Human-Computer Interaction -- INTERACT'03
M. Rauterberg et al. (Eds.)
Published by IOS Press, (c) IFIP, 2003, pp. 455-462

Deriving the Navigational Structure of a User Interface

Jean Vanderdonckt, Quentin Limbourg, Murielle Florins

Université catholique de Louvain, Louvain-la-Neuve, Belgium
{vanderdonckt, limbourg, florins}@isys.ucl.ac.be

Abstract: Model-based approaches to user interface (UI) design typically focus on creating mappings between concrete features of the user interface design and abstract models of the tasks, users, and domain. Earlier research in this area has focused on widget selection, since this was considered to be the least complicated mapping to investigate. We take this work a step further, showing how the navigational structure of the user interface should be influenced by features of the task model. This relationship between the task model and the navigational structure of the UI is expressed as a set of formal rules, enabling automated design assistance and evaluation. A user interface design assistant (UIDA) is used to organise design alternatives based on formal notations for describing both the task and dialog structures and to provide designers with assistance on this.

Keywords: Model-based user interface design, design automation, design assistance, user interface assistant.

1 Introduction

Model-based user interface (UI) design involves the creation of formal, declarative models that describe both the look and feel of the UI, and the tasks, domain, and users that it is intended to support (Johnson, 1992). A typical development life cycle begins with user-task elicitation (Tam *et al.*, 1998), when models of the task, domain, and users are obtained. After that, the domain model is exploited to produce a data model for database design and application modelling. Task modelling is today recognised as a useful starting point for the generation of the other models because it helps to ensure user-centred design (Johnson *et al.*, 1995). After these initial, abstract models are elicited, three mechanisms are used to refine them and to produce more concrete models of the UI design:

- *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 (Paternò *et al.*, 1999).
- *Model linking/binding*: one or many already specified models are processed to establish relationships between the elements in the various models available (Griffiths *et al.*, 2001).
- *Model composition*: models which are already specified are partially or totally assembled either to rebuild the source models or to build another model (e.g., in reverse engineering) related to the same real world (Stirewalt, 1999).

There are at least three key **abstract models**. The *task model* describes the goals that the user hopes to accomplish, and the actions that must be taken to accomplish them. The *domain model* describes the objects and data that the user will be concerned with. The *user model* describes properties of the users themselves, such as their level of expertise, or their security clearance model.

There are at least two key **concrete models** as they represent something that straightforwardly. The *presentation model* describes the visual appearance of the user interface. It specifies which widgets have been selected, and where they are placed, among other things. The *dialog model* describes the mechanics of how the user is to interact with the UI. It specifies the navigational structure of the UI, and the used interaction techniques.

Model-based user-interface design focuses on finding mappings between these various models. For example, when researchers showed how to perform interactor selection based on features of the application domain, they were attempting to create a domain-presentation mapping (Puerta & Eisenstein, 1999). In this paper, we are interested by one particularly important combination of mappings: task-presentation-dialog. We want to show how decisions about the UI look and feel relate back to features of the task model.

The rest of this paper is structured as follows: the following section reviews some related work. Then we define the notations that we will use in this

paper: the Window Transitions notation for representing features of the presentation-dialog model and the ConcurTaskTrees notation (Paternò *et al.*, 1997) for representing the task model. We will then present and explain in depth two representative examples of rules for deriving features of the presentation and dialog models from the task model. The following section provides an example of such a derivation, and includes a discussion of how such these derivation rules can be supported by a software tool. Finally, identify some directions for future work are identified and justified.

2 Related Work

To drive user interface design, there are several reasons to start with the task model:

- The task model centers the design process around the user's own experiences and goals, rather than the limitations and affordances of system design (Johnson *et al.*, 1995).
- Task models are well-understood, and can be very expressive (Johnson *et al.*, 1999).
- Data models or domain models tend to focus on generic tasks (e.g., insert, delete, modify, list, check, search, print, transfer), which may be difficult to relate to the user's idea of what tasks should be accomplished.
- Finally, derivation of other models from a task model is also allowed to come closer to the designer's world, but there is a potential risk of information loss in the transformation as they are progressively reduced. Design decisions are more concrete than the abstract information contained in task models.

There are two types of relations by which task models can be organised. *Structural relations* recursively decompose complex tasks into simpler subtasks, end with atomic actions on domain objects. *Temporal relations* provide constraints for ordering (sub-)tasks according to the task logic.

MOBI-D focuses on structural relations between task elements. It includes a design assistant that allows the designer specify a preference ranging from a single window in which all subtasks can be achieved all the way to the opposite end of spectrum, where each atomic subtask gets its own window (Tam *et al.*, 1999). SEGUIA also bases its grouping algorithm on structural relations (Vanderdonckt & Berquin, 2001): a presentation unit is selected for each subtask of the first level, and each presentation unit is consequently decomposed into windows to be grouped according to various alternative strategies as long as a threshold for some cognitive load metric is not exceeded. Paternò *et al.* (1999) employ a bottom-up approach that exploits the structural relations first, and then the temporal relations to group presentation elements based on the concept of an activation set. Such a set groups items corresponding to subtasks that can be carried out in some related way. TADEUS also generates both a presentation and a dialog model from a task model, according to predefined correspondence tables (Schlungbaum & Elwert, 1996). Our work differs from these approaches in that it systematically considers a larger set of temporal operators, it derives multiple navigational structures from these operators thanks to a decision tree, and that it provides a graphical notation to represent it and a software support. UI patterns can be associated with repetitive task model configurations.

3 Windows Transition Notation

We define a *window transition* is any change in the set of windows that are visible; for example, a previously minimised window might become maximised, or the front window might be moved to the background. The window in which the transition was initialised is called the *source window* (WS); the window that is affected by the transition is the *target window* (WT). A "window" means any container composed of widgets that enable the user to perform tasks, such as a dialog box, a notebook, a tabbed dialog box, or a web page.

Schematically, WS $\xrightarrow{A}$ WS $\Leftrightarrow$ an event generated within source window WS (by the user or the system) causes a transition of type A to target window WT. Fig. 1 depicts the graphical representation of this transition: each window is represented by a window icon, and a transition of type A is shown by an arrow labeled with "A." An arrow marker specifies that the user is still able to work with WS after the transition. If no such marker exists, the user is only able to work with WT.

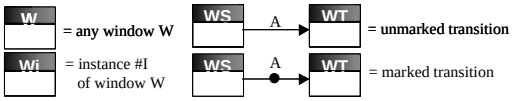


Figure 1: Notation for a window transition.

Fig. 2 specifies any operation to be performed on the source and target windows when a window transition is initiated. Any operation performed on the source window, respectively the target window, is depicted by an icon on the left side, respectively the right side of the transition arrow.

Icon	Name	Operation definition
	Opening operations	open the window ={any state → max, title, min, tiling, normal overlap, system overlap, user overlap}
	Closing operations	close the window = { any state → close }
	Reducing operations	reduces the window = {max→ titling, min, tiling, normal/system/user overlap; titling → min; tiling → titling, min; normal/ system/user overlap → titling, min}
	Restoring operations	Restores the window = {min → titling, tiling, max, normal/ system/user overlap; titling → tiling, max, normal/system/user overlap; tiling → max, normal/ system/user overlap; normal/ system/user overlapping → max}

Figure 2: Notation for generic window operations.

This graphical notation allows us to investigate combinations of transitions graphically. We will assume that, when no icon is specified on the left side of the arrow, the window remains in its current status, but simply gives the focus to the destination window. In Fig. 3, the upper-left transition closes W1, and opens W2. The upper-right transition closes W4 and keeps W3 open. The lower-left transition opens W6 and gives it the focus. The lower-right transition opens both W8 and W9, and keeps W7 open. 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.

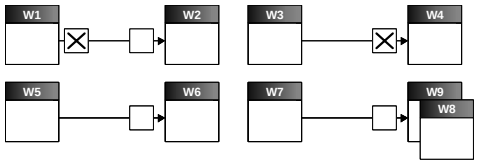


Figure 3: Some examples of window transitions.

When no icon is specified at the destination of the arrow, we assume that the target window retains its previous status, and is given the focus. For example, in Fig. 4, W1 opens W2, which is then successively deactivated, giving the focus to W1, in its initial state. By combining these transitions with the marker, we can specify *modal dialogs*, as depicted in the second part of Fig. 4.

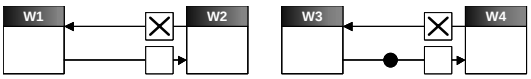


Figure 4: Some examples of window transitions.

4 The Task Notation

The ConcurTaskTree notation is used to represent the task model (Paterno *et al.*, 1997). This notation includes both structural and temporal relations. Tasks maybe decomposed into subtasks, and tasks that are all children of a single task can be linked together according to several temporal operators.

4.1 Derivation Rules

Having defined formal notations for describing the task model and the navigational structure of the UI, we will now describe two example rules for deriving the navigational structure from temporal and structural properties of the task model.

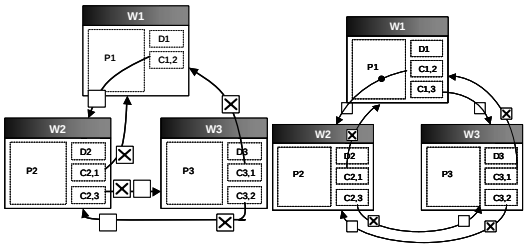


Figure 5. Alternative specifications for Enabling op.

Enabling operator. When one task enables another, the corresponding window transitions are fairly obvious. Let's assume that task 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. 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 labelled "Next" or "Name of T3". C3,2 returns to W2 and can be labelled "Previous" or "Name of T2". C3,1 can be any dialog element for closing W3 to come back to W1 as "Ok", "Close", "Cancel".

Enabling operator with information passing. When information needs to be passed between subtasks, the situation grows much more complex. The information itself will be represented by *I*; *WI* represents the window materialising I, *PI* the presentation of this information and by *DI* the dialog structure involved in manipulating this information. For instance, it could be a "Search" push button, a "Validate" icon to check the data consistency. There are several properties of the information itself that are relevant to the choice of windows transitions here. The information passed between task

elements is said *visible* iff the user is able to access the information. When no interaction is required to access the information, it is said to be *observable*. When the user must perform some interaction in order to access the information, then it is said to be *browsable*. These properties are motivated by the observation that not all information items should be displayed all together at a time. Critical information should certainly be observable, but unimportant information should not. Rather, the user should be able to access this information, but only on demand. Information may be either *internally* or *externally* browsable, depending on whether the information is presented within or outside the scope of the initial window. If the externally browsable information must be closed before returning to the initial window, then the dialog that presents the information is *modal*. If the information can remain visible, then the dialog is *modeless*. Information simultaneously visible in multiple task elements is *shared*. These parameters can be specified either globally for a particular task or locally for particular sub-tasks.

Fig. 6 depicts a decision tree of the various combinations of properties of the information passed between subtasks T2 and T3. Each configuration ID will be referred to in the text. The region labeled "A" is repeated for (11), (12), (13), and (14). Configuration (1), where the information is not visible, is identical to the simple enabling operator (Fig. 5a). Configuration (2) is similar to that operator too, but W3 is replaced by the window configuration shown in Fig. 7a.

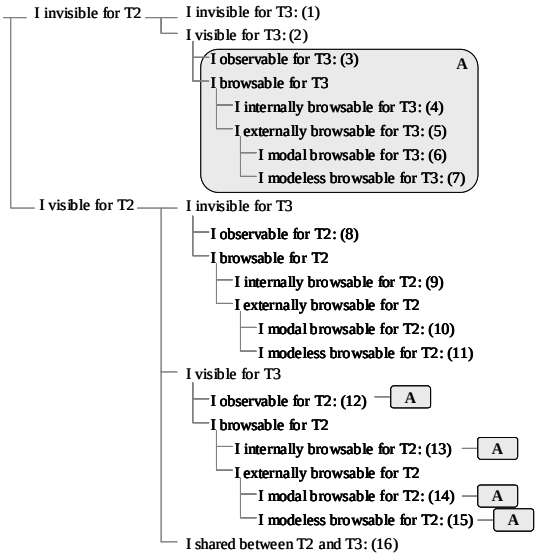


Figure 6: Decision tree of alternative specifications.

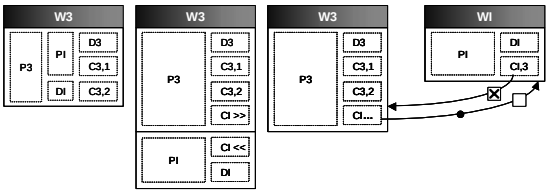


Figure 7: Particular changes when I is invisible for T2.

When I is internally browsable (4), the ideal set of transitions is shown in Fig. 7b. In this case, there is an 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 (5), two cases between W3 and WI are possible: the dialog could be modal (6) or modeless (7) as represented with the marker on Fig. 7c. When 'I' is observable for T2 (8), then W2 should be replaced in Fig. 5a by Fig. 8a. When 'I' is internally observable for T2 (9), W2 becomes structured as represented in Fig. 8b. When 'I' is externally observable for T2, Fig. 8c provides the change for modeless case (11). For modal case (10), the marker is removed.

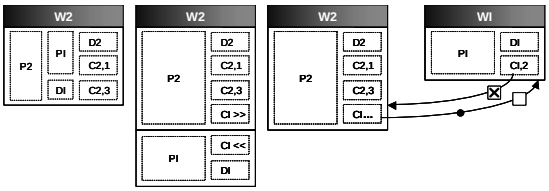


Figure 8: Particular changes when I is visible for T2.

The visibility of 'I' for both T2 and T3 raises a potential problem of redundancy. When 'I' is observable for T2, the pattern represented by the "A" rectangle in Figure 6 is duplicated for each subcase. In these cases, there is no risk of redundancy, since the sequence between T2 and T3 requires 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 (12). Since opening W3 closes W2 and vice versa, 'I' needs to be displayed only once, depending on the time the user wants to access information in 'I'.

This reasoning also holds for subsequent categories (13), (14) and (15), except in the case where the two instances of WI are modeless. These windows should be independent due to the enabling operator between the two subtasks: this is why closing W2 closes only the first instance of WI, and closing W3 closes only the second instance, as shown in

Fig. 9b. Fig. 10 graphically depicts the last special configuration (16) where 'I' is shared among W2 and W3. In this case, WI can be created by both W2 and W3, but only once. WI should stay visible until T3 is achieved in W3. Therefore, WI could be created from W2 and destroyed from W3 (i.e. when leaving W3). Usability considerations render some of the configurations of Fig. 6 undesirable, such as:

- The invisibility of 'I' from both T2 and T3 (1) should obviously be avoided, since the information is never visible to the user.
- Modeless browsability of 'I' in T2 and/or T3 should be preferred to modal browsability, so that the user does not forget the information after closing the modal dialog.
- 'I' should be visible in W3 to achieve T3.
- WI, as every window, should be controlled by one dialog element at a time, but several dialog elements concurrently (e.g., different widgets).
- In order to foster information *persistence*, the I should remain visible is some way during the transition from W2 to W3.

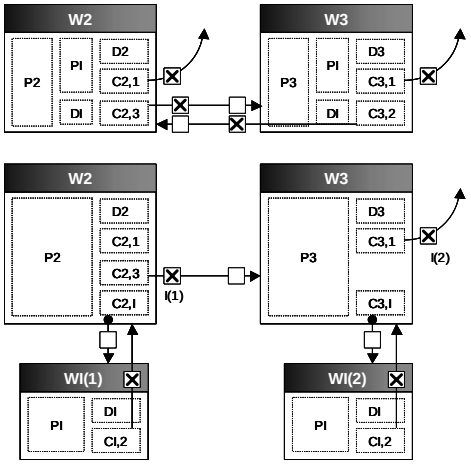


Figure 9: Changes when I is visible for T2 and T3 (a,b).

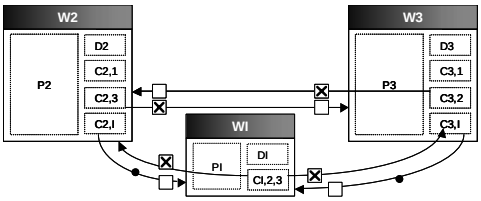


Figure 10: Change when I is shared.

4.2 Expansion of Dialog Actions

The current definition of the dialog model based on windows transition can be augmented by other icons representing dialog actions performed on the windows. In particular, the activation or the deactivation of any widget or container consequent to a dialog action can be also specified similarly. Fig. 11 reproduces icons representing respectively a default action (e.g., a default push button), an active action (e.g., a possible direct manipulation), a deactivated action), an active and an inactive container. DEMAIS (Bailey *et al.*, 2001) uses a similar technique, but for multimedia presentations. Fig. 12 provides icons that represent operations that are allowed to be performed on windows that may consequently lead to another action (e.g., triggering a semantic function, displaying another window). The two last icons of Fig. 12 specify that a window can be resized in any direction or not at all.

Figure 11: Icons for activation and deactivation.

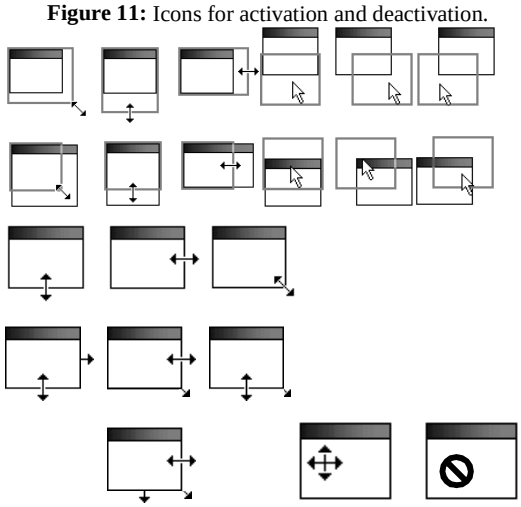


Figure 12: Allowed and prohibited window operations.

4.3 Example

Fig. 13 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, 3 steps are performed:

1. Identification of the customer. When a customer calls, the operator asks the customer whether s/he is already a registered customer of the company. If the caller is a new customer, she provides her first name, last name and complete address. If the caller is an old customer 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 can be unsuccessful. Three cases may occur, but in all cases, the identification number of the customer is available as soon as the customer is identified.

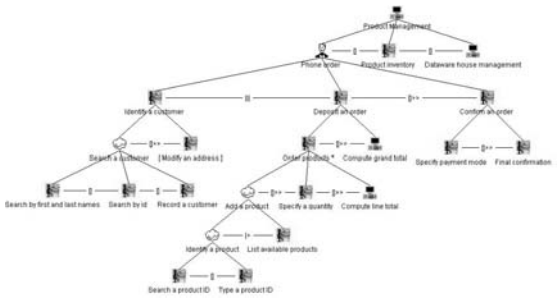


Figure 13: ConcurTaskTree task model "Phone order".

1. The ID is provided correctly, and does exists in the database. The operator then confirms the customer's address: if the customer moved, s/he is now able to provide the new address.
2. The ID is provided incorrectly, or the caller does not know her ID. The operator can then try to search the database using the customer's name instead. The operator may either succeed (in this case, the customer's address is confirmed and possibly modified) or fail (in this case, the operator assumes that it is a new customer and adds a new record to the database);
3. The provided ID does not exist in the database. Again, the operator assumes that the caller is a new customer and thus creates a new entry.

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 to confirm that the label and product number match.

3. The record of the order. Once the customer is identified and the order is 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.

We have three subtasks at the top level, and the first two are concurrent, whereas the third is sequential. The operator could begin by identifying a customer or depositing an order, but these two subtasks are must both be accomplished before the final confirmation can be performed. To identify a customer, three alternatives have been identified in the scenario. As only one of them needs to be fulfilled, there is a global choice between them. After the customer is identified, the operator can modify the address of the identified person, if necessary (optional subtask). To deposit an order, the user can order several products (hence, the iteration operator). This task consists of adding a product (whatever the source is), 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 sequentially two operations: specifying the payment mode and providing a final confirmation. The "Confirm an order" 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.

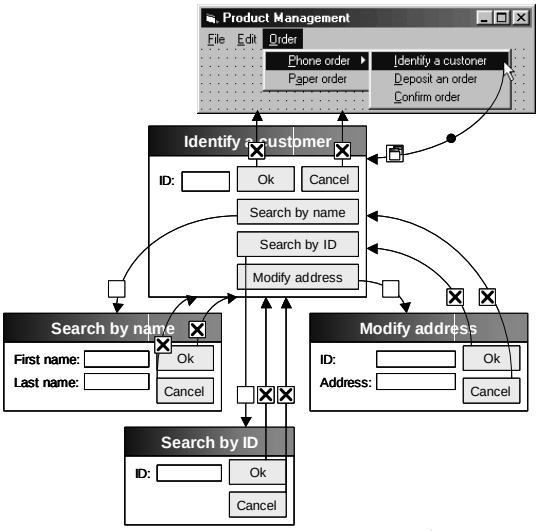


Figure 14: Derived presentation, dialog for 1st sub-task.

Fig. 14 shows a possible use of derivation rules to derive window transitions for the first subtask of "Phone order". Since there is a choice between the leaf nodes of the left subtree, an implicit choice has been selected and applied, through the use of a menu. The "Record of a customer" leaf node is not presented here to keep the drawing concise, but the dialog is similar. "Modify an address" can only be achieved subsequent to identifying a customer: the "Modify address" push button will therefore be activated after the customer is identified. The optional character of this subtask makes the separate presentation more appropriate. The ID, which is the information passed here, is observable from both source and target windows.

Fig. 15 shows a possible use of our derivation rules to produce the window transitions for the second subtask, "Deposit an order". The upper left section of the figure shows the result of the concurrency operator between the first two subtasks of "Phone order". The window entitled "Deposit an order" includes the "Add products" button, which can be clicked 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. "Specify a quantity" has been omitted here, but should be part of "Add a product" window as the quantity is an additional information item. "Compute line total" can be represented by a simple push button. The 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 having "List available products" as right sibling specifies that it should be accessible from both "Search a product ID" and "Type a product ID".

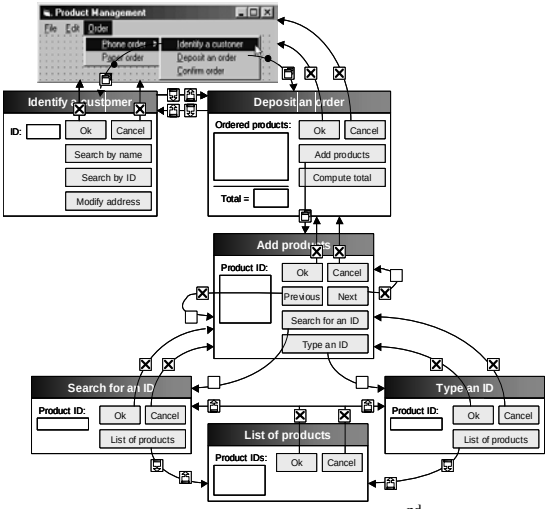


Fig. 15: Derived presentation, dialog for 2nd sub-task.

Fig. 16 shows a possible use of derivation rules to derive presentation and dialog for the third subtask, "Confirm an order". The information passed between the two children is decided to be 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.

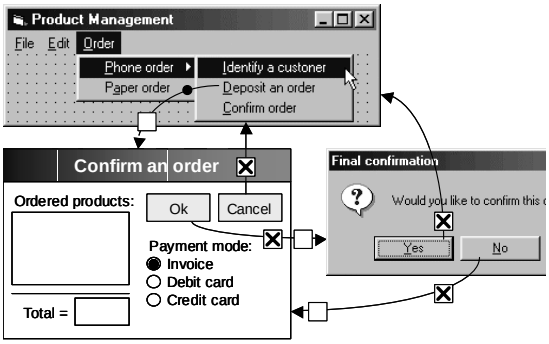
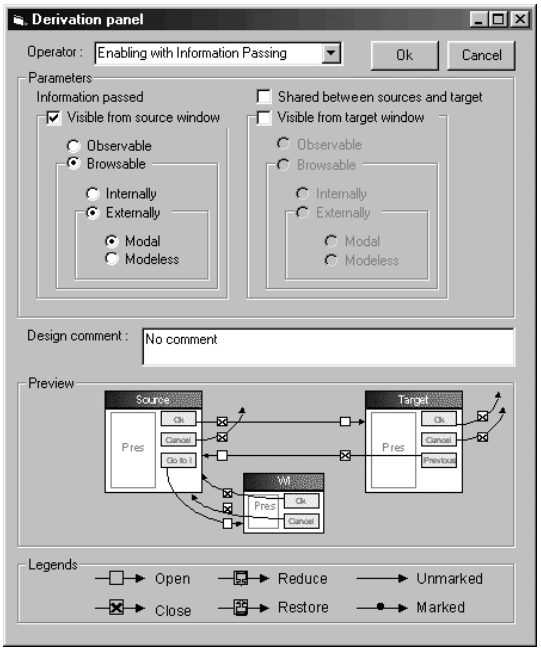


Figure 16: Derived presentation, dialog for 3rd sub-task.

5 TOOL SUPPORT

One important design requirement for a software tool that supports the definition and the use of these derivations rules is to ensure predictability of produced results, while maintaining flexibility regarding to designers. Novice designers tend to prefer having a design assistant with more guidance up to the point their acquired experience will suffice. When they become expert, they no longer want to refer to the assistant systematically, but only on demand. Figure 17 shows a UI design assistant (UIDA) that provides designers with 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 is produced depending on available design knowledge. Fig. 17 maximises predictability, guidance, ease of use, and rapidity. Fig. 18 shows a mini-editor where more experienced designers would be able to graphically specify the transitions between the windows and the objects holding these transitions, thus intended to maximise predictability, creativity and flexibility. A combination of both approaches can be imagined: a design can start with a first definition based on the initial parameters and refine it graphically to take into account any customisation, personalisation, or adaptation.



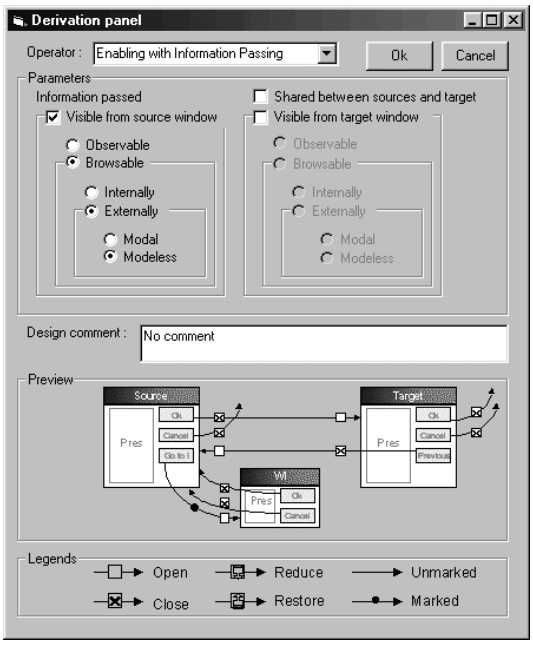


Figure 17: Guided definition of derivation rules.

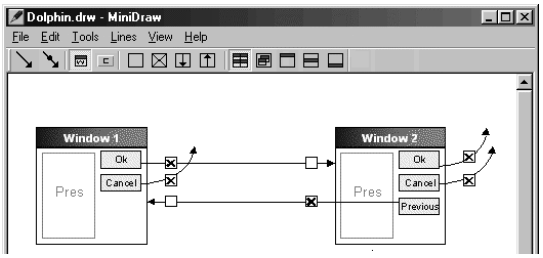


Figure 18: Free definition of derivation rules.

CONCLUSION

Future work can be imagined along four axes:

1. Dimension of derivation rules: under the same working hypotheses, some other alternative specifications could probably be defined. Nut it is hard to assess the point to which no such new alternatives exist: if one has the sake of completeness, a definition of alternative specifications according to a state space search should be defined to restrict the scope of derivation rules. Generalisation of these rules should also be investigated by progressively relaxing hypotheses to verify how each rule is general.

2. Dimension of usability assistance on derivation rules: it is definitely crucial to introduce guidelines to select appropriate use of derivation rules. An important piece of research should implement the different UIs (with their presentation and dialog completed) that can be derived via the derivation rules for a same task and conduct user testing of these different UIs to identify their usability quality, which is not done yet. Only by validating the rules by user testing could guidelines be provided as a confident rule based on experimental validation. Furthermore, as we can more or less predict the quality of any derived UIs, the designer may be better informed about the potential consequences of applying any derivation rule with different values of their parameters.

3. Dimension of model consideration: the consideration of domain and/or user models along with the task model to derive presentation and dialog models would certainly improve the quality of the results, yet a certain integration remains to be achieved. It could be worthwhile to see how knowledge captured in domain and user models affect the derivation of presentation and dialog models. A user model may influence the presentation model.

4. Dimension of resulting usability: it is also critical to assess the quality of UIs derived from the task model depending on the rules parameters. An extensive experiment towards this goal should be able to identify the impact of local changing of parameters on the overall usability quality. Up to now, only a theoretical argumentation based on respected design guidelines can hold. Experimental studies should be the next step.

References

Bailey, B.P., Konstan, J.A. & Carlis, J.V. (2001), DEMAIS: Designing Multimedia Applications with Interactive Storyboards, *Proc. of. Multimedia'2001* (Ottawa, 30 September-5 October 2001), ACM Press, New York, pp. 241-250.

Brown, J., Graham, N. & Wright, T. (1998), The Vista Environment for the Coevolutinary Design of User Interfaces Supporting the Design Process, *Proc. of CHI'98*, pp. 376-383.

Eisenstein, J. & Puerta, A. (2000), Adaptation in Automated User-Interface Design, *Proc. of IUI'2000,*, pp. 74-81.

Griffiths, T., Barclay, P., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C. & da Silva, P. (2001), Teallach: a Model-based UI Development Environment for Object Databases, *Interacting with Computers*, 14(1), 31-68.

Johnson, P. (1992), Human-Computer Interaction: Psychology, Task Analysis and Software Engineering, McGraw-Hill.

Johnson, P., Johnson, H., Wilson, S. (1995), Rapid Prototyping of User Interfaces Driven by Task Models, in *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Carroll (ed.), John Wiley & Sons, pp. 209-246.

Johnson, P. (1999), Tasks and Situations: Considerations for Models and Design Principles in Human Computer Interaction, *Proc. of HCI'Interantional 1999*, pp. 1199-1204

Paternò, F., Mancini, C. & Meniconi, S. (1997), ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, *Proc. of INTERACT'97*, pp. 362-369.

Paternò, F., Breedvelt-Shounet, I., de Koning, N. (1998), Deriving Presentations from Task Models, *Proc. of EHCI'98* (Creete, 14-18 September 1998), Kluwer Acad. Pub.

Puerta, A.R., Eisenstein, J. (1999), Towards a General Computational Framework for Model-Based Interface Development Systems, *Proc. of IUI'99*, pp. 171-178.

Schlungbaum, E. & Elwert, T. (1996), Dialogue Graphs - a Formal and Visual Specification Technique for Dialogue Modelling, *Proc. of FAHCI'96* (Sheffield, September 1996).

Stirewalt, K.R.E. (1999), MDL: a Language for Binding UI Models, *Proc. of CADUI'99* (Louvain-la-Neuve, 21-23 October 1999), Kluwer Academics, Dordrecht, pp. 159-170.

Tam, R.C., Maulsby, D. & Puerta, A.R. (1998), U-TEL: A Tool for Eliciting User Task Models from Domain Experts, *Proc. of IUI'98* (San Francisco, 6-9 January 1998), pp. 77-80.

Vanderdonckt, J. & Berquin, P. (1999), Towards a Very Large Model-based Approach for UI Development, *Proc. of UIDIS'99* (Edinburg, 5-6 September 1999), pp. 76-85.