The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results

Quentin Limbourg, Jean Vanderdonckt, and Nathalie Souchon

[1] Université catholique de Louvain, Institut d'Administration et de Gestion
Place des Doyens, 1 - B-1348 Louvain-la-Neuve, Belgium
{limbourg,vanderdonckt,amisi}@qant.ucl.ac.be

Abstract. Model-based interface development environments typically involve the manipulation and the organization of multiple component models in order to develop one or many user interfaces corresponding to the models. In this context, the mapping problem poses the challenge of defining the elements of these models and to link them into an appropriate way that reflects design options. The task-dialog and task-presentation mapping problem investigates to what extent a dialog model and a presentation model can be derived from a task model and subsequently linked from it. Some preliminary results for solving this type of mapping problem at a high level of abstraction are provided and exemplified on some particular model representations. A task model for a single-user interactive application is formed with the ConcurTaskTrees notation. From the definition of its operators, a set of systematic rules for simultaneously deriving and linking elements for both dialog and presentation models are presented. These rules are organized into a decision tree presenting designers with design alternatives. To depict the models' elements that can be derived and linked from these rules, the Windows Transition graphical notation is exploited.

1 Introduction

Model-based approaches for user interface development [5,8,14] consist in capturing design knowledge in various models according to a declarative manner that can be further processed by software tools. These models are progressively enriched and exploited to finally obtain a running user interface (UI). Concepts typically abstracted in these models, but not necessarily limited to, are user tasks, domain elements, users, presentation items, and dialog structures [10].

In practice, these modeling activities are rarely achieved one after another because the design activity is intrinsically an open, iterative, and ill-defined problem. This means that designers do not necessarily complete the models that are required to perform the next step in the development life cycle. Rather, they like to early investigate what the other models could look like without waiting that the other one are stabilized, complete, and consistent. Moreover, these models are sometimes built by different persons, at different times, thus potentially leading to independent models that could evolve separately. When such models need to be related one to each other again, several relationships need to be established, updated or even restored, which is a tedious and repetitive task [4]. We hereby define *the mapping problem* [10] as the problem of handling relationships between all involved models so as to support the

P. Palanque and F. Paternò (Eds.): DSV-IS 2000, LNCS 1946, pp. 227−246, 2001.
© Springer-Verlag Berlin Heidelberg 2001

development of a user interface. To keep relationships between these models as they are built when they are evolving, three mechanisms are supplied:

1. *Model derivation*: the elements and the relationships of one or many still unspecified models are derived from the elements and relationships of one or many already specified models according to transformation rules, the parameters of which are controlled by the designer [1,7,12].
2. *Model linking/binding*: the elements and the relationships of one or many already specified models are processed to establish, update or restore previously established relationships (one or many of them) between the models elements [1,3,5].
3. *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 another one [11].

In this paper, we are interested by one type of mapping: to what extent is it possible to solve the mapping problem from a task model onto both a presentation model and a dialog model? The task model is chosen as source model because such a rich model can be effectively exploited to derive elements belonging in other models. Second, presentation and dialog models are chosen as target models because they often work hand in hand and because they are largely intertwined (Fig. 1).

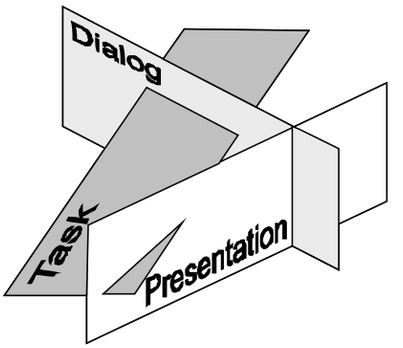


Fig. 1. Intertwining of task, presentation and dialog models.

A *task model* is here understood as a declarative description of an interactive task to be carried out by a user or a population of users through a particular user interface. A task model typically contains two types of internal relationships: *structural relationships* according which a task is recursively decomposed into sub-tasks to end up with actions working on domain objects, and *temporal relationships* that provide constraints for ordering (sub-)tasks according to the task logic.

A *presentation model* is here understood as a declarative description of the external representation of a user interface according to multiple channels of use. A presentation model also involves structural relationships (e.g., how a window container is decomposed into its sub-elements) and arranging relationships (e.g., how two elements are aligned on the users's screen).

A *dialog model* is here understood as a declarative descriptive of how the final user can interact with the presentation through the various interaction devices. It usually contains some specification of the input that the user interface enables and of the output that the system may convey through the user interface.

The *task-presentation mapping* [10] is a type of mapping problem where elements and relationships of a presentation model can be mapped according to those existing in the task model, preferably reflecting how it is organized. Similarly, the *dialog-presentation mapping* [10] is a type of mapping problem where elements and relationships of a dialog model can be mapped according to those existing in the task model, preferably reflecting how it is organized. The goal of this paper is finally to explore how these two types of mapping problems can be addressed in terms of model elements and relationships.

The rest of this paper is structured as follows: Sect. 2 reviews some related work for explaining why starting from the task model is recommended and what kind of strategies have been developed so far for model derivation from a task model. Sect. 3 clarifies the hypotheses assumed in this work and their scope. In particular, notations used for the different models are specified: the ConcurTaskTrees notation [6,9] for task model and the Window Transitions notation for mid-level dialog model. The representation for the presentation model is left open and informal. The different subsections of Sect. 3 explain derivation rules for each ConcurTaskTrees operator and express the results of this derivation according to the Window Transition notation. Sect. 4 provides an example of such derivation and Sect. 5 concludes by reporting on some open issues and future work.

2 Related Work

Starting from a task model for model derivation is important for these reasons:

1. It is a naturally important knowledge source where the task is expressed according to the user words, not in terms of system vocabularies [14].
2. Other models do not contain information expressive enough to derive something significant. The expressiveness power of the derivation can be improved.
3. Other types of models, like data models or domain models are not especially appropriate as they induce a set of particular tasks (e.g., insert, delete, modify, list, check, search, print, transfer), which do not necessarily match the user task. Rather, to carry out another interactive task, which can be any combination of parts or whole of these tasks, the user has to switch from one predefined task to another. This is not user-centered. Exceptions can still occur: for instance, Teallach [1] derives task, presentation and dialog models from a domain model provided by the underlying data base management system. In this case, UIs for data-intensive systems are the target applications, which is a specific domain in information systems.
4. Finally, derivation of other models from a task model is also allowed (e.g., an activity-chaining graph from a task model [13]) to come closer to the designer's world, but there is a potential risk of information loss in the transformation [2].

Tam *et al.* focus on structural relations: by browsing the hierarchical decomposition, the amount of sub-tasks for each level and the amount of levels can be computed to let the designer specify a preference ranging from one window presenting all ele-

ments corresponding to all sub-tasks to many windows presenting for example the different levels of sub-tasks, level by level (Fig. 2).

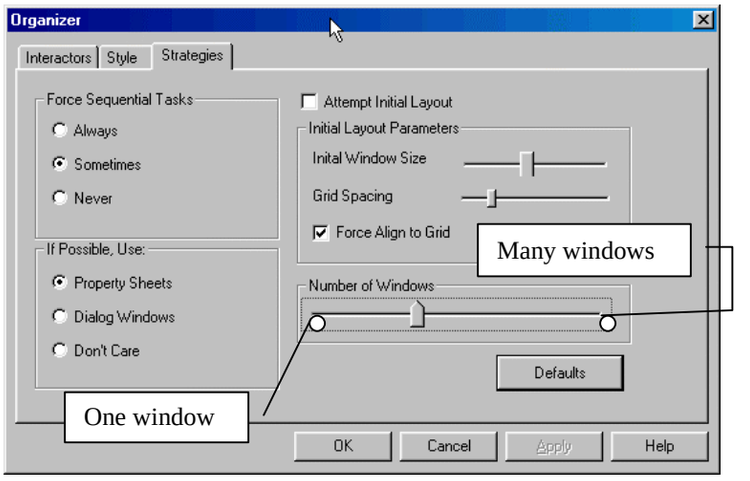


Fig. 2. Derivation of presentation elements from a task model.

SEGUIA [16] also groups presentation elements based on structural relations: a presentation unit is selected for each sub-task of the first level, 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 metric is not exceeded.
Paternò *et al.* have investigated how a presentation model can be derived from a task model [7] specified in the ConcurTaskTrees notation [6]. In this research, a bottom-up approach is exploiting first the structural relationships, and then the temporal relationships to group presentation elements based on the concept of an activation set. Such a set groups the presentation units corresponding to sub-tasks which can be carried out in some related way. An interesting feature of Teallach [2] is its ability to derive new models from existing one and link any pair of models in task, presentation and dialog models.

Zhou [15] developed a planning-based system that infers the presentation of highly interactive graphics according to constraints on presentation elements manipulated through a task model. This inference is based on a form of visual discourse where sequences of temporally ordered actions of a model are parsed to automate the presentation of the elements manipulated by each action.

3 Derivation Rules for Presentation and Dialog Models

Before defining and exploring derivation rules, some hypotheses need to be set up. In the rest of this paper, the ConcurTaskTrees notation [6] is used for representing the task model as it already supports both structural and temporal relations commonly

used in task analysis. The Window Transition notation is used for representing the mid-level dialog, which is located at the window interaction level.

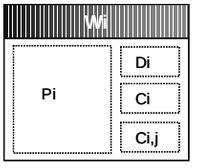


Fig. 3. Composition of presentation and dialog elements for W*i*.

First, a mono-window approach is adopted here to simplify the problem: one window is derived for each (sub-)task element appearing in the task decomposition. A multi-window approach where several windows are derived either for a single (sub-)task element or for grouped elements has already been analyzed in [13]. Let T*i* denote the i^{th} task or sub-task element in this task model and W*i* the corresponding window (Fig. 3). Only system tasks with or without user feedback (such as user confirmation for launching an automated process) and interactive tasks are here considered. No user tasks or other types of task (e.g., mechanical task on a machine) are covered as they are not linked to the interactive system.

Let P*i* denote the presentation part which is proprietary to W*i*. P*i* typically consists of presentation elements like control widgets: edit box, radio button, check box, and list box. Let D*i* denote the dialog part which is proprietary to W*i*. D*i* typically consists of dialog elements like "Ok", "Cancel", "Close", "Search" push buttons, icons, command gestures, function keys. Let C*i,j* denote a dialog widget allowing a transition from W*i* to W*j*. A *window transition* is defined as any mean to go from one source window (WS) to any target window (WT), where WS and WT are two windows materializing parts or whole of (sub-)tasks. More formally, WS $\xrightarrow{A}$ WS $\Leftrightarrow$ one event generated within source window WS (by user or system) enables a transition of type A to target window WT. A graphical representation of this transition exists (Fig. 4): any window is represented by a window icon, a transition of type A, by an arrow labeled with A, and an arrow marker specifyies that the user can work with WS, while working with WT. If no such marker exists, the user can only work with WT.

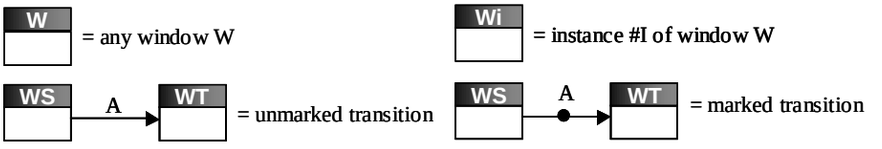


Fig. 4. Graphical notation for a window transition.

A set of basic window transitions is defined in Fig. 5 to specify what are the operations performed on the source and target windows when the window transition is achieved.

Transition icon	Transition name	Transition definition
	maximization	enlarge the window to the maximum possible size to fir the screen boundaries
	titling	reduce the window to its title bar. Note: the window is still active in this position
	minimization	reduce the window to its icon
	display with tiling technique	arrange the window so that it appears tiled with possibly existing windows.
	display with normal overlapping technique	arrange the window so that its size and locations allow partial overlapping of windows. When the window has been reduced before, e.g., by a minimization or a titling, then it is restored to its previous size and location.
	display with user-defined overlapping technique	arrange the window so that its size and locations allow a partial overlapping of windows as defined by the user.
	display with system-defined overlapping technique	arrange the window so that its size and locations allow a partial overlapping of windows as defined by the system. When the window has been reduced before, e.g., by a minimization or a titling, then it is restored to its system-defined size and location
	closing	close the window. Note: it can prompts the user to save unsaved documents or information

Fig. 5. Notation for basic window operations.

Third, it is assumed that widget selection is performed independently as follows: each domain element manipulated by the task element T*i* gives rise to one or several presentation elements to be included in the presentation model and to be manipulated in the dialog model, e.g. through abstract interaction objects [5,13]. These objects are turned into concrete ones, belonging to P*i*.

Fourth, for the simplicity and the concision of this paper, and without any loss of generality, it is assumed that any parent task element (except the root element) is decomposed into two child task elements. From this simplification, a generalization can be easily applied by expanding the derivation rules for any combination of elements.

3.1 The Enabling Operator: T2 >> T3

According to previously stated hypotheses, let us assume that T1 is decomposed into two sub-tasks T2 and T3 which are related by the enabling operator. Fig. 6a 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 as "Ok", "Close", "Cancel".

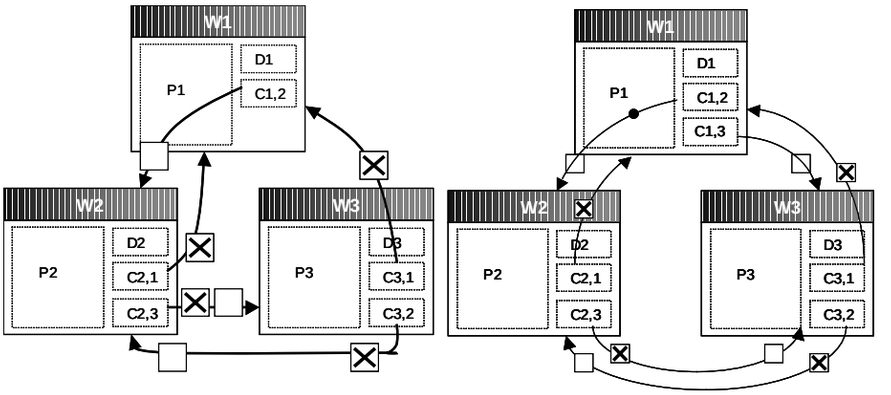


Fig. 6. Alternative specifications for the enabling operator (a,b).

This solution reinforces sequentiality between T2 and T3, but does not visually present T3 accessible from T1. Another solution might be feasible where the above dialog is enriched by presenting the access to T3 (by activating its control C1,3) after T2 is achieved in W2. This solution presented in Fig. 6b emphasizes circularity between elements: C2,1 can be "Ok", "Cancel", C2,3 can be "Next" or "Name of T3", C3,1 can be "Ok" or "Cancel", C3,2 can be "Previous", "Back" or "Name of T2".

3.2 The Enabling Operator with Information Passing: T2 [I]>> T3

This operator is much more rich than the previous one as it allows the presentation of I, the information passed from sub-task T2 to sub-task T3. Let us denote by WI the window materializing I, P*I* the presentation of this information and by D*I* the dialog related to this information if needed. For instance, it could be a "Search" push button, a "Validate" icon. In order to explore 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 in some certain interaction. When no interaction is required, this information is said *observable*. When some interaction is required, this information is said *browsable*. The introduction of this property is motivated by the observation that not all information items should be displayed all together at a 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. Information browsable can be in this state *internally* or *externally* depending on the information is presented within or outside the scope of the initial window, respectively. In the last case, the dialog can be *modal*, respectively *modeless*, if the externally browsable information should be terminated, respectively not terminated, before returning to the initiating window. Information simultaneously visible in multiple task elements is said *shared*. The designer can specify these parameters either globally for a particular task or locally by redefinition for particular sub-tasks.

Fig. 7 depicts a decision tree resulting from the examination of different configuration cases induced by visibility from sub-tasks 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) configurations.

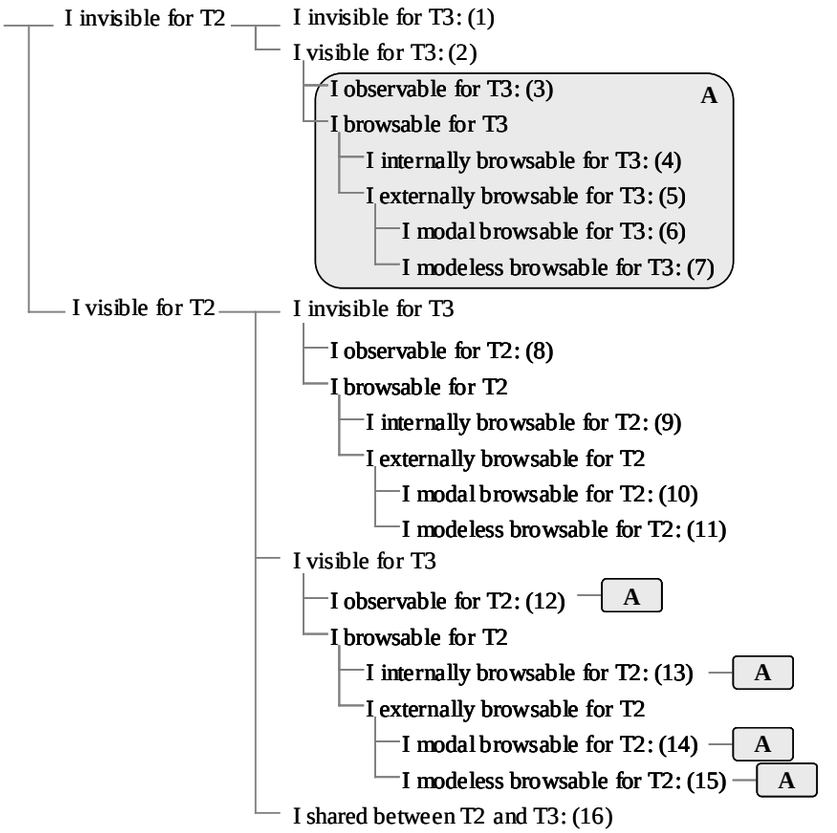


Fig. 7. Decision tree of alternative specifications for the enabling operator with information passing.

Configuration (1) is similar to the simple enabling operator (Fig. 6a). Configuration (2) is similar to that operator too, but W3 is replaced by the window configuration reproduced in Fig. 8a. When I is not observable for T3, it is considered that I

should be browsable in some way. When I is internally browsable (4), Fig. 8b 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 (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. 8c.

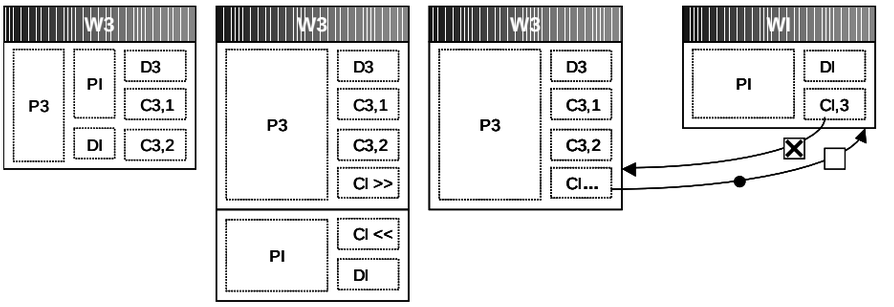


Fig. 8. Particular configuration changes when I is invisible for T2 **(a,b,c)**.

When I is observable for T2 (8), only W2 should be replaced in Fig. 6a by Fig. 9a. When I is internally observable for T2 (9), W2 becomes structured as represented in Fig. 9b. And, similarly to above, when I is externally observable for T2, Fig. 9c provides the change for modeless case (11). For modal case (10), the marker is removed.

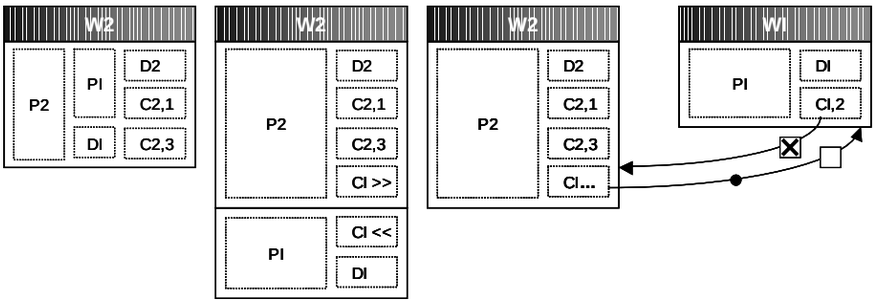


Fig. 9. Particular configuration changes when I is visible for T2 **(a,b,c)**.

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. 10a 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 is only displayed once.

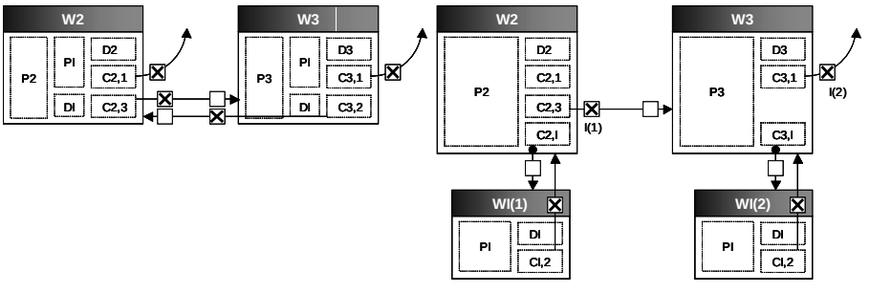


Fig. 10. Particular configuration changes when I is visible for both T2 and T3 **(a,b)**.

This reasoning also holds for subsequent categories (13), (14) and (15) except the case where the two instances of WI are modeless. They should be independent due to the enabling operator between the two sub-tasks: this is why closing W2, respectively W3, also closes the first, respectively the second, instance of WI as shown in Fig. 10b. Fig. 11 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 once. WI 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).

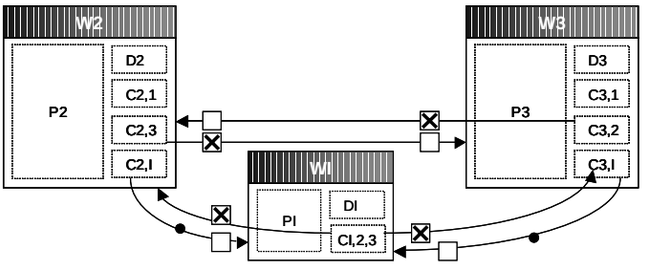


Fig. 11. Particular configuration when I is shared among T2 and T3.

Although theoretically possible configurations are gathered in Fig. 7, it might be desirable to avoid some configurations for usability reasons. Here is an incomplete set of guidelines providing some assistance to the designer this way:

- The invisibility of I from both T2 and T3 (1) should be avoided since no user feedback of I is produced towards the user.
- Modeless browsability of I in T2 and/or T3 should be preferred to modal browsability as the window initiating WI and WI can be used simultaneously, thus not preventing the user to forget information in the initiating window.
- I should be visible in W3 in some way as it will be used to achieve T3.
- WI, as every window, should be controlled by one dialog element at a time, but several dialog elements concurrently. It can be updated according to the results produced in other related windows.
- 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.

3.3 The Suspend/Resume Operator: T2 | > T3

The suspend/resume operator can be specified thanks to the reducing and restoring set of operations defined in [7]. In Fig. 12, W2 reduces itself and restores W3 consequently and vice versa. Of course, it is assumed that W1 initiated both W1 and W2 before with a marked transition. Each window can close itself, thus returning to W1.

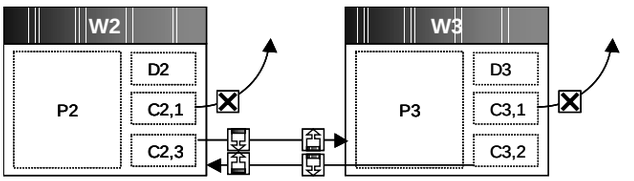


Fig. 12. Specification for the suspend/resume operator.

3.4 The Choice Operator: T2 [ ] T3

Sub-tasks T2 and T3 are considered mutually exclusive in their decomposition with the choice operator. Therefore, this choice can be either *implicit* when the wanted sub-task is initiated from its parent, deactivating the other, or *explicit* when the wanted sub-task is initiated by directly selecting it, automatically reducing the other. In Fig. 13a, the choice between W2 and W3 is made implicit: C1,2 and C1,3 are active at the beginning, but the selection of any of them automatically deactivates the other. For instance, C1,2 is deactivated if C1,3 is cliqued on and vice versa. In Fig. 13b, the choice between W2 and W3 is made explicit: first, W2 and W3 are simultaneously activated, but reduced, from C1,2,3. Closing W2 restores (thus, reactivating) W3 and vice versa. C2,1 and C3,1 are intended to support the achieving of sub-tasks, thus returning to the parent task element.

W2 and W3 are related to each other with a XOR operator. Any other logical operator (AND, OR) and combination of these operators, for example, (W2) XOR (W3 and W4), expressed in first order logic formula should be worth to investigate. AND/OR/XOR graphs provide probably a better notational support in this situation as shown in the Activity Chaining Graph (ACG) of TRIDENT [13].

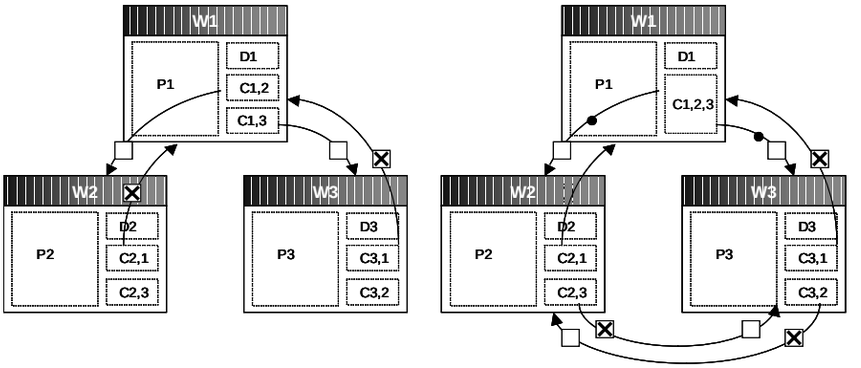


Fig. 13. Alternative specification for the choice operator: implicit **(a)** or explicit **(b).**

3.5 The Disabling Operator: T2 [> T3

The disabling operator mostly relies on the set of reducing operations. The anchor defined in W2 should be responsible for holding the window transition between W2 and W3 (Fig. 14). Another dialog element C3,2 completing the inverse process can be imagined as well.

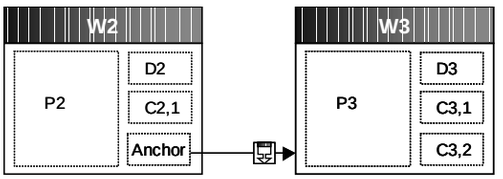


Fig. 14. Specification for the disabling operator**.**

3.6 The Iteration Operator: T2*

In the iteration operator, the T2 sub-task is repeated *n* times, where *n* is unknown. In these circumstances, the creation and access of the multiple instances of T2 can be specified *with* or *without* accumulation, depending on task and user parameters. When multiple instances of T2 are accumulated in many instances of W2 (Fig. 15a), the first W2 instance is created from W1 and subsequent instances ($i = 2,\ldots,n$) from this instance. Each new instance is created from its predecessor, can be closed independently and can return to its predecessor. Multiple instances of W2 could be also created from W1, but it forces the user to switch back to this window to create another one. By creating the successor instance from the current window, there is no change of context of use, thus preserving task continuity. W2 instances are therefore accumulated as overlapping windows (e.g., system defined, or user defined).

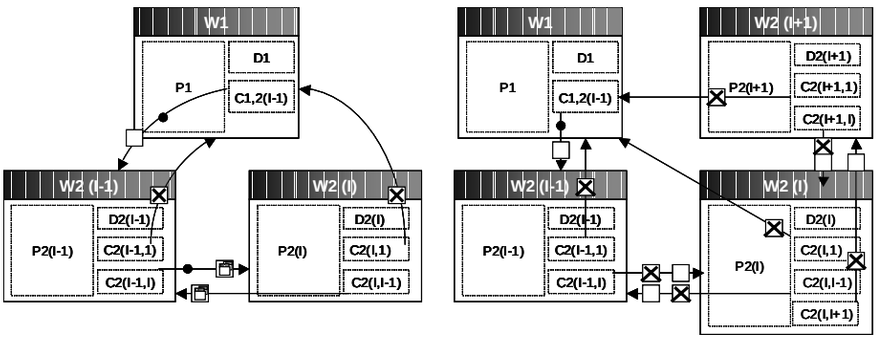


Fig. 15. Alternative specifications for the iteration operator**.**

Conversely, when multiple instances of W2 are not accumulated (e.g., because having too many instances on the screen could become impossible to manage), there

is only one W2 instance at a time, the predecessor being accessible with the CI,I-1 element (e.g., labeled "Previous") and the successor being accessible with the CI,I+1 element (e.g., labeled "Next") with $i = 2,\ldots,n\text{-}1$. This situation is represented in Fig. 15b. Each new W2 instance closes its predecessor. Some interconnexion between the multiple instances is here preserved, thus preventing the user to loose task continuity.

3.7 The Finite Iteration Operator: $T2^n$

The finite iteration operator is a particular case of the general iteration operator. As such both specifications with and without accumulation are still applicable for this operator where *i* is known at design-time rather than at run-time. Therefore, as previously discussed, two additional specifications can be imagined (Fig. 15) provided that the value of *n* does not exceed the cognitive load capacity of the user. Some studies show for example that most current tasks should not involve more than three windows as carrying out more that three concurrent tasks is usually beyond the users' limit. The left specification in Fig. 16 creates *n* instances of W2 when the CI dialog element is operated. The multiple instances then reduce themselves to restore the next instance and so forth. In the right specification in Fig. 15, these *n* instances are considered as completely autonomous as no way switching from one instance to another is provided, except by returning to the parent task. Again, guidelines should be stated here to avoid catastrophic application of this derivation rule.

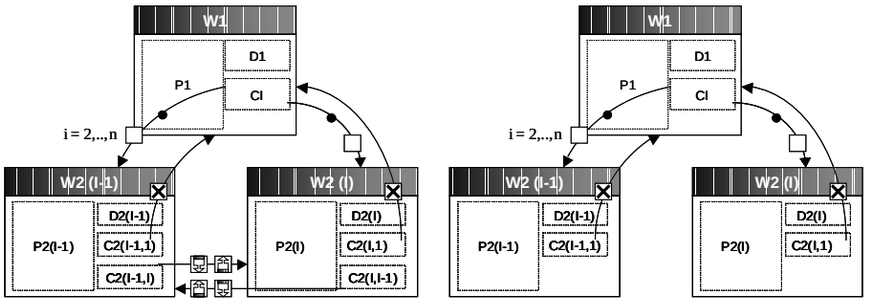


Fig. 16. Other alternative specifications for the finite iteration operator**.**

3.8 The Concurrency Operator: T2 | | | T3

The concurrency operator can be considered of a particular case of the right specification of Fig. 16. Indeed, as T2 and T3 are completely independent from each other, their W2 and W3 windows can be straightforwardly created from W1 with independent closing returning to the parent. However, these presentations are completely independent. Thus, alternative presentations addressing this condition might be tried as well as tabbed dialog box, switching windows or notebook presentations.

According to the working hypotheses, we focus here on two sub-tasks only. When generalizing this derivation rule to many sub-tasks, it might be important to consider

to what extent this rule can be expanded without causing usability problems such as the user getting lost in the work space, the windowitis disease (the user having too many windows to look at simultaneously is becoming ill!), or the user being confused by loosing the focus window. The definition of derivation rules is one aspect to be investigated; the usability of their results is another one that needs validation.

3.9 The Synchronization Operator: T2 |[I]| T3

This operator, as the enabling with information passing operator does, manipulated I, the information passed among sub-tasks. This synchronization operator is rather similar to the enabling with information passing operator, except that no connection should be established between the sub-tasks which are autonomous. Indeed, in the latter operator, the sequence imposes such a relation that the former does not. Therefore, two starting combinations could be imagined:

1. T2 should remain *separate* from T3: in this case, use derivation rules defined in the decision tree of Fig. 6.
2. T2 could be *joined* with T3: in this case, the WI window containing the information passed could be either *independent* (Fig. 17a: WI is controlled from the parent window only) or *dependent* (Fig. 17b: WI is controlled by both parent window W1 and joint window W2+3 containing the merging of T2 and T3).

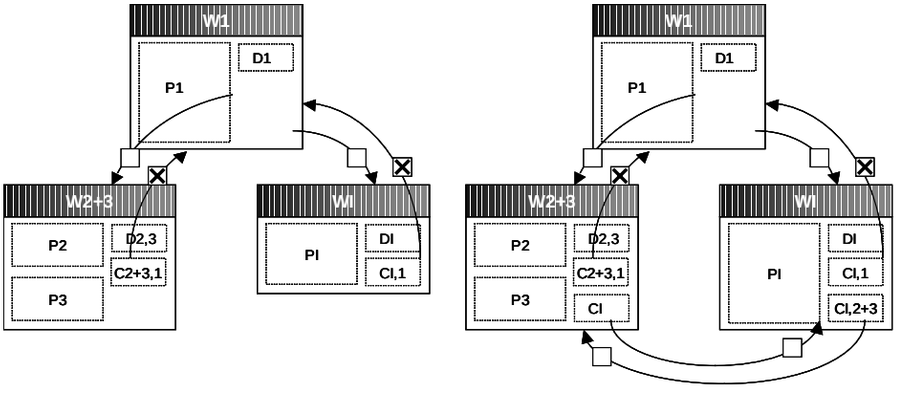


Fig. 17. Other alternative specifications for the synchronization operator **(a,b)**.

4 Example

Fig. 18 depicts the ConcurTaskTrees diagram related to the interactive task “Phone order”, one of the tasks belonging to the general application “Product management”. The scenario resulting from task analysis is described in [7], section 5. The other tasks of this application are not specified here. Since it consists of three sub-tasks at the first level, only the two first are concurrent: indeed, the user could begin by identify-

ing a customer or depositing an order, but these two sub-tasks are required to perform the final confirmation.

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. Only after one of them is the user able to modify the address of the identified person, if needed (optional sub-task). 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.

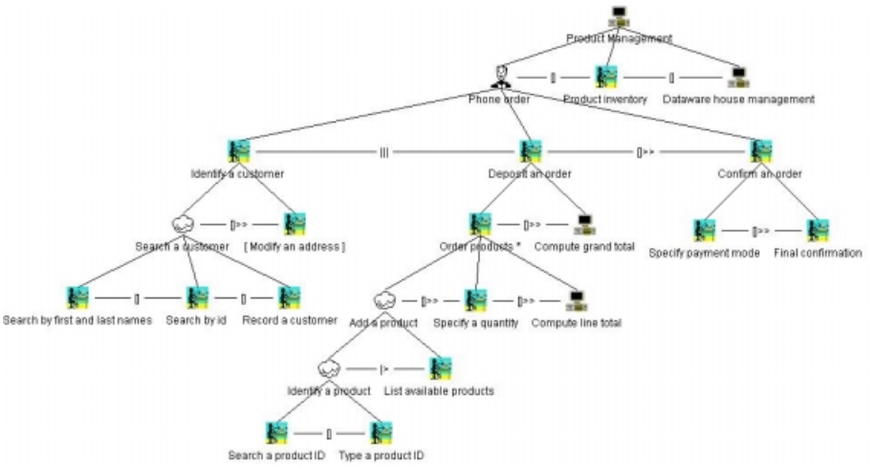


Fig. 18. ConcurTaskTree diagram of the task "Phone order".

Fig. 19 shows a possible use of derivation rules to derive presentation and dialog for the first sub-task of "Phone order". It is assumed here that the names of these three sub-tasks 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 (Fig. 13a) has been selected and applied.

The "Record of 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 after achieving one of them. The optional character of this sub-task 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. 18).

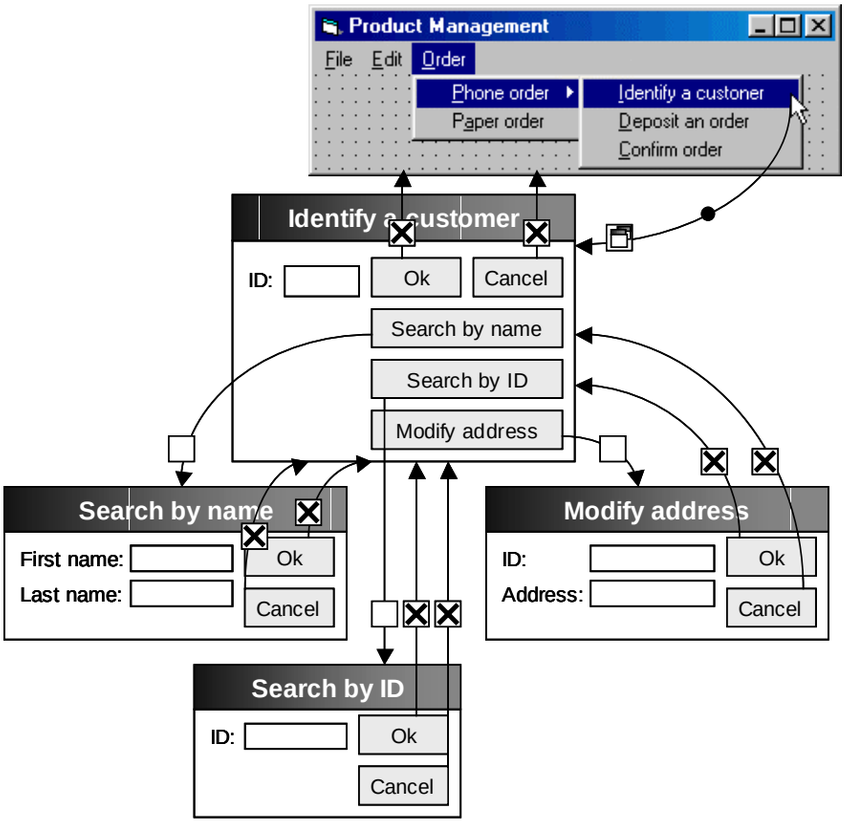


Fig. 19. Derived presentation and dialog for the first sub-task.

Fig. 20 shows a possible use of derivation rules to derive presentation and dialog for the second sub-task “Deposit an order”. Due to the concurrency operator between this second sub-task of “Phone order” and the previous one (Fig. 18), derivation rules of Fig. 16a have been applied between them (Fig. 20 upper left). This produces a window entitled “Deposit an order” from which “Add products” children can be repeated due to the repetition attribute on “Order products”. The derivation rule for infinite iteration without accumulation (Fig. 13b) has been exploited to produce this window, equipped with “Previous” and “Next” buttons. The “Specify a quantity” and “Compute line total” 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 an appropriate push button to trigger the semantic function (system sub-task). 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 sub-task. 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”.

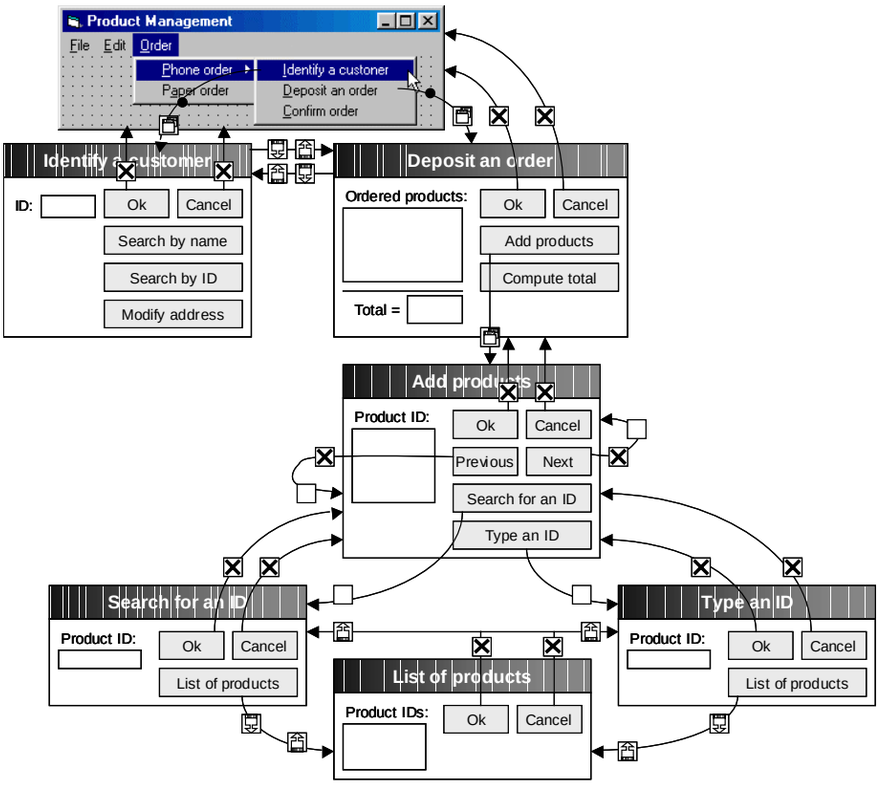


Fig. 20. Derived presentation and dialog for the second sub-task.

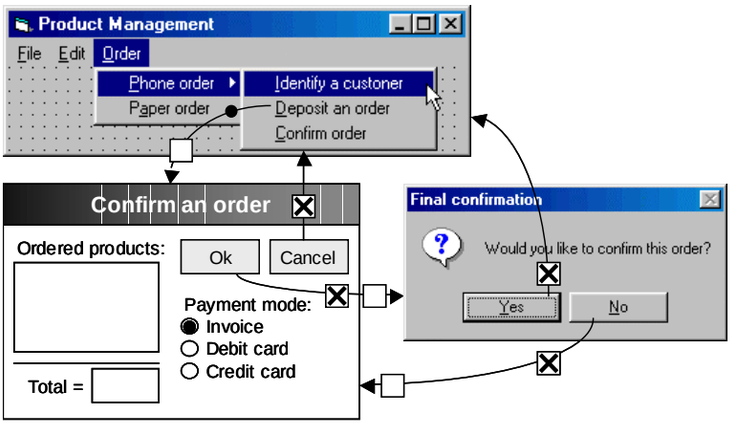


Fig. 21. Derived presentation and dialog for the third sub-task.

Fig. 21 shows a possible use of derivation rules to derive presentation and dialog for the third sub-task "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. 18) therefore produces two sequential windows, one of them is being here refined as a message dialog box. This is a two-step confirmation: the first being by pushing the "Ok" push button on the first window to complete the order with the payment mode and the second being by pushing the "Yes" push button on the message dialog box.

5 Conclusion and Future Work

Future work can be imagined at least along four dimensions:

1. *Dimension of derivation rules*: under the same working hypotheses that have been assumed here, 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. Generalization of these rules should also be investigated by progressively relaxing the hypotheses that have been assumed one by one. This would permit to verify the ability of each rule to be more general. Finally, only the temporal relations have been used here, not the structural relations (which has been simplified through the hypotheses). Combining both types of relationship should be further analyzed.
2. *Dimension of usability of elements derived*: solving the mapping problem for task-presentation and task-dialog is intended to go beyond the unique goal of automated generation of user interfaces. In this context, the ultimate goal is to automate as much as possible the process of producing a user interface that directly satisfies some usability properties by construction. To prove that such a generated UI is usable, it is required to prove that such properties are intrinsically respected. Bastide & Palanque have argued that only starting with a simple task model might be insufficient to reach these properties [2]. Either the task model does not contain information enough to derive something significant or the task model may lead to any user interface that does not necessarily satisfy such usability properties. In the mapping problem, the ultimate goal is rather to construct model elements and to link them with appropriate relationships that inform the development process. This process should be conducted under the supervisory of the designer.
3. *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. 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 such as KLM or a technique like action analysis could be automatically applied on the derived UI to compute their predictive accomplishment complexity and time respectively.

4. *Dimension of tool support for derivation rules*: the definition of derivation rules within a supporting tool have not been examined so far nor the automatic evaluation of guidelines at design time as a critique method. To allow such a tool support, a corresponding formal method is required.

Another direction for future extensive research is the introduction of domain and/or user models along with the task model to derive presentation and dialog models. It could be worthwhile to see how knowledge captured in domain and user models affect the derivation of presentation and dialog models. In particular, the influence of a user model on the presentation model should be very interesting. But this is another story.

6 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. In: Vanderdonckt, J., Puerta, A. (eds.): Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 139–158
2. Bastide, R., Palanque, Ph.: Conformance and Compatibility between Models as Conceptual Tools for a Consistent Design of Interactive Systems. Position paper for the CHI'99 Workshop "Tool Support for Task-Based User Interface Design" (Pittsburgh, May 15-20, 1999).
3. Birnbaum, L., Bareiss, R., Hinrichs, T., Johnson, C.: Interface Design Based on Standardized Task Models. In: Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'98 (San Francisco, January 1998). ACM Press, New York (1998) 65–72
4. Elnaffar, S.S., Graham, N.T.: Semi-Automated Linking of User Interface Design Artifacts. In: Vanderdonckt, J., Puerta, A. (eds.): Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 127–138
5. Johnson, P., Johnson, H., Wilson, S.: Rapid Prototyping of User Interfaces Driven by Task Models. In: Carroll, J.M. (ed.). Scenario-based Design: Envisioning Work and Technology in System Development. John Wiley, New York (1995) 209–246
6. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. of IFIP Int. Conf. on Human-Computer Interaction Interact '97 (Sydney, July 1997). Chapman & Hall, London (1997) 362–369
7. Paternò, F., Breedvelt-Shouten, I.M., de Koning, N.M.: Deriving Presentations from Task Models. In: Proc. of IFIP Workshop on Engineering the Human-Computer Interaction EHCI'98 (Creete, September 1998). Kluwer Academics Publisher, Dordrecht (1998)
8. Paternò, F.: Model-Based Design and Evaluation of Interactive Application. Springer Verlag, Berlin (1999)
9. Paternò, F, Mancini, C.: Developing Task Models from Informal Scenarios. In: Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'99 (Pittsburgh, May 1999). ACM Press, New York (1999) 228–229

10. Puerta, A.R., Eisenstein, J.: Towards a General Computational Framework for Model-based Interface Development Systems. In: Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'99 (Los Angeles, January 1999). ACM Press, New York (1999) 171–178
11. Stirewalt, K.R.E.: MDL: a Language for Binding User Interface Models. In: Vanderdonckt, J., Puerta, A. (eds.): Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics, Dordrecht (1999) 159–170
12. Tam, R.C., Maulsby, D., Puerta, A.R.: U-TEL: A Tool for Eliciting User Task Models from Domain Experts. In: Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'98 (San Francisco, January 1998). ACM Press, New York (1998) 77–80
13. Vanderdonckt, J., Bodart, F.: Encapsulating Knowledge for Intelligent Interaction Objects Selection. In: Proc. of ACM Conf. on Human Aspects in Computing Systems Inter-CHI'93 (Amsterdam, 24-29 April 1993). ACM Press, New York (1993) 424–429
14. Wilson, S., Johnson, P.: Bridging the Generation Gap: From Work Tasks to User Interface Designs. In: Vanderdonckt, J. (ed.): Proc. of the 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur-5-7 June 1996). Presses Universitaires de Namur, Namur (1996) 77–94
15. Zhou, M.: Visual Planning: A Practical Approach to Automated Presentation Design. In Proc. oF IJCAI'99 (Stockholm, August 1999).