3585, pp. 170–183. Springer, Heidelberg (2005)
14. Palanque, P., Bastide, R., Winckler, M.: Automatic Generation of Interactive Systems: Why A Task Model is not Enough. In: Proc. of 10th Int. Conf. on Human-Computer Interaction HCI International 2003, Heraklion, June 22-27, 2003, pp. 198–202. Lawrence Erlbaum Associates, Mahwah (2003)
15. Puerta, A., Eisenstein, J.: XIML: A Common Representation for Interaction Data. In: Proc. of 6th ACM Int. Conf. on Intelligent User Interfaces Conference IUI 2002, San Francisco, January 13-16, 2002, pp. 216–217. ACM Press, New York (2002)
16. Stanciulescu, A., Vanderdonckt, J.: Design Options for Multimodal Web Applications. In: Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI 2006, Bucharest, June 6-8, 2006, pp. 41–56. Springer, Heidelberg (2006)

17. Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F.: A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In: Proc. of 7th ACM Int. Conf. on Multimodal Interfaces ICMI 2005, Trento, October 4-6, 2005, pp. 259–266. ACM Press, New York (2005)
18. State Chart XML (SCXML): State Machine Notation for Control Abstraction. W3C Working Draft, February 21 (2007), http://www.w3.org/TR/scxml/
19. Trindade, F.M., Pimenta, M.S.: RenderXML – A Multi-platform Software Development Tool. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 292–297. Springer, Heidelberg (2007)
20. XML User Interface Language (XUL), Mozilla Foundation (January 2008), http://www.mozilla.org/projects/xul/
21. Weiser, M.: The world is not a desktop. Interactions 1(1), 7–8 (1994)
22. Winckler, M., Palanque, P.: StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 61–76. Springer, Heidelberg (2003)
23. Winckler, M., Barboni, E., Farenc, C., Palanque, P.: SWCEditor: a Model-Based Tool for Interactive Modelling of Web Navigation. In: Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI 2004, Funchal, January 14-16, 2004, pp. 55–66. Kluwer, Dordrecht (2005)