Designing the User Interface on Top of a Conceptual Model

Matti Pettersson

Department of Computer Science, University of Tampere
P.O.Box 607, FIN-33101 Tampere
mpe@cs.uta.fi

Abstract: The paper shows how a conceptual model represented as a combination of an Entity-Relationship diagram and a Petri net can be used in user interface design. We first discuss how the conceptual model can contribute to the design of usable systems in general. We then describe the specification of the interface as a refinement process from the events of the conceptual model to the operations of the interface.

1 Introduction

One of the critical issues in software engineering is how to get user interface design as an integral part of software development. Traditionally software engineering methods seem to take quite diverse standpoints. Some methods suggest that user interface design is part of the requirements engineering, some consider it as being part of the implementation.

Within HCI literature it is often stated that an interactive system should be built on a conceptual model [24, 8, 17]. However, there seems not to be any consensus of the role of such models. Possible interpretations vary from an overall description of the application domain to an implementation concept (as an example see the discussion reported by Cockton, [5]). The point of view taken in this paper is that the designer of a direct manipulation interface needs a model of the application domain as the interface should show the user the concepts and behaviour of the application domain on the screen [23, 24, 17, 12]).

There is not much discussion in HCI literature how a conceptual model in general or a model of the application domain can be built and represented and how exactly it relates to the user interface design (c.f. [14] as an exception see [1]). The techniques used to describe user interfaces seem to suppose that the user interface is an independent part of the software and start the design from scratch (as an example see [10]).

Conceptual modeling is often discussed within information system research and particularly within the data base community, where several analysis methods have been suggested to modeling of the concepts and behaviour of the application domain (as an example see [18]). Heuser, Peres and Richter [11] use a combination of Entity–Relationship modeling [3] and Petri nets [19, 22] to describe static and dynamic features of the application domain, respectively. In this paper we show how user interface design

can benefit from representations created in the form of a conceptual model in general. Moreover, we show how the user interface can be specified on top of a conceptual model. The benefit of the approach described is that it shows how the user interface design can be viewed as a continuation of the modeling of the application domain, both conceptually and methodologically. The method described is in a trial use in a project, where a weather monitoring system is being developed to support road maintenance activities depending on changing weather conditions. The system consists of several graphical workstations connected to a large data base.

The organization of the paper is as follows. We first introduce the conceptual model modified from [11]. We then discuss the usefulness of the model from the point of view of user interface design. We then proceed by a general discussion of interactive systems as the details of using the model as a basis for user interface design depends on the type of interaction we are aiming at. Finally we show a specification of the user interface as a Petri net.

2 Conceptual model

Figure 1 shows the conceptual model from [11]. The part of Figure 1 drawn in heavy lines displays an E–R diagram for the IFIP working conference case study [18]. Entities and relationships are represented as usually. Light lines, circles and boxes are added to model the dynamics of the system as a Petri net. They describe the events that manipulate the data within the system.

A simplest form of a Petri net appears towards the lower left corner of Figure 1. It consists of two circles, or places, and of a box, or a transition. Places are labeled "Receipt open" and "Receipt closed". Transition is labeled "Close". There is a token, or marking, in "Receipt open" indicating the state of the receipt. Transition "Close" corresponds to the event of closing the receipt. When the transition "Close" fires, the token is moved from its input place "Receipt open" to its output place "Receipt closed" to indicate the new state of the receipt. In the model, the letter 'e' labels the edges describing the path the token is moved. In a Petri net, the firing of a transition is possible if, and only if, there is a token in each of its input places and there is no token in any of its output places. This is the mechanism controlling the firing of transitions.

The type of net in Figure 1 is actually a condition/event net. The tokens within the net are tuples of the form <x1,...,xn>, where x1,...xn are variables. Each place can contain several tuples of the same type as long as the values of the variables differ. And instead of a token, the transitions (or events) move sets of tuples while firing. Each tuple corresponds to a data item. For example, event "Submit paper" inserts a set of tuples of the form <pap, top> to place "Submitted paper with topic". Tuple <pap, top> represents an item consisting of a paper and its topic. Event "Submit paper" has no input places, so from the system's point of view submitting is always possible. Whether the papers submitted are sent to referees or ignored depends on the state of the receipt. Event "Receive paper" is enabled only when the receipt is open, otherwise event "Ignore late paper" is enabled.

Note the double–head arrows. They are used to represent restoring actions. In a Petri net, the token is no more available in its original place after the transition has fired, unless the transition explicitly restores the token. For example, event "Receive paper" would take the token from "Receipt open" thus effectively closing the receipt, with the exception that the token would not end to "Receipt closed". But as restoring actions are quite common in practice, a special symbol is used for it.

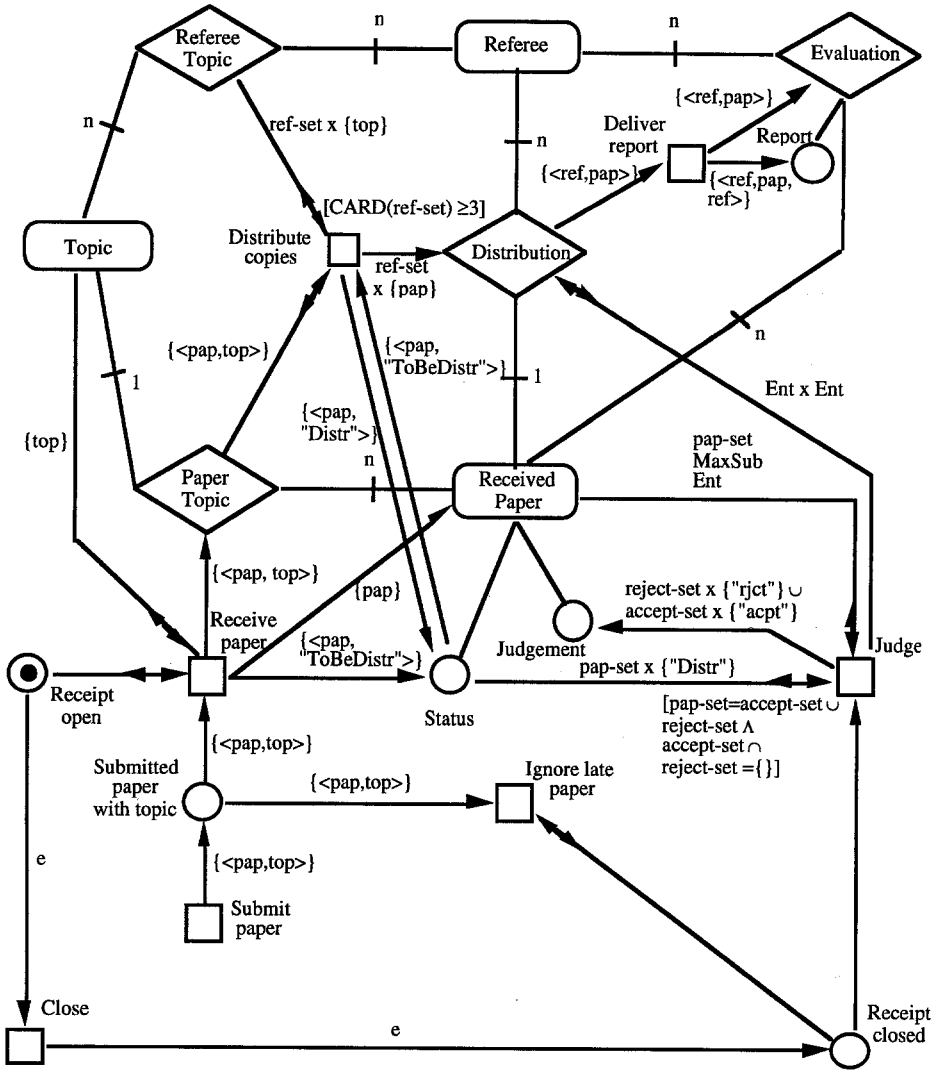

Figure 1. Conceptual model for the IFIP working conference case study.

In a condition/event net, there are two additional mechanisms to control the firing of the events. The first one is variable instantiation. All instantiations of a variable related to a certain tuple must be the same around an event. For example, there are several instantiations of variable "top" around event "Receive paper". Firing of the event takes each tuple available in "Submitted paper with topic" and compares it against the topics of the conference in "Topic". If a match can be found, the tuple is moved forwards as indicated by the arrows leaving the event.

The second mechanism to control the transitions is conditions associated with events. In Figure 1, conditions appear in brackets, '[]'. For example, the condition associated with event "Distribute copies" states that each paper should be distributed to at least three referees.

In the model, the E–R diagram and the Petri net are integrated in such a way, that beside circles, both the entities and the relationships of the model can serve as places. Moreover, each event must respect the rules implicit in the ER–model. These rules are discussed in the next section.

3 Conceptual model and user interface design

In this section, we first take a look at the static aspects of the model, i.e. the E–R diagram and discuss how it contributes to the user interface design. An E–R model describes the data manipulated within the system. As such it gives the user interface designer useful information concerning

- the vocabulary of the domain,
- the data and the attributes to be displayed on the screen,
- the connections between pieces of data,
- cardinalities of the relationships, and
- the rules for preserving the integrity of the system.

In the following, we take a closer look at each of these.

In an ideal case the conceptual model is built up together with the user of the system. Thus it reflects the user's terms of the entities of the domain. When these terms are used in the user interface, the interface is based on the user's language, thus fulfilling one of the usability principles by Molich and Nielsen ([15], see also [16]).

Data is central in direct manipulation interfaces. Systems are usually built to manipulate data. The early successes of WYSIWYG editors were based on the fact that they showed at their interface the data the user was manipulating [23]. As a consequence of making the data visible, it was possible to make the operations and their results visible and more direct [12]. The user got feedback of the actions in a natural way and her memory load was decreased.

Taking the data in general and the static aspects of the system in particular as a starting point for the user interface design helps in keeping the interface easy to control by the user. The sequences of operations remain short thus reducing the navigation effort and the number of steps needed while moving from a task to the next. Dialogue remains simple.

By showing the connections between pieces of data, a conceptual model provides a meaningful structure for the data. Moreover, it shows all the possible ways how the user may want to access a piece of data. For example, in the case of Figure 1, the user may be interested in whether a paper refereed by a given referee was accepted or not. In practice, it may be impossible to incorporate all possibilities under one graphical interface, without visualizing the whole SQL. The first task of the user interface designer is to try to catch the essence of the system under development. In this particular case, the essence might be managing papers in the referee process, which suggests that it is plausible to focus on paper objects in user interface design.

Cardinalities of the relationships are important for the design of the user interface. Cardinalities differing from 1:1 suggest that the interface must be capable of showing several data items instead of only one. As an example consider showing the referees of a given paper. Moreover, systems based on the model must not override the cardinalities in the model. For example, the model in Figure 1 states that no more than one paper should be distributed to any referee. Thus the system based on the model should not allow distributing more than one paper per referee. The user interface may be designed in such a way that it makes this constraint visible. This prevents errors thus increasing the usability of the system [15].

An E–R model includes some integrity rules implicitly. An example is the referential integrity rule stating that an entity participating in a relationship, must exist [7]. For example, it is not allowed to have an entry in "Paper Topic" relationship, if there are no corresponding paper and topic entries in the system. From the point of view of the user interface design this means that the existence of the entries must be checked at some point of interaction. Part of the design is to decide upon the policy to be followed to maintain the rule.

The dynamic part of the conceptual model shows the actions the system should take to manipulate the data described by the static model. Part of the actions may be automatic, whereas some of them require interaction with the user. The user interface designer's task is to turn the actions requiring user's activity into a user interface. Moreover, the model shows the prerequisites and consequences of actions, as the user expects them to be.

The dynamic part of the model also makes explicit the conditions that must be met in order to fire the event. This helps designing error messages that are both precise and constructive. Precise error messages provide the user with exact information about the cause of the problem. Constructive error messages provide meaningful suggestions to the user about what to do next [15].

There are several proposals of how the components of the interface relate to the components of the application. The problem is that there is usually no one–to–one correspondence between the components of the two types. Data from an application object is displayed by several interface objects and an interface object shows data from several application objects. Properties of the objects of the two domains can not easily be put together in one object.

One possible solution is to go to the attribute level, where a correspondence can be found more easily. This is suggested in [8]. However, from the point of view of designing the interface, it is not desirable to consider the interface as consisting of separate attributes, each of which can be controlled by the user, but to try to keep attributes and operations relating to one application object close to each other.

A correspondence can be found between user's operations at the interface level and the events of the system. For each event of the system requiring user's activity there must be at least one operation or a sequence of operations in the interface to fire the event. One of the challenges of user interface design is, how to make these operations visible to the user. This suggests a model of an interactive system, where the events requiring user's activity connect the user interface and the system and serve as a starting point to user interface design. A direct manipulation interface assumes close interaction between the user interface and the rest of the system. How exactly this interaction can be implemented is discussed in the next section.

4 Interaction between the interface and the system

One of the important discussions around user interfaces in the 80's was the discussion of the architecture of interactive systems. The Seeheim model of interactive systems [9] got a status of some kind of a standard. The basic idea was to separate the user interface part of the system from the rest of the application and then further divide the interface in three parts, each having a role of its own in the processing of the exchange between the user and the system. The architecture was an analogy from compilers. The tasks of the components were to do lexical, syntactic and semantic analyses to user's input and provide as a feedback information of the success and failure of each step, and finally show the results of the computation.

The evolution towards graphical, direct manipulation interfaces questioned the applicability of the Seeheim model. It turned out that the separation was not as simple as in the case of command or menu based interfaces [26]. A direct manipulation interface should give the user continuous feedback of the state of the system as a whole. In a sense it blurs boundaries between the feedback types. However, we maintain that the separation to the interface and to the rest of the system is both possible and feasible, but not in the mechanistic way suggested in the Seeheim model (see also the discussion in [6]).

The separation does not make sense from the users' point of view [14]. In many cases we cannot point to a set of concepts and say that these are the interface concepts and to another set of concepts and say that these are the system concepts. A direct manipulation interface shows the semantic (or system) concepts in its interface [24]. But the separation is possible, in principle, from the technical point of view as different features of objects (or concepts) are central in the interface and in the rest of the system. From the point of view of the interface, the visual appearance and visual behaviour of the objects and the spatial relations between objects are central. From the point of view of the rest of the system, the state of the objects appearing as values in the data structures of the system is central.

For example, Bødker [2] discusses whether the size of a font is a system concept or an interface concept. Quite correctly she argues that it is both. But from the point of view of the interface it is central how a character of a certain font type and size is displayed and at what point of the screen. From the point of view of the system, the font type and size are only values at a certain point in the data structure of the system.

The maintainability of the system seems to be the most important benefit from the separation. In itself, it increases the modularity of the system in a meaningful manner as not all modifications cross the boundary. A portion of changes consists of changes either in the interface only or in the system only. Unfortunately there are no figures available to describe what the actual portions might be. A successful separation also increases the portability of the software between windowing environments.

Moreover, if the development project relies on some method of system analysis, the implementation should maintain the results of the analysis. In case of a change in requirements, it is easier to modify the system, if there is a clear connection between the results of the analysis and the code that implements the system. For example, if object analysis is used to find the most suitable objects to describe the application domain, the objects should be easily found at the code level too. The common method of deriving system objects from interface object classes usually requires decomposing the application domain objects into several interface objects as there is no one–to–one relationship between the interface objects and the application domain objects. For the same reason, decomposing the interface inside application domain objects spreads the interface all around the code, which again decreases maintainability and portability of the system.

The separation of the application objects from the interface objects may of course in object oriented code increase the number of objects and the communication overhead, but for example, C++ programming language has tools to make this communication as efficient as references within an object.

Separating the interface leads to another problem requiring a solution: which part, the system or the interface (or both) has the control [4]. A common solution in graphical environments is that the interface has the control. This is a feasible solution as in this way it is easier to implement the system so that the user is given the sense of controlling the system and not the other way around.

Letting the user have the control over the system does not mean that the user should be able to do anything she likes independently of the state of the system, even if many of us would like an automatic teller machine which gives us money regardless of the balance of our account. As the conceptual model in Figure 1 describes, each action, the user wants to take with the aid of the system has a set of conditions, to be checked before the operation may be executed. Whatever the type of the interface, the task of the interface is to give the user feedback on whether the function she wants to execute can be executed or not. Typical to a direct manipulation interface is an attempt to check the conditions associated with the operations as early as possible, in order to keep the user informed of the state of the system. This check may in some cases be done even beforehand. As an example consider the dimmed choices in your favourite menu based word processor.

This requirement for direct manipulation interfaces means that there must be a continuous interaction between the interface and the system. There are two consequences. First, the model for an interactive system should not only show that there are two components, the interface and the system, but show also how the components interact with each other. Second, the system and the interface cannot be layered on top of each other in a mechanistic way, but rather the design of the interface should be seen as a refinement from the actions in the system to the operations of the interface [20].

5 User interface specification

In this section, we show how the user interface can be specified as a refinement from the events of the conceptual model in Figure 1. The structure of the Petri net allows the refinement of each event separately. In practice, the interface designer may want to consider the events in parallel as they may be parts of the same interface, having its own dynamics too, like sequencing of dialog boxes or selecting suitable tools for an operation. In this paper we deal with only one event to keep the size of the example small.

In the following section we specify an interface for assigning referees to papers. In some designs it may be considered an automated function, but in this paper we assume that the user wants to have a tight control over the assignment process and thus it should be interactive. The starting point for the user interface design is "Distribute copies" event in Figure 1. The following preconditions for the event can be derived from the conceptual model:

- Paper is drawn from "Paper Topic".
- Paper status is "ToBeDistr".
- Referee is drawn from "Referee Topic".
- Referee's topic and paper's topic must match.

The event has also a set of post conditions:

- Each paper–referee pair formed is inserted to "Distribution". No pair may appear more than once.
- A paper must be distributed to at least 3 referees.
- Paper's status is changed to "Distr".
- Each referee must not refer more than one paper.

Let us assume that we want to specify a user interface, where referees are assigned to papers by dragging them (or labels with their names) with the mouse on top of the papers. The interface shows a list of papers and a list of referees available. The topics are visible in both lists. The user presses the mouse button on top of a referee's name and drags the name on top of the paper she wants to be distributed to the referee.

No usability tests have been done to test whether the solution works or not in practice and a detailed evaluation falls beyond the scope of this paper. The good point is that all the referees and and papers are visible at the same time. Moreover the interface is quite symmetric in the sense that it does not force the user to select either the paper first or the referee first. Problems may arise if the number of papers and referees increases, but then perhaps the visibility of papers and referees may be limited to one topic at a time. If even this is insufficient, then the process might be automated.

The specification refines the actions in the conceptual model to the level of interface operations. These operations are mouse presses on various graphical elements like buttons, boxes etc. The graphical elements are usually included in user interface toolkits, so the specification may rely on these components. In practice, this means that the specification does not tell us how the user moves the cursor on top of a button and how she presses the button. But the specification should show what happens when the user presses the button.

In the case of a drag operation, we must specify how the suboperations, pressing the mouse button, moving the mouse and releasing the mouse button relate to each other and how the system responds to the suboperations. A drag operation should be standard within an environment and thus needs no specification, but cloning (Pettersson, 1991, 25]. Let us assume that the drag operation starts by the press of the mouse button on an object and ends when the mouse button is released on top of another object. Between these events mouse may be moved at the screen. The drag may be interrupted by releasing the mouse button outside any object. So, there are operations of three types: mouse down, move and mouse up. Each operation generates a point describing the location of the mouse at the moment the event takes place. An overall schema of the drag operation is given in Figure 2.

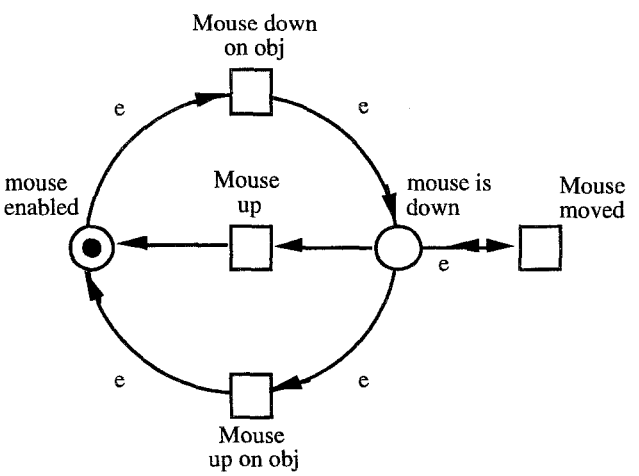

Figure 2. An overall schema for the drag operation.

The next step is to embed the general schema in the context of "Distribute copies" event. Each pre and post condition of the "Distribute copies" should be bound to the suboperations of the drag. There is no general rule how the connecting should be done, rather it depends on the semantics of the operations and the places.
The result of the embedding is shown in Figure 3. In reality, some of the conditions may seem to be unnecessary in the sense, that some combinations may be excluded by the user interface design. For example, the rule that each referee must not have more than one paper to be refereed, may be easily implemented in that way in our exemplary user interface. When the referee is moved on top of the paper, it disappears from the list of referees. However, in Figure 3, we have faithfully included all conditions appearing in the conceptual model. We have also included some details from the conceptual model that are actually not used, to keep the figure more readable.

The explanation of the specification is as follows. The user presses the mouse button. The first condition (Actually, there is no order defined in a Petri in which the conditions should be checked, but here we describe the process in the order that seems natural. In reality ordering should be considered while implementing the interface.) is that the mouse is enabled for this particular operation. The condition bounds the operation to the rest of the interface. Next, if the coordinates of the point the mouse is in at the moment of pressing is within the area of a referee, "Mouse down on ref" event starts to fire.

First it is checked that no other papers are assigned to the referee. In the specification this is described as inserting a tuple consisting of the referee and of any paper to place "Distribution". If this can be done without violating the cardinality constraint, the referee is accepted as selected referee and the tuple is taken back. Selecting the referee enables "Mouse moved", "Mouse up" and "Mouse up on pap" events. If the mouse is moved "Mouse moved" event fires. If the mouse is moved on top of an undistributed paper the paper is highlighted (Feedback is not shown in Figure 2). When the mouse is released on top of a paper, the paper becomes selected. The tuple corresponding to the referee and to the paper selected is added to "Distribution". If the cardinality of the referees assigned to the paper equals three, the status of the paper becomes "Dist" (distributed).

Notice that we have changed the original model at this point. The original model says that "at least three". The specification in Figure 2 limits the number to three referees exactly. This limitation can be relaxed, by not considering the status of the paper while assigning referees, i.e. extra referees can always be assigned to a paper, but the status is changed when there are enough referees (The status is needed elsewhere to check that all papers have been distributed). The need for the modification is a result of assigning the referees interactively, one by one and not all in parallel as is assumed in the conceptual model. (As a matter of fact it is problematic, how assignment could be done without considering an upper limit for the number of referees per paper.)

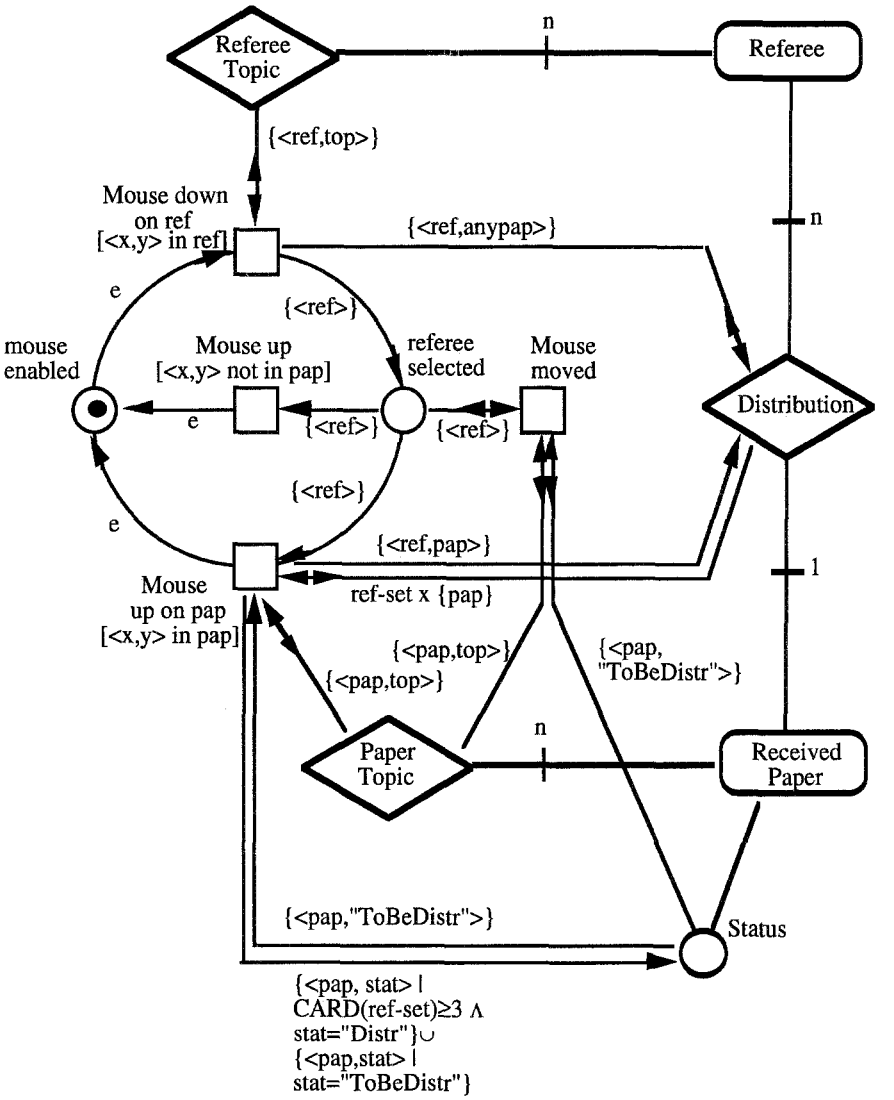

Figure 3. A specification of the interface for the "Distribute copies" event.

The next step in the specification process would be the specification of the feedback associated with each operation of the user. This was partially done in the description above. However, the exact specification of the feedback is out of the scope of this paper.

6 Related work

Petri nets have been used to model user's interactions with the system [27], but not on a model of the application domain. Pettersson [20] uses a layered model to describe the user interface as a Petri net. The lowest layer, the application model represents the application domain, but there is no representation of the static aspects. Janssen, Weisbecker and Ziegler [13] combine the ER model with a variant of a Petri

Janssen, Weisbecker and Ziegler [13] combine the ER model with a variant of a Petri net, but they model the dialogues directly, without considering the dynamics aspects of the application domain.

Quite similar to the use of Petri nets is the approach presented by Sukaviriya, Foley and Griffith [25]. They use pre and post conditions associated with each action on lexical, syntactic and semantic levels. Our approach is different as we consider the specification of the dialogues as a refinement process from the dynamics of the application domain to the domain of the user interface.

7 Conclusions

In this paper we show how the conceptual model of the application domain can be used as a starting point for specifying the user interface. The conceptual model used in this paper was based on a combination of the Entity–Relationship model and of a Petri net [11]. The model seems to be a suitable starting point for specifying the user interface as it shows both the static and dynamic aspects of the application domain. Moreover, as Petri nets are capable of describing parallel processes, strict ordering of events can be avoided in the conceptual model. Thus it supports user interface design, where the user has the control over the system and not vice versa.

The specification process described in the paper is a kind of refinement process from the events of the conceptual model to the operations of the interface. However, it is not a pure refinement in the usual sense. The user interface designer has to consider also the dynamics of the user interface as a whole.

References

1. Braudes R.E. (1991), Conceptual modelling: a look at system–level user interface issues, in Karat J. (ed.), Taking Software Design Seriously, Academic Press.
2. Bødker S. (1987), Through the interface—a human activity approach to user interface design, *DAIMI PB–224, Aarhus University.*
3. Chen P.P. (1976), The entity relationship model: towards a unified view of data, *ACM Transactions on Data Base Systems, Vol. 1, No. 1.*
4. Cockton G. (1987), Interaction ergonomics, control and separation: open problems in user interface management systems, *Report No. AMU8711/02H, Scottish HCI Centre.*
5. Cockton G. (1992), Critical issues: Conceptual design, Larson J. and Unger C. (Eds.) Engineering for Human-Computer Interaction.
6. Coutaz J. (1993), Software architecture modeling for user interfaces, Amodeus Project Document: SM/WP33.
7. Date C.J. (1982), An introduction to database systems, Addison–Wesley.
8. Foley J.D. and van Dam A. (1982), Fundamentals of Interactive Computer Graphics, Addison-Wesley.
9. Green M. (1985), Report on dialogue specification tools, in *Pfaff (1985).*
10. Green M. (1986), A survey of three dialogue models, *ACM Transactions on Graphics, Vol.5, No.3.*
11. Heuser C.A., Peres E.M. and Richter G. (1993), Towards a complete conceptual model: Petri nets and Entity–Relationship diagrams, *Information Systems, vol.18, No.5, pp 275-298.*

12. Hutchins E.L., Hollan J.D. and Norman D.A. (1986), Direct manipulation interfaces, in *Norman D.A. and Draper S.W. (Eds.), User Centered System Design*, Lawrence Erlbaum Associates.
13. Janssen C., Weisbecker A. and Ziegler J. (1993), Generating user interfaces from data models and dialogue net specifications, in *Proceedings of the Interact'93 Conference.*
14. Kuutti K. and Bannon L. (1993), Searching for unity among diversity: exploring the "interface" concept, in *Proceedings of the Interact'93 Conference.*
15. Molich R. and Nielsen J. (1990), Improving a human–computer dialogue, *Comm.ACM, Vol.33, No.3, pp. 338-348.*
16. Nielsen J. (1993), Usability Engineering, Academic Press.
17. Norman D. (1984), Stages and levels in human–machine interaction, *International Journal of Man–Machine Studies, Vol.21, pp. 365-375.*
18. Olle T.W., Sol H.G. and Verrijn-Stuart A.A. (Eds) (1982), Information systems design methodologies: A comparative review, North-Holland.
19. Peterson J.L. (1981), Petri net theory and the modelling of systems, Prentice–Hall.
20. Pettersson M. (1991), Specifying the user interface, *Lic.thesis, University of Tampere, Department of Computer Science.*
21. Pfaff G.E. (ed.)(1985), User Interface Management Systems, Springer-Verlag, Berlin.
22. Reizig W. (1984), Petri nets, An introduction, Springer–Verlag.
23. Shneiderman B. (1983), Direct manipulation: a step beyond programming languages, *IEEE Computer, (August 1983).*
24. Shneiderman B. (1987), Designing the user interface, strategies for effective human-computer interaction, Addison–Wesley.
25. Sukaviriya P.N., Foley J.D. and Griffith T. (1993), A second generation user interface design environment: the model and the runtime architecture, in *Proceedings of the Interact'93 Conference.*
26. Tanner P. and Buxton W. (1985), Some issues in future user interface management systems (UIMS) development, in *Pfaff (1985).*
27. van Biljon W.R. (1988), Extending Petri nets for specifying man-machine dialogues, *International Journal of Man–Machine Studies, Vol.28, No.4, pp 437-455.*