

COPYRIGHT NOTICE

This material is strictly CONFIDENTIAL. Any use of this material, including but not limited to, its analysis, examination, distribution, modification, reproduction or incorporation into other work is governed by the terms of the XIML Research License Agreement and is restricted to licensees under such agreement. Any use by non-licensees is strictly prohibited, is a violation of US and International trade and copyright laws, and is subject to severe penalties and claims for damages. Licensing information can be obtained by visiting <http://www.ximl.org/>. The reproduction by any means of this material is expressly prohibited except as noted in the terms of the XIML Research License Agreement. All digital or hard copy reproductions of this material must bear this Copyright Notice in its entirety.

© Copyright 1999-2002 RedWhale Software (<http://www.redwhale.com/>)

Using a Task Model to Derive Presentation and Dialog Structure

Jean Vanderdonckt, Quentin Limbourg

Université catholique de Louvain

IAG, Place des Doyens, 1

B-1348 Louvain-la-Neuve, Belgium

+32-10/47.85.25

{limbourg, vanderdonckt}@qant.ucl.ac.be

Jacob Eisenstein, Angel Puerta

RedWhale Software Corporation

Town & Country Village Suite 273-277

Palo Alto CA 94301, USA

+1 650-{321-1681, 325-4587}

{jacob, puerta}@redwhale.com

ABSTRACT

User-centered design typically begins with a task model, because this model describes the set of tasks that the user-interface should support, from the end user's point of view. User-interface modeling often seeks to develop concrete models of presentation and dialog structure, based on abstract models of task and domain information. Whereas the relevance of the domain model to these more concrete models is relatively well understood, less is known about how to exploit the task model to generate presentation and dialog information. In this paper, we explore the problem of generating mappings between elements in a task model and the concrete structure of the user-interface. This mapping problem is handled through the use of a decision tree, which classifies task relations using dialog properties such as visibility, observability, and browsability. For each classification, a presentation and dialog structure is recommended, using the Windows Transition graphical notation.

Keywords

User-interface modeling, knowledge-based user-interface design, user-centered design, dialog model, presentation model, task model

INTRODUCTION

User-interface modeling seeks to capture design knowledge in declarative and formal user models [4,8]. In a typical development life cycle, user-task elicitation [12] usually initiates the cycle by allowing the end user to specify in-

formation about the task model, and possibly about the domain and user models as well. This activity is often performed through the creation of a scenario. After that, the domain model may be exploited to obtain a data model for database design and application modeling. These abstract models are then refined in order to create a final, more concrete model of the running UI. We consider task modeling to be particularly important for two reasons: the task model serves as a starting point for the development of other models; it also helps to ensure that design remains user-centered [5,11,14].

Three mechanisms are used to establish mappings between these models as they are being developed:

- *Model derivation*: one or many unspecified models are derived from one or many already specified models according to transformation rules, the parameters of which are controlled by the designer [1,9,11].
- *Model linking/binding*: one or many already specified models are processed to establish or restore previously established relationships between the models elements [1,2].
- *Model composition*: one or many already specified models are partially or totally assembled either to rebuild the source models (as in reverse engineering) or to build new models [10].

In a comprehensive approach to UI modeling, several map-

ping problems present themselves [7]: task-dialog, task-presentation, domain-presentation, task-user, task-domain, and presentation-dialog. In this paper, we are interested by one particularly important combination of mappings: task-presentation-dialog. We will explore the relation between a UI's task model and its presentation and dialog structure.

The task model is chosen as the source model because it is a semantically rich model that has been that can be exploited to derive other models. The presentation and dialog models are chosen as target models because together they are responsible for both the look and feel of the final running user interface, and because their behaviors are tightly integrated.

The rest of this paper is structured as follows: we first review some related work that details some initial strategies for exploiting the task model as a source. We then define the notations used in this work: the Window Transitions notation for presentation-dialog modeling (more specifically, transitions between presentations) and the ConcurTaskTrees notation [5,6] for task modeling. The following section presents and explains two representative examples of derivation rules for their corresponding task-based ConcurTaskTrees operators. Next, we apply these derivation rules to an example UI, and discuss how the process could be supported by a software tool. Finally, we identify areas of future work.

RELATED WORK

Much of the existing research in user-interface modeling focuses on the task model. There are several reasons for this:

- The task model is the knowledge source where the UI is described in the user's own words, not in terms of the system or the designer [14].
- Relative to other models, the task model is highly expressive. Consequently, we can expect the expressiveness of the derived models also to be significant [6].
- When other types of models, such as data models or domain models, are used as a starting point, the derived tasks (e.g., insert, delete, modify, list, check, search, print, transfer) do not necessarily correspond to the user's tasks. In other words, the use of other source models can lead to a UI design that does not support a user-centered task model. This problem can be avoided when the derivation rules are domain-specific: for example, Teallach derives task, presentation and dialog models from a domain model provided by the underlying database management system, for the specific domain of data-intensive systems [1].

In order to derive presentation and dialog models from a task model, two types of tasks relations can be exploited: *structural relations*, in which a task is recursively decomposed into subtasks, ending with atomic actions that operate on domain objects; *temporal relations*, which provide constraints for ordering (sub-)tasks according to the task logic.

Tam et al. focus on structural relations: by browsing the hierarchical decomposition, the number of subtasks for each level and the number of levels can be computed to let the designer specify a preference ranging from one window presenting all elements corresponding to all subtasks to many windows presenting, for example, the different levels of subtasks, level by level [7,12].

SEGUIA [13] also bases its grouping algorithm on structural relations: a presentation unit is selected for each subtask of the top level, and each presentation unit is consequently decomposed into windows. These can be grouped according to different strategies as long as a threshold for some cognitive load metrics is not exceeded.

Paternò et al. have investigated how a presentation model can be derived from a task model [6] specified in the ConcurTaskTrees notation [5]. In this research, a bottom-up approach exploits the structural relations first, and then the temporal relations to group presentation elements based on the concept of an activation set. Each activation set groups the presentations corresponding to subtasks that can be carried out in some related way. TADEUS also generates both a presentation and a dialog model in dialog graphs from a task model according to predefined correspondence tables [9].

THE WINDOWS TRANSITION NOTATION

A *window transition* is hereby defined as any method to go from one source window (WS) to a target window (WT). A window is here considered more as a container widget than just a window, strictly defined: it can be a dialog box, a notebook, a tabbed dialog box, or a web page.

More formally, $WS \xrightarrow{A} WT \Leftrightarrow$ one event generated within source window WS (by user or system) enables a transition of type A to target window WT. Fig. 1 depicts the graphical representation of this transition: a window is represented by a window icon, a transition of type A by an arrow labeled with A, and an arrow marker can specify that the user is still able to work with WS, while working with WT; this is a modeless transition. If no such marker exists, the user is only able to work with WT; this is a modal transition.

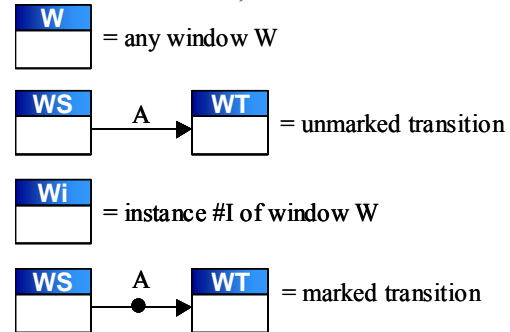


Fig. 1. Notation for a window transition.

A set of window operations is defined in fig. 2. Any operation performed on the source window is depicted by its icon on the left side of the transition arrow; any operation per-

formed on the target window is depicted by its icon on the right side of the transition arrow.





Operation icon	Operation name	Operation definition
	opening operations	open the window = {any state \rightarrow max, title, min, tiling, normal overlap, system overlap, user overlap}
	closing operations	close the window = { any state \rightarrow close }
	Reducing operations	reduces the window = {max \rightarrow tiling, min, tiling, normal/system/ user overlap; tiling \rightarrow min; tiling \rightarrow tiling, min; normal/system/user overlap \rightarrow tiling, min}
	Restoring operations	restores the window = { min \rightarrow tiling, tiling, max, normal/ system/user overlap; tiling \rightarrow tiling, max, normal/system/user overlap; tiling \rightarrow max, normal/ system/user overlap; normal/ system/user overlapping \rightarrow max }

Fig. 2. Notation for generic window operations.

With the above notation, we can investigate transition combinations without suffering from too much representational complexity. When nothing is specified on the left sibling of the arrow, the window remains in its current state, but always gives the focus to the destination window. For example, in fig. 3, when W1 is closed, W2 is opened in any state (it can be refined afterwards according to any corresponding basic operation), but W3 closes W4 and remains in its current state. Similarly, W5 opens W6 in any open state. W7 opens W8 and W9 in their current states. In this last case, only one arrow represents the double opening, but it might be helpful to distinguish them as other window transitions may be specified in the same model.

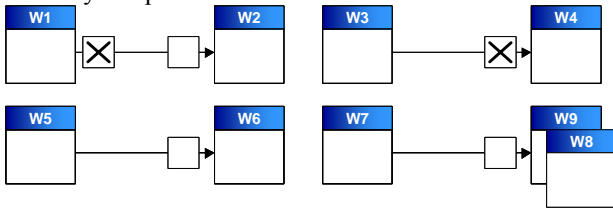


Fig. 3. Some examples of window transitions.

When nothing is specified at the destination of the arrow, it is assumed by default that the window in concern is (re)activated in its previous state. For example, in fig. 4, W1 opens W2 while remaining in its current state, deactivating and giving the focus to W2. When W2 is closed in the second transition, the focus is given back to W1 thanks to the undefined destination. With this combination and the marker, modeless as well as modal transitions can be specified, as depicted in the second part of fig. 4.

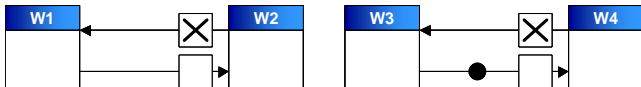


Fig. 4. Some examples of window transitions.

THE CONCURTASKTREE NOTATION

The task models will be denoted using the ConcurTaskTree notation [5]. In this notation, a task is recursively decomposed into subtasks to end-up with leaf task elements. Beside this structural relation, children of a single parent task element can be linked together according to several operators summarized in Table 1.

Notation	Operator	Description
$T1 \gg T2$	Enabling	T1 and T2 are two purely sequential task elements, that is T2 is carried out when T1 is finished.
$T1[] \gg T2$	Enabling with information passing	This operator is similar to the normal enabling operator, but information is passed from T1 to T2
$T1 > T2$	Suspend/resume	T1 can suspend the accomplishment of T2 and resume it when suspended
$T1 [] T2$	Choice	T1 and T2 are two task elements whose accomplishment is mutually exclusive.
$T1 [> T2$	Disabling	The first action executed in T2 disables T1
$T1^*$	Iteration	T1 can be carried out several times
$T1n$	Finite iteration	T1 can be carried out a finite amount of times defined by the value of n .
$T1 T2$	Concurrency	T1 and T2 can be carried out in any order, but the two are needed (sometimes, this operator is referred to interleaving)
$T1 [x] T2$	Synchronization	T1 and T2 are in parts or whole synchronized in their accomplishment, that is some actions are synchronized on some shared domain objects
$[T]$	Optional	Carrying out the task element T is not required
T	Recursion	A same task T can repeat itself in its own definition, thus allowing recursion of task accomplishment

Table 1. Operators of the ConcurTaskTrees notation.

DERIVATION RULES

In this section, we show how two frequently used operators of the notation can initiate a generation of presentation and dialog elements expressed according to the Windows Transition notation.

The enabling operator

Let us assume that T1 is decomposed into two subtasks T2 and T3, which are related by the enabling operator. Fig. 5a depicts the graphical representation of presentation and

dialog elements corresponding to this situation: C1,2 allows the transition from the parent window W1 corresponding to T1 to the first child window W2 corresponding to T2 in the sequence. This element can be named accordingly (e.g., with the name of T2). Similarly, C2,1 cancels the current dialog in W2 and returns to W1; C2,3 allows the sequential transition from W2 to W3 and can therefore be labeled “Next” or “Name of T3”. C3,2 returns to W2 and can be labeled “Previous” or “Name of T2”. C3,1 can be any dialog element for closing W3 to come back to W1, such as “Ok”, “Close”, “Cancel”.

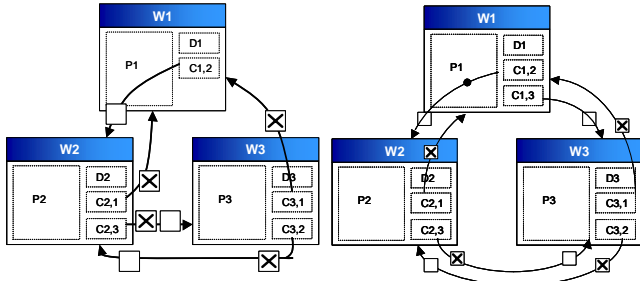


Fig. 5. Alternative specifications for the enabling operator

The enabling operator with information passing

This operator is much more rich than the previous one as it allows the presentation of *I*, the information passed from subtask T2 to subtask T3. Let us denote by *WI* the window materializing *I*, *PI* the presentation of this information and by *DI* the dialog related to this information, if needed. For example, a “Search” push button, or a “Validate” icon. In order to explore the design alternatives that this operator may engender, we introduce the *visibility* property. The information passed between task elements is said *visible* if and only if the user is able to access the information. When no interaction is required to access the information, it is *observable*. When some interaction is required, it is *browsable*. The introduction of this property is motivated by the observation that not all information items can or should be displayed together at one time. Critical information should definitely be observable, but unimportant information should not. Rather, the user should be able to access this information, but only on demand. Browsable information can be described as being *internally* or *externally* browsable, depending on if the information is presented within or outside the scope of the initial window. If externally browsable information must disappear when the user returns to the initiating window, the interaction is said to be *modal*. Otherwise, it is *modeless*. Information simultaneously visible in multiple task elements is said to be *shared*. The designer can specify these parameters either globally for a task or locally by redefinition for particular subtasks. Fig. 6 depicts a decision tree resulting from the examination of different configuration cases induced by visibility from subtasks T2 and T3. Each configuration ID will be referred to in the text. The A region as graphically defined is repeated for (11), (12), (13), and (14).

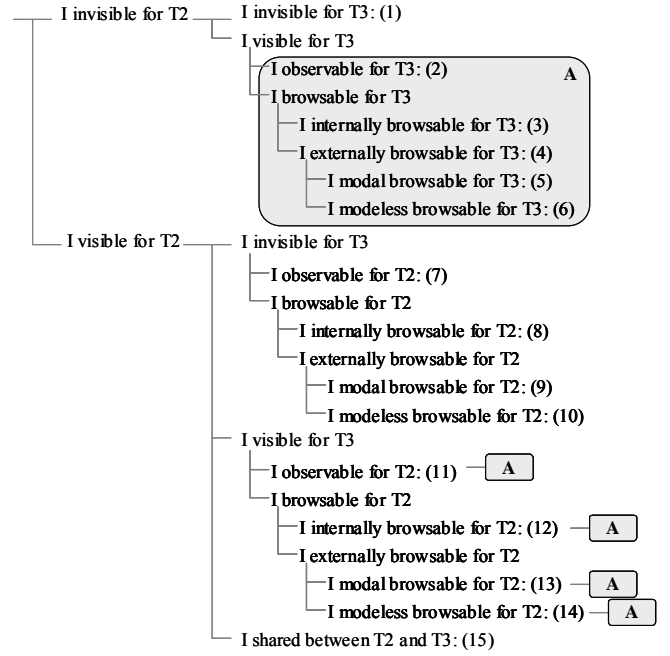


Fig. 6. Decision tree of alternative specifications for the enabling operator with information passing.

Configuration (1) is similar to the simple enabling operator (Fig. 5a). Configuration (2) is similar to that operator too, but W3 is replaced by the window configuration reproduced in Fig. 7a. When ‘*I*’ is not observable for T3, it is considered that ‘*I*’ should be browsable in some way. When *I* is internally browsable (3), Fig. 7b reproduces the resulting expanding window with two dedicated dialog elements: *CI*>> expands the current window to let ‘*I*’ appear (e.g., via a “More>>” button) while *CI*<< reduced the current window to remove *I* (e.g., via a “Less<<” button). When ‘*I*’ is externally browsable (4), two cases between W3 and W1 are possible: the dialog could be modal (5) or modeless (6), as represented with the marker on Fig. 7c.

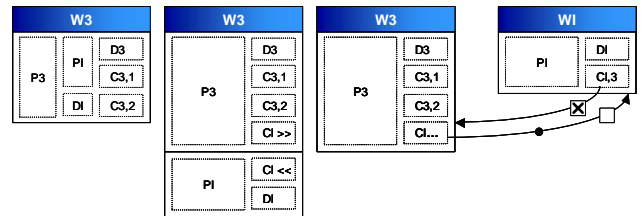


Fig. 7. Configuration changes when *I* is invisible for T2.

When ‘*I*’ is observable for T2 (7), only W2 should be replaced in Fig. 5a by Fig. 8a. When ‘*I*’ is internally observable for T2 (8), W2 becomes structured as represented in Fig. 8b. And, similarly to above, when ‘*I*’ is externally observable for T2, Fig. 8c provides the change for modeless case (10). For modal case (9), the marker is removed.

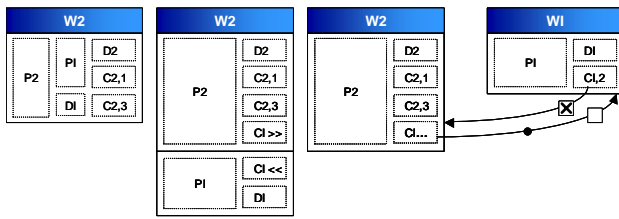


Fig. 8. Configuration changes when I is visible for T2.

The visibility of I for both T2 and T3 raises a potential problem of redundancy and/or consistency. When ‘I’ is observable for T2, the pattern represented by the “A” rectangle on Fig. 6 is duplicated for each sub-case. In these configurations, there is no risk of redundancy of ‘I’ presentation and dialog elements since the sequence between T2 and T3 imposes that W2 and W3 cannot be displayed simultaneously. Therefore, ‘I’ is never displayed two times on the user screen. For instance, Fig. 9a graphically represents the ‘I’ observability in both T2 and T3, which is a possible configuration in (11). Since opening W3 closes W2 and vice versa, ‘I’ is only displayed once.

This reasoning also holds for subsequent categories (12), (13) and (14) except the case where the two instances of W1 are modeless. They should be independent due to the enabling operator between the two subtasks: this is why closing W2 or W3 also closes, respectively, the first or second instance of W1 as shown in Fig. 9b. Fig. 10 graphically depicts the last special configuration (15) where ‘I’ is shared among W2 and W3. In this case, W1 can be created by both W2 and W3, but once. W1 should stay visible until T3 is achieved in W3, not before. Therefore, W2 could be created from W2 and destroyed from W3 (i.e. when leaving W3).

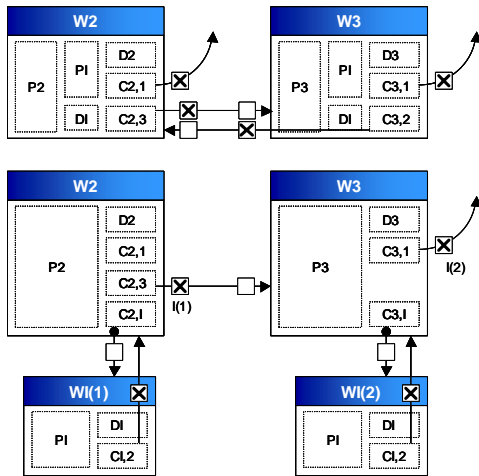


Fig. 9. Configuration changes when I is visible for both T2 and T3 (a,b).

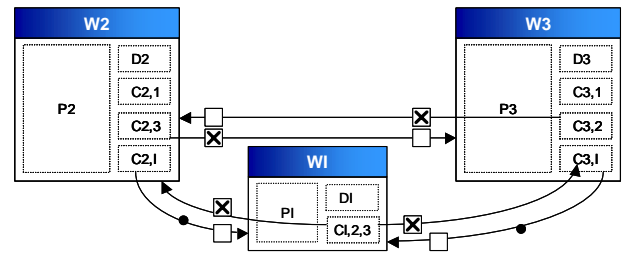


Fig. 10. Configuration when I is shared.

Although all of the configurations gathered in Fig. 6 are theoretically possible, it might be desirable to avoid some configurations for usability reasons. The development of a complete set of guidelines is beyond this paper's scope, but we present a few examples here:

- The invisibility of ‘I’ from both T2 and T3 (1) should be avoided since no user feedback of I is produced towards the user.
- In figure 9b, W2 and W3 can both initiate W1. All transitions in such a situation should be modal, so that the user will not forget which window initiated W1.
- The presentation and dialog should foster information *persistence*, the faculty of I remaining visible is some way when the transition from W2 to W3 is operated.

THE DERIVATION RULES IN ACTION

Fig. 11 depicts the ConcurTaskTrees diagram related to the interactive task “Phone order”, one of the tasks belonging to the general application “Product management.” When a customer makes a phone order to the company, three steps are performed:

1. Identification of the customer

When a customer calls, the operator asks the customer whether she is already a registered customer of the company. If not, the customer provides first name, last name and complete address. If the customer is already registered and still knows her identification number, she tells the operator what this number is. The operator then searches this ID through the company's database. ID registration by phone is often fraught with human error and searching such numbers can be unsuccessful. Three cases may occur:

1. The ID exists in the database. The operator reads the address listed for the ID to the customer, in order to verify that it is the correct ID. If the customer moved, she may provide the new address;
2. The ID exists in the database, but it is incorrect. The operator then asks for the customer's name, trying to locate the customer with this more precise information. The operator may either succeed (in this case, the customer's address is read and could be modified if necessary) or fail (in this case, the operator adds a new record to the database);

- The ID does not exist in the database: similarly, the operator assumes the caller to be a new customer and thus creates a new entry.

If the customer is already registered, but does not know her identification number, the operator asks for the name, to try to locate the customer. Again, the search may either succeed or fail. In all cases, the identification number of the customer is available as soon as the customer is identified.

2. The definition of the order

For each product requested, the customer provides the product number and the quantity to the operator. For each product, the operator dictates the label of the article until each product number and label match.

3. The record of the order

Once the customer has been identified and the order has been defined, the operator asks the customer the mode of payment (which can be cash, by credit card or by bank transfer) and the delivery date. The operator then records the order.

The other tasks of this application are not specified here. The task model consists of three subtasks at the first level. Only the first two are concurrent: the user could begin by identifying a customer or depositing an order, but both subtasks are required to perform the final confirmation.

To identify a customer, three alternatives have been identified in the scenario. Only one of them needs to be fulfilled; there is a global choice between them. Only after identifying the customer is the user able to modify the address of the identified person, if necessary. To deposit an order, the user can order several products (hence, the iteration operator). This task consists of adding a product, specifying its quantity and computing the line total (by computer). A suspend/resume operator is specified to enable users to switch from any of these views to a list of available products. To confirm an order, the user has to perform two operations sequentially: specifying the payment mode and providing a final confirmation. The “Confirm an order” subtask passes information (i.e., the ordered products and the grand total) so that the user will be able to see the contents of the bag before finally deciding.

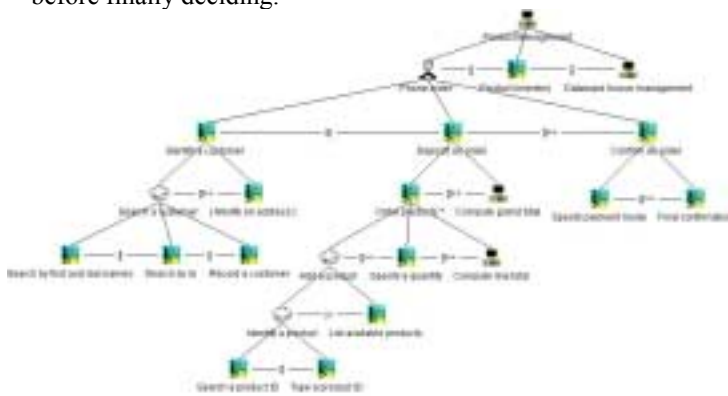


Fig. 11. ConcurTaskTree diagram of the task “Phone order”.

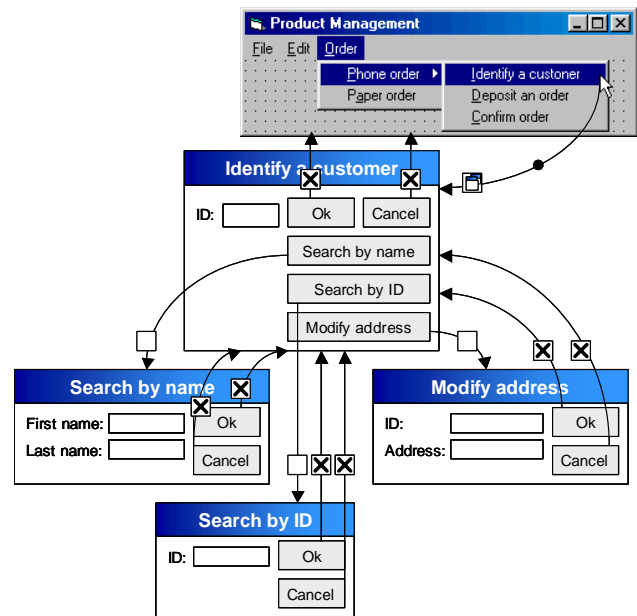


Fig. 12. Derived presentation and dialog for the first subtask.

Fig. 12 shows the presentation and dialog that can be derived from the first subtask of “Phone order”. It is assumed here that the names of these three subtasks lead to the three items of the “Phone order” sub-menu to reflect this decomposition. Since there is a multiple choice between the leaf nodes of the left sub-tree, an implicit choice has been selected and applied. The “Record a customer” leaf node is not presented here to keep the drawing concise, but the dialog is similar. “Modify an address” is conditional to identifying a customer: the “Modify address” push button will therefore be activated the customer has been identified. The optionality of this subtask makes the separate presentation more appropriate. The ID, which is the information passed here, has been considered important to be observable from both source and target windows.

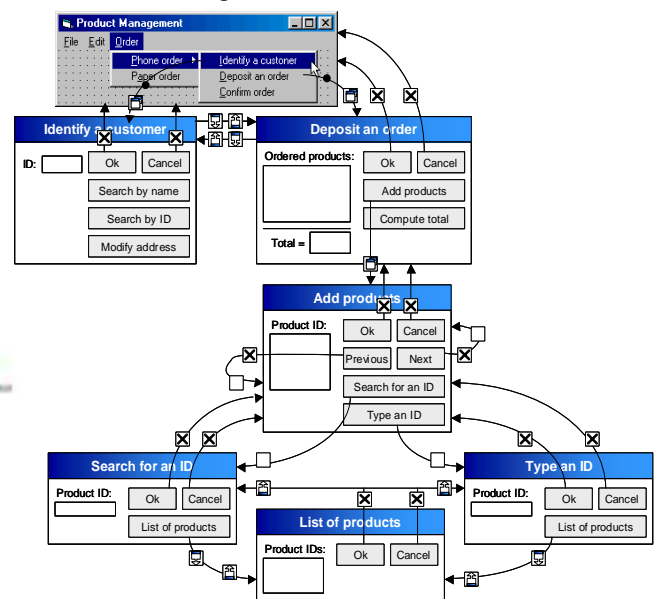


Fig. 13. Derived presentation and dialog for the second subtask.

Fig. 13 shows how we can derive presentation and dialog for the second subtask “Deposit an order.” Note that the first two subtasks of “Phone order” are concurrent; this has been respected in the dialog design. In the window entitled “Deposit an order”, the “Add products” dialog can be visited repeatedly, due to the repetition attribute on “Order products.” The derivation rule for infinite iteration without accumulation has been exploited to produce this window, equipped with “Previous” and “Next” buttons. The “Specify a quantity” and “Compute line total” tasks have been omitted here, but should be part of “Add a product” window as the quantity is an additional information item and the “Compute line total” can be materialized by a push button. The choice derivation rule that has been applied for “Search a product” and “Type a product” is consistent with the one used in the first subtask. More interestingly, the suspend/resume operator that applies to “List available products” specifies that it should be accessible from both “Search a product ID” and “Type a product ID”.

Fig. 14 shows a possible use of derivation rules to derive presentation and dialog structures for the third subtask “Confirm an order”. The information passed between the two children is observable from “Confirm an order” only. The enabling operator with information passing (Fig. 6) therefore produces two sequential windows, one of which is refined as a message dialog box. This is a two-step confirmation: the first by pushing the “Ok” push button on the first window to complete the order with the payment mode and the second by pushing the “Yes” push button on the message dialog box.

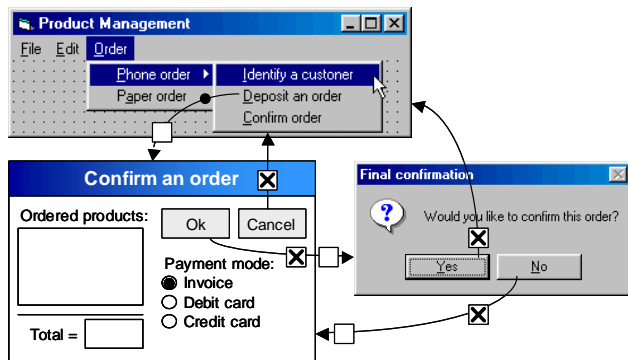


Fig. 14. Derived presentation and dialog for the third subtask.

TOOL SUPPORT

One important design requirement for a software tool that supports the definition and the use of these derivation rules is to ensure predictability of produced results, while maintaining flexibility. We expect novice designers to prefer having a design assistant with more guidance. When they become expert, they may no longer want to refer to the assistant at all times, but only on demand.

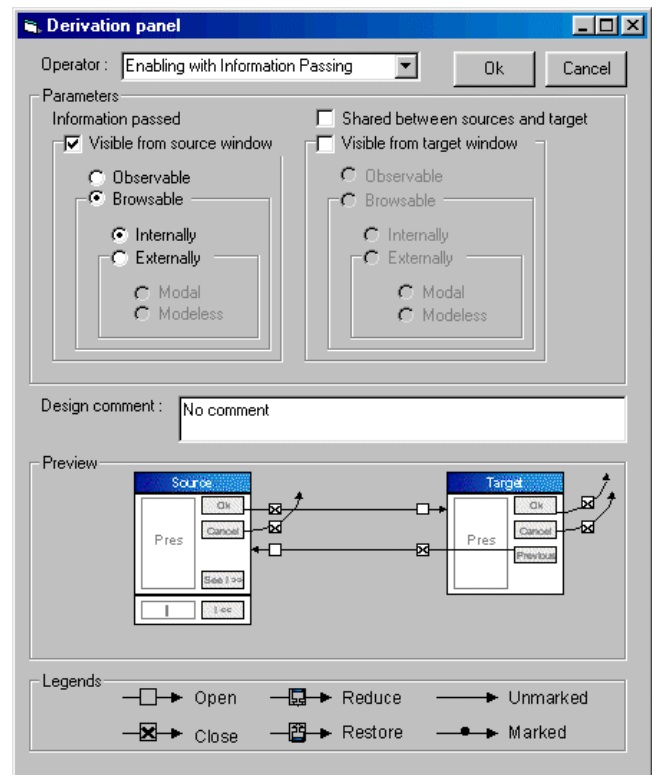


Fig. 15. Guided definition of derivation rules.

Fig. 15 shows a dialog box of a design assistant that could provide designers with some guidance in defining and applying derivation rules: as the designer changes the parameters governing the definition of the rules, a graphical preview represented according to the introduced notation is produced so as to enable designers to understand the relationships between parameters and the final results. For each combination of parameters, a design comment may be produced depending on available design knowledge.

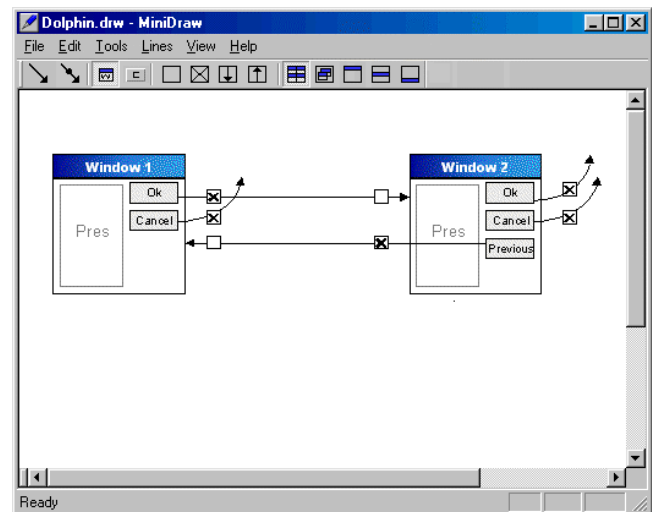


Fig. 16. Free definition of derivation rules.

The dialog box of fig. 15 is therefore intended to maximize predictability, guidance, ease of use, and rapidity. Conversely, fig. 16 shows a mini-editor where more experi-

enced designers could graphically specify the transitions between the windows and the objects holding these transitions. This more open approach is intended to maximize predictability, creativity and flexibility.

A combination of both approaches can be imagined: a designer can start with a first definition based on the initial parameters and then customize it with the graphical editor.

FUTURE WORK

Future work can be imagined along four directions:

1. *Derivation rules*: Under the same working hypotheses that have been assumed here, some other alternative specifications could probably be defined. Generalization of these rules can also be investigated by progressively relaxing the hypotheses that have been assumed one by one. In addition, only the temporal relations have been used here, not the structural relations. Combining both types of relation, as achieved in [14] for example, should be analyzed further.
2. *Usability Assistance*: It would obviously be desirable if the use of derivation rules were governed by usability guidelines. In cases where the derivation rules generate several possible presentation and dialog structures for a single task, we hope to identify the best one through implementation and user testing. Having a language to describe task, presentation and dialog element would also be important to store the specifications in a more formal way, thus allowing multiple functions to be executed, such as model checking, verification of properties.[???] We can even imagine that a predictive model or a technique like action analysis could be automatically applied on the derived UI to compute their predictive accomplishment complexity and time respectively.
3. *Incorporating additional models*: The consideration of domain and/or user models along with the task model to derive presentation and dialog models would probably improve the quality of the results. In addition, it would be interesting to identify just how the knowledge captured in domain and user models affects the derivation of presentation and dialog models. In particular, the influence of a user model on the presentation model should be investigated. For example, one can imagine a system that adapts the presentation to include more wizards and directed interactions when the user is identified as a novice, and more open interactions (such as modeless browsability) when the user is an expert.
4. *Evaluating resulting usability*: We hope to assess the quality of UIs derived from the task model, with the rule parameters as an independent variable. Towards this goal, we should identify the impact of local changing of parameters on the overall usability quality. Up to now, only theoretical argumentation based on respected design guidelines can hold. Experimental studies should be the next step.

ACKNOWLEDGMENTS

Jean Vanderdonckt wishes to thank the Fulbright-Hayes American Commission of Educational Exchange for supporting this work.

REFERENCES

1. Barclay, P.J., Griffiths, T., Mc Kirdy, J., Paton, N.W., Cooper, R., Kennedy, J.: The Teallach Tool : Using Models for Flexible User Interface Design. *Proc. of CADUI'99* (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 139–158.
2. Brown, J., Graham, N., Wright, T., The Vista Environment for the Coevolutionary Design of User Interfaces Supporting the Design Process, in *Proc. of CHI'98*, ACM Press, 376-383.
3. Dittmar, A. and Forbrig, P., Methodological and Tool Support for a Task-Oriented Development of Interactive Systems, in *Proc. of CADUI'99* (Louvain-la-Neuve, October 1999), Kluwer Academics, 271-274.
4. Eisenstein, J. and Puerta, A., Adaptation in Automated User-Interface Design, in *Proc. of IUI'2000* (New Orleans LA, January 2000), ACM Press, 74-81.
5. Paternò, F., Mancini, C., Meniconi, S., ConcurTask-Trees: A Diagrammatic Notation for Specifying Task Models, in *Proc. of Interact '97* (Sydney, July 1997). Chapman & Hall, 362-369.
6. Paternò, F., Breedvelt-Shounet, I., de Koning, N.: Deriving Presentations from Task Models. In *Proceedings of IFIP Workshop on Engineering the Human-Computer Interaction EHCI'98* (Crete, September 1998). Kluwer Academics Publisher, Dordrecht (1998)
7. Puerta, A.R. and Eisenstein, J., Towards a General Computational Framework for Model-Based Interface Development Systems, in *Proc. of IUI'99* (Los Angeles CA, January 1999), ACM Press, 171-178.
8. Puerta, A.R., The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development, in *Proc. of CADUI'96* (Namur, June 1996), Presses Universitaires de Namur, Namur, 19-36.
9. Schlungbaum, E., Individual User Interfaces and Model-based User Interface Software Tools, in *Proc. of IUI'97*, ACM Press, 229-232.
10. Stirewalt, K.R.E.: MDL: a Language for Binding UI Models. In *Proc. of CADUI'99* (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 159–170
11. Szekely, P., Retrospective and Challenges for Model-Based Interface Development, in *Proc. of CADUI'96* (Namur, June 1996), Presses Universitaires de Namur, xxi-xliv.
12. Tam, R.C., Maulsby, D., Puerta, A.R., U-TEL: A Tool for Eliciting User Task Models from Domain Experts, in *Proc. of IUI'98* (San Francisco, January 1998), ACM Press, 77-80.
13. Vanderdonckt, J. and Bodart, F., Encapsulating Knowledge for Intelligent Interaction Objects Selection, in *Proc. of InterCHI'93* (Amsterdam, April 1993), Addison-Wesley, 424-429.

14. Wilson, S. and Johnson, P., Empowering Users in a Task-Based Approach to Design Experience and Requirements, in *Proc. of DIS'95*, ACM Press, 25-31.