Multi-level Dialog Modeling in Highly Interactive Web Interfaces

Efrem Mbaki[1], Jean Vanderdonckt[1], Josefina Guerrero[1], Marco Winckler[2]
[1]Université catholique de Louvain, Louvain School of Management,
Belgian Lab. of Computer-Human Interaction, Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

[2]IRIT, Université Toulouse 3, France, 118 route de Narbonne,
F-31062 Toulouse cedex 9, France
http://liihs.irit.fr/winckler/, winckler@irit.fr

Abstract

As web user interfaces become more sophisticated both in functionalities and reactivity, the dialog of such user interfaces is highly interactive and therefore raises the need for abstracting these capabilities into an advanced dialog model that enables modeling such dialogs. To address this need, multi-level dialog modeling enables designers to model a dialog at two inter-related levels of abstraction (i.e., concrete and abstract dialog) and at five levels of granularity: object-level (dialog at the level of a particular objects such as a widget), low-level container (dialog at the last level of decomposition of user interface containers, such as a group box), intermediary-level container (dialog at any non-terminal level of decomposition such as a dialog box or a web page), within-application level (dialog at the level of an interactive application), and across applications-level (dialog across user interfaces of different interactive applications).

1. Introduction

Among all models involved in Model-Driven Engineering (MDE) of User Interfaces (UIs) of any interactive application in general or for a web application in particular, the dialog model is probably one of the most challenging remaining problems due to several reasons:

- *Lack of ontological definition* [12]: different terms, e.g., dialog, navigation, behavior, dynamics, conversation, the "feel", are inconsistently used to refer to the dynamic aspects of a UI, as opposed to the presentation, which refers to as the static aspects of a UI, e.g., its layout. In this paper, we define a *dialog model* as the model that captures all dynamic aspects of a UI behavior. This therefore includes dynamics at any level of any object that may appear in a UI. This definition will lead us to define five particular levels.
- *Lack of precise abstraction* [8]: in principle, MDE suggests three levels of abstraction (i.e., computing independent model, platform-independent model, and platform-specific model). These three levels are rarely observed in the area of dialog modeling where the platform-specific level remains predominant.
- *Lack of continuity* [12]: when two levels of abstractions are covered, it is not always obvious to see how model-to-model mappings (whether it is achieved through transformations or not) are ensured to establish and maintain continuity between them.
- *Lack of expressiveness* [20]: the demand for more sophisticated dialogs stems for a dialog model that is capable enough to accommodate the description of desired dynamic aspects, such as animations, transitions, the two traditional forms of adaptation (i.e., adaptability and adaptivity). A modern dialog model should be expressive enough to model dynamic aspects.
- *Risk for modeling complexity*: it is likely that a more expressive model tend to be more complex to define and therefore to use in general.

In this paper, we attempt to address the problem of dialog modeling in a comprehensive way in order to address the aforementioned shortcomings. For this purpose, Section 2 will report on significant work related to the area of dialog modeling. Section 3 will define the semantics of a dialog model respectively at the platform-specific level (concrete UI) and at the platform-independent level (abstract UI) with a derivation from a computing-independent model (task model). Section 3 will then define five levels of granularity. Section 4 will conclude the paper by presenting some future avenues of this work.

2. Related Work

Dialog models enable to reason about the UI behavior. Consequently, dialog models are often considered as a continuation of task model concepts. This explains why the task model has been extensively used to derive a dialog model, for instance, in an algorithmic way [13] or in a logical way supported by model-to-model transformations [12] and graph grammars [7,12]. We hereafter give a brief survey of dialog modeling methods that percolated into the field of Human-Computer Interaction (HCI) development methods [3, 8, 9, 16]:

- *Backus-Naur Form (BNF) grammars*: they are typically used to specify command languages [9]. Command languages express commands that modify the state of the UI on the user's initiative. Grammars are particularly good in detecting inconsistencies within command sets. An inconsistent UI may contain unordered or unpredictable interaction. Inconsistency renders the UI error prone and hard to learn. Grammars are both efficient and effective for expressing sequential commands or users actions in general, but become complex for multimodality.
- *State transition diagram* are a finite state machine representation that consists in a graph of nodes linked by edges [21]. Each node represents a particular state of the system. Each edge species the input (i.e., event) required to go from one state to another. State transition diagrams have been subject to several extensions [21] and specializations, like Statecharts [10] that provide a mean for specifying the dynamic behavior of the interface. State transition diagrams present several drawbacks for modeling the UI. Indeed, today's UI tend to be modeless where one state can lead to many states. Furthermore this can be done using many different widgets of the UI. Theses two requirements match the quality criteria of reachability and device multiplicity. In consequence, state transition diagrams are prone to a combinatorial explosion and tend to replace nodes by screen prints. In [14], the transition space is restricted to events and transitions that are triggered by window managers in graphical state transition diagrams, thus supporting only simple windows transitions [20]. Many other forms of dedicated state transition diagrams have been extensively used for dialog modeling without identifying which one is superior to another one with respect to various criteria: dialog charts [1], dialog flows [4], interaction object graph [5], Abstract Data Views [6], dialogue nets [11].
- *Statecharts*: similarly to state transition diagrams, they are supported by a graphical representation of dynamic aspects of systems [10]. Some work especially address the modeling of UI behavior with statecharts [18, 22]. Statecharts represent state variables with rounded rectangles called states. State changing mechanisms are represented with edges between states. State changing is triggered by events and can be further conditioned. Statecharts facilitate the representation of state nesting, state history, concurrency and external interrupts. Statecharts [10] propose solutions to the shortcomings of state transition diagrams: statecharts have representational capacities for modularity and abstraction. The number of states with respect to the complexity of the modeled system increases slower with statecharts than with state transition diagrams. Statecharts avoid the problem of duplicating states and transitions. States in statecharts are hierarchical and capable of representing different levels of abstraction. Statechart are more convenient for multimodal interfaces as they provide nesting facilities, external interrupt specification and concurrency representation. Statecharts have been also specialized for specifying the dialog of web interfaces through StateWebCharts, that can be edited via a SWCEditor [22].
- *And-Or graphs*: borrowed from Artificial Intelligence, AND-OR graphs have been used to branch to various sub-dialogs depending on conditions, for instance in [3]. And-or graphs have been expanded towards function chaining graphs [3] by combining them with data flow diagrams [19].
- *Event-Response Languages*: they treat input stream as a set of events [9]. Events are addressed to event handlers. Each handler responds to a specific type of event when activated. This type is specified in a condition clause. The body of the event generates another event, changes internal state of the system or calls an application procedure. Several formalisms are suited for event-response specification. They can be distinguished following their capacity in managing dialog state variables and concurrency control. Production rules and push-down automata [16] are often used to describe event-response specifications.
- *Petri Nets* are a graphical formalism associated with a formal notation. Petri nets are best suited to represent concurrency aspects in software systems. Petri nets represent systems with state variables called places (depicted as ellipses), and state-changing operators called transitions (depicted as rectangles. Connections between places and transitions are called arcs (represented by edges). State marking mechanism called tokens (represented by black solid circles distributed around places). State change is the consequence of a mechanism called firing. A transition is red when all of its input places contain tokens. Firing involves the redistribution of tokens in the net i.e., input tokens are withdrawn from input places and output tokens are added in output places. Like State Charts, Petri nets hold mechanisms to represent additional conditions and nested states. Petri nets