Integration of User Interface and Conceptual Modeling

Babak Amin Farshchian*, John Krogstie, Arne Sølvberg

Faculty of Electrical Engineering and Computer Science
Norwegian Institute of Technology
University of Trondheim, Norway

Abstract We present an approach to integrate the modeling of user interfaces and application systems. The focus will be on the user interface description language. The language has been refined through several case-studies, and are currently integrated with the PPP approach for conceptual modeling. Further ideas for integration of the conceptual modeling languages and those for user interface description include user modeling facilities.

1. INTRODUCTION

Although conceptual modeling techniques such as data, process, and rule modeling are being used to a large degree in developing application systems, these techniques are not alone very helpful when specifying the user interface. User interfaces of modern GUI-applications comprise much of the total code and the translation of conceptual models into run-time systems necessitates a large amount of manual coding. The most favored solution has so far been using a third party UIMS (User Interface Management System) or some kind of toolkit to implement the user interface.

User interface description techniques on the other hand, are seldom useful in the early modeling phases when one is modeling the functionality of the whole information system and not only the computerized solution. At this point, one should not be interested in technical details of a user interface to a computerized situation.

We will present an approach to user interface modeling that can be integrated with conceptual modeling techniques. Our approach is based on a dialogue modeling language

*Direct responses to this paper to , Babak Amin Farshchian mail: UNIT-NTH/IDT, N-7034 TRONDHEIM, NORWAY; email baf@idt.unit.no; phone: +47 73594427, fax: + 47 73594466.

which, together with a presentation language, helps the user to model and execute an early prototype of the user interface, which then can be improved, in parallel with the further development of the application system. The two parts are developed independently, and are integrated in an easy and practical way. This implies that the developers on each side do not need to rely on the others to do their job, and the final products are fully compatible. The advantages of this approach are numerous, as will be explained later.

The paper outlines the formalisms used in the UIMS which is a part of PPP (Processes, Phenomena, Programs) ICASE environment developed at the Norwegian Institute of Technology.

In Section , we will briefly describe the main aspects of the existing languages used in PPP, and how they are related. In Section the languages for user interface modeling are presented, whereas the mechanism used to integrate these languages and PPP's conceptual languages is described in Section . A comparison with related work is given in Section . Concluding remarks and future extensions are given in Section .

Throughout the paper, we will use an example to explain different aspects of our modeling approach. The example, which is actually part of our UIMS, is a graphical editor used to model UID_D (Dialogue description) part of the user interface. This version of the editor is a reduced statecharts editor.

A simple specification of the editor is given below.

> The editor is used to draw simple statecharts diagrams. Available symbols are states (in form of rounded rectangles), arrows (in form of splines with direction indicators), history, deep history, selection and condition (all in form of circles with different labels). A state is drawn using a rubber rectangle which stretches to follow the mouse cursor during drawing session. An arrow is drawn using a rubber spline. Other symbols are drawn without rubbers since they have predefined shape, i.e. circle.
>
> To draw symbols, there must exist a diagram. The user can create new diagrams by choosing *New Diagram* menu item in the file menu. The user can save a diagram by choosing *Save* from file menu, and providing a name for it. A diagram can be checked in using *Check in* menu item, which will invoke consistency checker to check the diagram before it is checked in to the repository. The user can also check out an existing diagram or load a file containing a saved diagram.
>
> A set of object manipulation commands is provided: Create, Move, Resize, Delete, Delete All and Properties. The *Properties* command will bring up a dialogue box, letting the user change some attributes of a group of symbols.

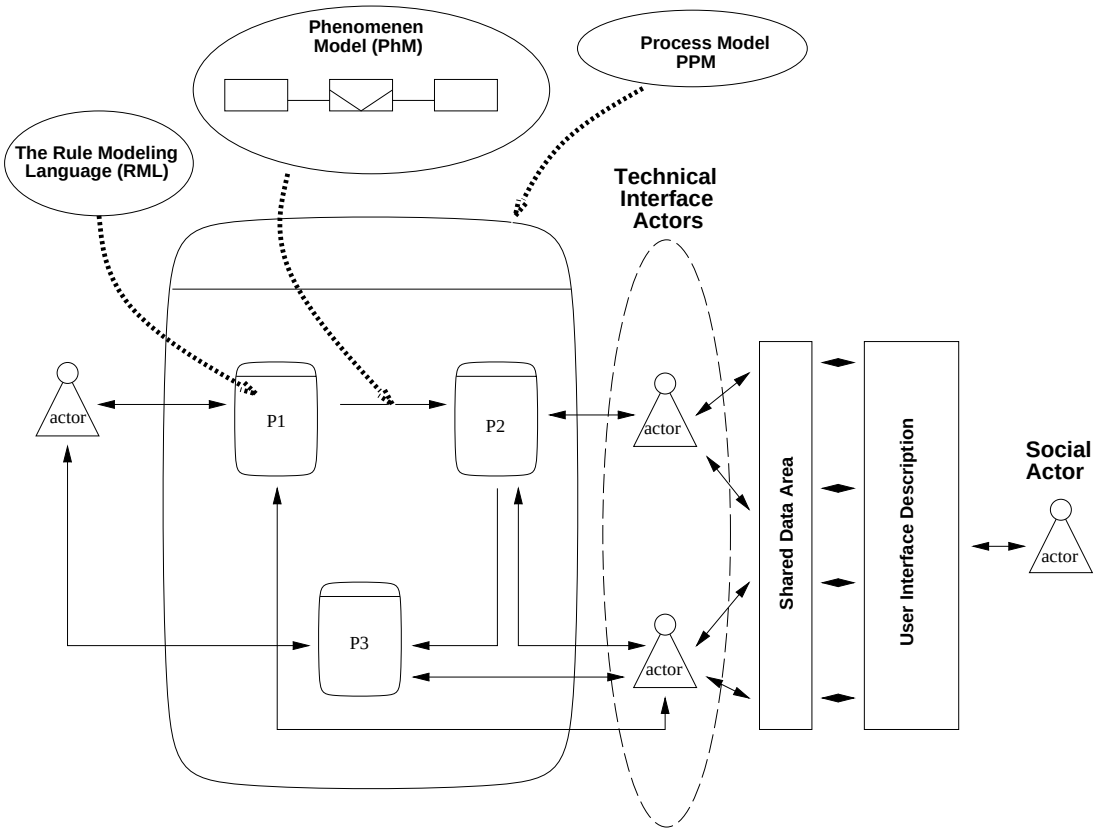

Figure 1: An overview of the PPP conceptual model and its connection to the user interface model.

2. CONCEPTUAL FRAMEWORK

The languages used in the PPP approach [GLW91, Yan93], extended with rule-modeling as specified in [Kro95b, KS94] and actor-modeling as specified in [Kro95a] constitute the current conceptual framework. Four interrelated modeling languages are used. Figure 1 shows how the resulting models are related to each other and the user interface model.

- PhM (Phenomenon model), a semantic data modeling language. An extension of PhM called ONE-R is currently used to specify entities and relationships among entities. To express properties of phenomena and operations on them, data types and methods are provided.
- PPM (Process Port Model), an extension of DFD including control flows in the SA/RT-tradition. A PPM is a network of processes, stores, timers, actors, and roles connected by flows. Processes might be decomposed in additional PPMs. Items, which are the objects being transported on flows can be described by a subset of the

ONE-R model.

- The actor modeling language including the modeling of both social and technical actors (see section) and roles.
- RML (rule modeling language). A rule is either a rule of necessity or a deontic rule. A rule of necessity is a rule that must always be satisfied. A deontic rule on the other hand is a rule which is (socially) agreed upon among several persons [WMW89]. RML contain explicit construct for querying the ONE-R model. Action rules are typically linked to processes in PPM, giving the execution semantics for the processes [KMOS91].

Integration of these languages is supported by an experimental CASE-tool [YSS95], where the repository structure is based on a meta-model description of the languages, and extensibility is further supported by the use of a meta-meta-model. Modeling techniques including consistency checking, model execution, explanation generation based on user-modeling, filtering, and code-generation are being supported.

3. USER INTERFACE DESCRIPTION

The User Interface Description (UID) tool is a UIMS designed to work with the PPP ICASE tool [Far95]. We have based our approach on the Seeheim model of user interfaces [Pfa85]. In our model, a user interface consists of a dialogue part (called UID_D, UID Dialogue description), a presentation part (called UID_P, UID Presentation description), and an interface to the application.

Statecharts [Har86] are used to describe overall dialogue and sequencing of user actions (The UID_D part). A look-and-feel independent, graphical notation is used to model the more static part of a user interface, i.e. presentation of layout (The UID_P part). The third notation is a constraint-based formalism used to model interactive objects from different domains[1]. These three parts are then combined into a user interface which communicates with the application using a data sharing method.

3.1. UID_P language

UID_P is a language for specifying screen components like forms, dialogue-boxes and other static elements in the user interface. There are numerous form generation tools available in the marked, designed specially for one or several windowing systems. Our goal was to design a tool which was as independent as possible from the underlying windowing system. This showed to be a problem since the set of available user interface components is quite

[1] This part is currently integrated in UID_D but is planned to be extracted into a separate tool.

different depending on the platform used. Attributes of a push-button are for instance different in OSF/Motif and MS-Windows.

This *look-and-feel dependency* problem is discussed in [ZM90], and a solution based on a textual language for specification of user interface presentations is given. The designer writes a specification of a dialogue unit by giving the name of each field in the unit, type of interaction technique to be used with the field, and relations between different fields. The graphics look-and-feel related to each platform is saved in a database along with style rules for the same platform. The rules are defined in a graphical editor by graphics designers and then stored in this database. When the textual description of presentation units is finished, a graphical layout is created automatically.

A similar approach is used to eliminate graphical dependencies in UID_P. The textual language discussed above is not so easy to use for non-experts. We have developed a graphical language based on the typically hierarchical structure of presentation units; a dialogue box can be visualized semantically as a tree of elements. The root of the tree is the dialogue box itself and the nodes are different elements used in the dialogue box. What we needed was a set of *semantical elements* which could construct any kind of screen components. The following set of semantical elements are used [ZM90]:

- *Single-choice:* The user is allowed to choose one item among a set of choices.
- *Multiple-choice:* The user can choose several items in a choice set.
- *Text:* An editable text input field.
- *Single-choice-with-text:* A single-choice element with a text field associated with it.
- *Multiple-choice-with-text:* A multiple-choice element with text fields associated with each field.
- *Command:* Used to select items from a menu.
- *Number-in-range:* A slider like number selection element.
- *View:* In some cases, a view of application data is needed in a dialogue box.
- *Static:* A field that shows a piece of static information like a text or a bitmap.

We have to emphasize that these elements are independent from the underlying windowing system, and actually independent from any interaction system. They are solely used to obtain information from the user. Other aspects of dialogue units, like placement of buttons on a dialogue box, are not included in this set of semantical elements and are not a part of our model. A WYSIWYG editor is used to alter the final, automatically generated dialogue units if this would be needed.

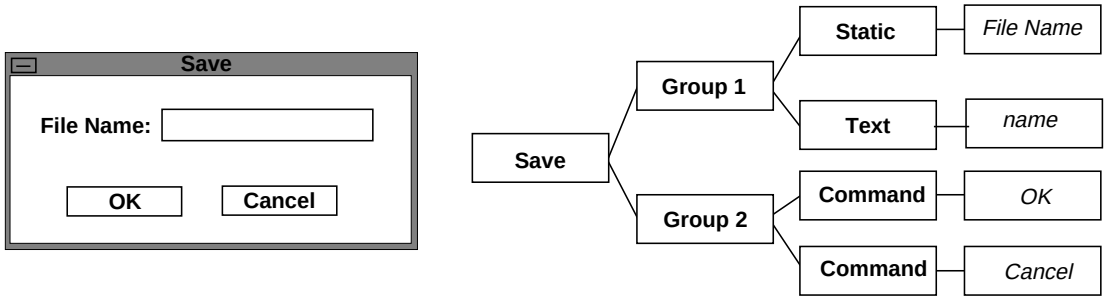

Figure 2: A simple save dialogue box modeled using UID_P

To let the developer use different facilities offered in different windowing systems, *attributes* can be attached to these elements. Each style-guide database will understand a set of attributes which can be used as some kind of specification for that particular windowing system, and will ignore other attributes. This makes it possible to customize the screen components for different windowing systems.

A simple save dialogue box is shown in figure 2. The hierarchy of the box is divided into two groups, where the first one has a static label (*File Name*) and a text input field, and the second one has two buttons (*OK, Cancel*).

3.2. UID_D language

One important issue regarding user dialogues is the notation used in the UIMS to model the dialogue. The notation employed in a UIMS defines the range of user interfaces that can be produced by that UIMS [Gre86]. Green defines two important properties for UIMS dialogue notations:

- *Descriptive power* is the set of user interfaces that can be described by the notation.
- *Usable power* is the set of user interfaces that can *easily* be described by the notation.

Amount of descriptive power can be specified by formal methods. Usable power cannot be specified formally, but has great importance in usability of a dialogue notation. A designer will not consider using a notation if she does not feel that the user interface she is developing is easy to describe by the notation.

A stand-alone UIMS is normally designed to satisfy the needs of user interface developers in a limited domain. In this case the descriptive power is of less importance; the important thing is whether the UIMS can be used easily to model and develop user interfaces in this particular domain. In a CASE environment, the descriptive power of the CASE tool will mirror the kind of user interfaces a UIMS related to it should be able to generate. The PPP

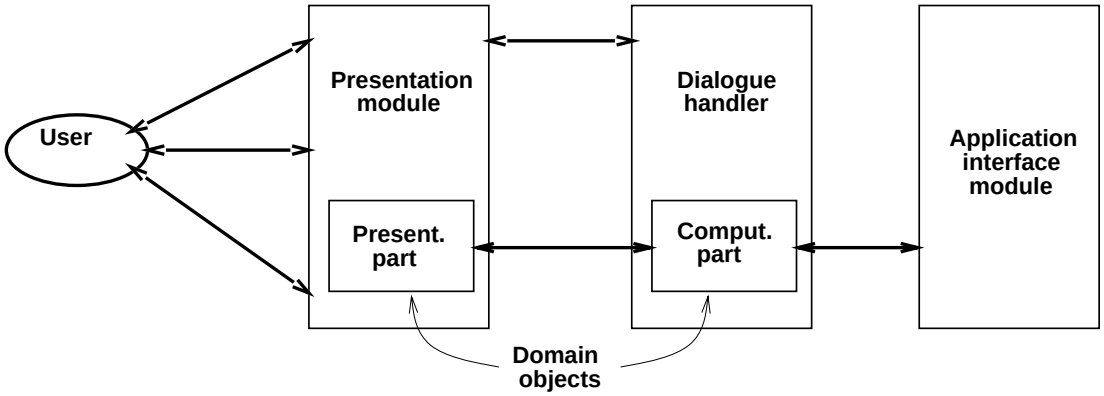

Figure 3: Seeheim model including our approach with two-leveled dialogue specification.

ICASE environment is based on structured analysis [GLW91] and is theoretically capable of modeling systems in any domain, transaction-based information systems being the main attraction.

We have developed a two-leveled approach for dealing with the problem of descriptive power, among other problems. The main idea is to use a more abstract, task-oriented way to model the dialogue part, dialogue meaning here the sequence of interaction between user and user interface. We attempt to divide the dialogue handler of Seeheim model into two parts, one for dealing with overall dialogue between the user and the user interface, and one for dealing with fine-grained, interactive part of the user interface. Figure 3 shows this division.

The first level of a user dialogue is modeled in an extensions of a specialized state transition modeling language called Statecharts [Har86, HLN+90]. A transition network contains a set of states which correspond to the modes of the user interface. Events are shown as arcs between these states. In its simplest form, arcs are marked with events and states are marked with actions. Arcs can also be attached to actions which in most cases reduces the number of states. In Statecharts, several states can be abstracted into a parent state in two forms: OR abstraction (OR-decomposition) where one of child states is activated upon entering the parent state and AND abstraction where all the children of a state are activated upon entering the parent state. For a detailed description of using Statecharts in modeling dialogues see [Far94, Far95, Raa93].

Figure 4 shows the UID_D model of our example. When the editor is running (*On* state), the user can execute different commands, which will move the editor to other states. A save command will, for instance, move the editor to *Save* state. In this state the user can do several operations on different objects, which are not apparent in the statecharts diagram, and she can also choose *OK* or *Cancel* to go back to the edit state.

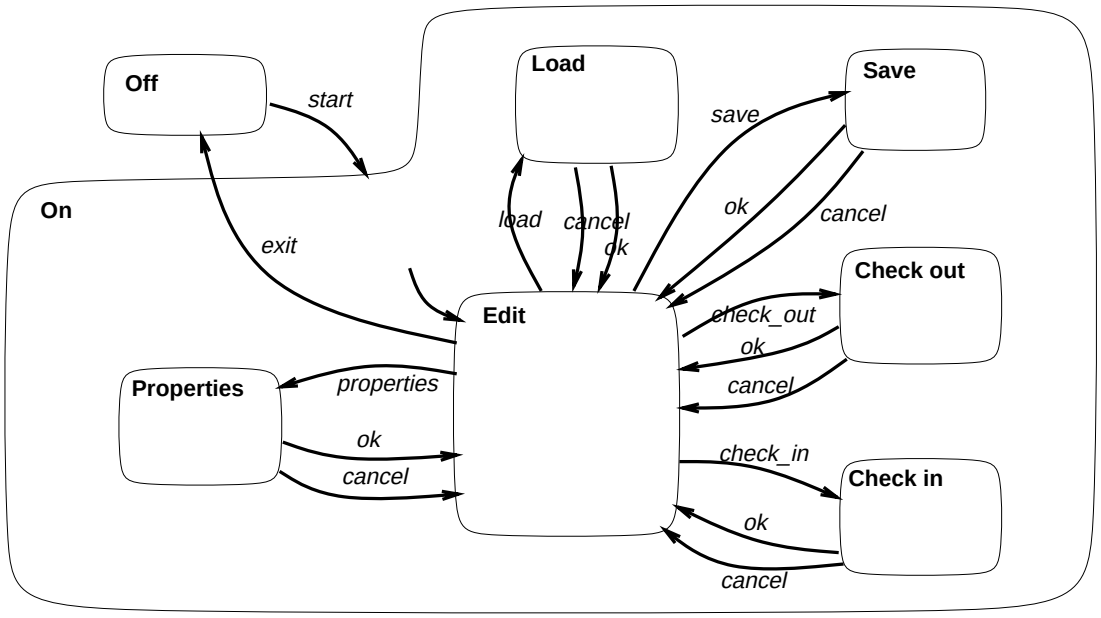

Figure 4: The example UID_D editor modeled in statecharts

We agree that a serious problem of transition networks is related to their limited descriptive and usable power [Gre86]. A dialogue specified as a transition diagram restricts the user to step through a predefined series of actions, a limitation which becomes apparent in direct manipulation user interfaces. Because of this restriction we have introduced a second level to our specification formalism, based on actors. We call these actors *Domain Objects*. These are interactive objects that communicate directly with the user and the information system, and are used to keep the user interface consistent with the application data. The user always manipulates some kind of domain object while interacting with the user interface, and the result of this manipulation is shown immediately to the user (direct manipulation).

A domain object consists of two separated parts that communicate with each other through a set of constraints. The *presentation* part is responsible for visualization of the internal state of the object, and to capture user actions. This part is normally kept in a library of visual objects with built in interaction mechanisms. The second part, called *computation* part, is responsible for communication with the application. This communication is done through a shared *data pool*. Each domain object can allocate an addressed area of this data pool, and can use this area to send and receive commands and data from the application. Each domain object has a parent window (here a statecharts state) which is responsible for broadcasting specific events to the object.

The *Domain Object Editor* is used to construct domain objects, This tool is mainly used in constructing direct manipulation user interfaces.

We can think of a domain object as a *view* of some portion of application data . Different users will need different views to the same data. Three characteristics of these views can be defined as [BRSS94]:

- *Focus* Each user interface needs only know a portion of the application data. The set of views will define which portion is needed for the user interface. This set is dynamic; views can be added or deleted from the set.
- *Presentation* Different users need different representation of the same data. One user may want a tabular representation of economical growth in the company while another user may need a graphical representation.
- *Position* Spatial arrangement of the views on the screen may give different information to different users. This arrangement may also need to be changed when focus or presentation are changed.

These properties of views give us an idea of how domain objects should be defined to provide us with consistent user interfaces.

We define three sets of constraints corresponding to the three properties of views. The first set, which we will call *data constraints*, deals with the consistency in the state of the view and the state of the portion of the application this view is representing.

The second set of constrains, *presentation constraints*, deals with the consistency between *computation* and *presentation* parts of views. Visual representations will be used to give visual feedback to the user when she performs operations like move, resize and reshape. This is similar to MVC (model-view-controller) of SmallTalk, but uses constraints instead of messages.

The third set of constraints, which we will call *position constraints*, will arrange the views in a user interface according to predefined rules. The end points of a link is, for example, moved when the independent object it is linked to is moved by the user. Figure 5 shows these three sets of constraints.

The domain object editor is designed to give a formalism for defining a computation and a presentation part, and specifying constraints mentioned above between these. So far we have been using a textual language for this purpose. One *template* is used for each part of a domain object. Regarding our example, a state is represented as a rounded rectangle, and a template for the presentation part of its object will be:

```
TEMPLATE RoundRect: PRESENTATION;
SHARED:
BEGIN
    STRING name;
```

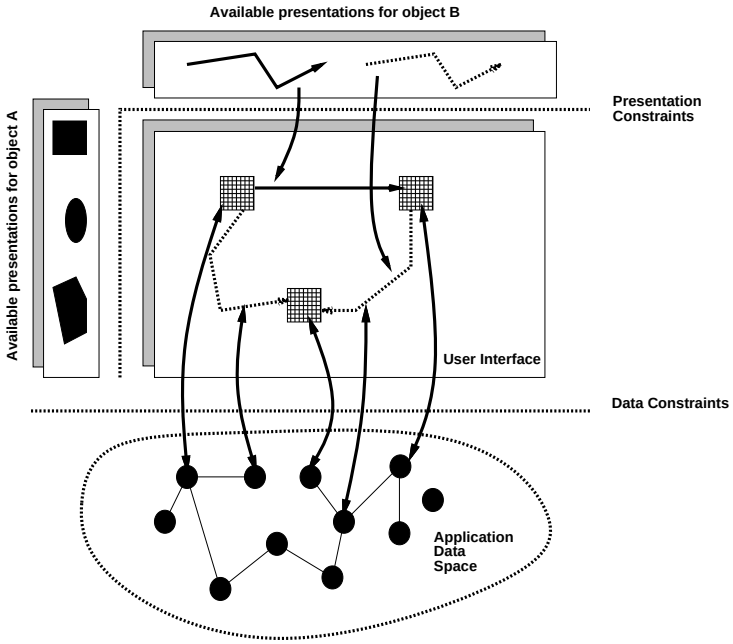

Figure 5: Different types of constraints in a user interface.

```
    POINT tl, br;
    INTEGER line_style, line_width;
    INTEGER colour, fill_pattern;
END;
BEHAVIOR:
BEGIN
    Resize, Move, Reshape, Delete;
END;
```

This template has a `SHARED` part where the data specified here is shared with the computation part. The `BEHAVIOR` part contains a list of operation this object is capable of performing.

The computation part of the same object will be:

```
TEMPLATE State: COMPUTATION;
SHARED: APPLICATION:
BEGIN
    STRING name;
    POINT tl, bl;
    INTEGER line_style;
```

```
    INTEGER colour, fill_pattern;
END;
SHARED: INTERFACE:
BEGIN
    STRING name;
    POINT tl, bl;
    INTEGER line_style, line_width;
    INTEGER colour, fill_pattern;
END;
```

This template has two fields for shared data, one for sharing with the user interface and one for sharing with the application. There is no behavior part needed here since the object is not visible for the user.

Constraints can now be defined between these two templates. These constraints correspond to presentation constraints mentioned above:

```
CONSTRAINT RoundRect<>State:
BEGIN
    RoundRect.name = State.name;
    RoundRect.tl = State.tl;
    RoundRect.bl = State.bl;
    RoundRect.line_style = State.line_style;
    RoundRect.line_width = State.line_width;
    RoundRect.colour = State.colour;
    RoundRect.fill_pattern = State.fill_pattern;
END;
```

In addition to the constraints, mechanisms for creation and deletion of objects are offered by the editor.

This two-leveled method of specifying user dialogues has some advantages which are important for our work. One is that the tool can be used for prototyping user interfaces without having any actual application present. The developer can begin specifying Statecharts diagrams for the user interface even before the designers of the application have begun their work on specifying the information system. The Statecharts model of the user interface is completely independent from the application and can be visualized using the UID_P language. This constructs a complete user interface which can be tested by the users.

Another advantage, which is extremely important in the context of CASE-tool environment UID is a part of, is the ability of specifying a diversity of user interfaces, ranging from user interfaces to transaction-based and business information systems, to graphical, direct

manipulation user interfaces. A user interface consisting of forms and documents can easily be realized using Statecharts and UID_P alone, without involving domain objects. On the other hand, the power of domain objects bring the possibility of specifying highly interactive graphical user interfaces with intensive interaction with the user.

The fact that a user interface can completely be constructed using a simple Statecharts diagram in conjunction with a set of UID_P diagrams, has a great impact in reducing the amount of work depending on what kind of user interface one is constructing. Usually, the developer must employ complicated mechanisms even when developing simple user interfaces. In UID, the user uses only the parts of the formalism needed for the actual user interface. Two extremes cases can be a user interface with dialogues based only on Statecharts, and a user interface based on domain objects[2].

The user interfaces developed using UID tool are, as mentioned above, technically independent from the underlying application. The only interaction with the application is via a shared data pool, which is defined to the outside world by a set of services. This means, ideally a user interface can be plugged to any application providing the services needed by the user interface. One future activity related to the UID project is to define a protocol for this layer which will act as a high level API.

4. INTEGRATION

The main integration mechanism used between UID and the PPP conceptual modeling languages is achieved through modeling a link between the UID and the PPM. There are several types of links that can be applied:

- Fields in the UID_P can be connected to types in ONE-R to ensure consistent types of data.
- Actions linked to events can be linked to triggers, i.e. clicking OK in a popup dialogue box activates a process in the PrM model. A more detailed user interface independent description of how this is done can be specified.[3].

[2]One Statecharts diagram is always needed to execute the interface, but this diagram need not be more complicated that a single state without any events.

[3]The indicated result is returned to the user interface, and taken care of by the internal storage structure of the interface. Our experience with this indicates that one sometimes needs to rearrange the process models doing this. This is not surprising, since one at this stage will be modeling a different situation than when using the process modeling language to model a not necessarily *computerized* information system. In most cases additional process modeling needed in connection to the user interface can be modeled in separate process models. Some might argue that one can avoid the above mismatch by using object-oriented modeling both for the conceptual modeling and the user interface modeling, but we think that also there similar problems will appear, since one is in effect modeling different things, and should thus expect some mismatch [Dav95, HS93]

- Technical actors that are to support modeled processes, can be used as a connection link between domain objects and the conceptual model.

Our main focus in the integration has been on the last method, where technical agents are used to communicate between the user interface and the application. The method is as follows: Two sets of services, one for the application and one the user interface are defined in a suitable stage of development. These sets are required to configure a consistent interface between the user interface and the application where the user interface knows what to expect from the application, and the application knows what kind of information the interface may and will provide to it. When these sets of services are defined, the development is continued. After finishing the conceptual model of the application, extra *interfacing actors* are inserted into the model. These actors are responsible for updating the application depending on the events happening in the user interface, and for providing necessary information to the user interface. User of the system, i.e. external entities, are modeled as social actors or roles.

5. RELATED WORK

UIMSs have been developed for a variety of user interface types. As mentioned in the introduction, most of these are addressing a specific domain.

StateMate [HLN+90] is an advanced prototyping tool that uses statecharts and a form generation tool to prototype information systems. One has tools for modeling, analysis, and execution of complete interactive systems. The user interface prototyping is based on statecharts and has some of the limitations related to state transition diagrams.

Rendezvous [HBR+94] is a UIMS designed to develop multiuser interfaces. The idea is to use *shared abstractions* and *views* to model respectively applications and user interfaces. These two are then linked to each other. A view can be compound, consisting of parts of several abstractions. Rendezvous is maybe the existing system that uses an approach most similar to ours. The difference is that in Rendezvous, every thing is based on objects and links between them. This makes it somehow difficult to obtain an overview of the system when the amount of object types and instances are increased, which is common for user interfaces.[4] Our approach offers a combination of an abstraction mechanism (statecharts) which functions in a top-down manner, and the power of object-orientation in a bottom-up manner.

[4]This is actually a general problem of the object-oriented methodology. There is a lack of abstraction and categorization in the systems using large amounts of objects.

6. CONCLUSION AND FUTURE WORK

We have in this paper presented an approach for the integrated modeling of user interfaces and information systems. Although they are integrated, it is also possible to use the modeling languages separately.

The main advantage of our approach is the great descriptive and usable power of the formalism. Using statecharts of UID_D in conjunction with domain objects and UID_P, one can construct a great variety of user interfaces, from form-based transaction systems to interactive graphical editors.

Work is being done to extend the formalism to multiuser interfaces. The approach will be to isolate the statecharts model of the user interface from the application, and to let the user interface communication with the user interface through domain objects. We can then construct views of the application for each user interface, opening for collaboration aware applications.

The UIMS is in implementation phase. The first part is planned to be implemented in 1995. This includes UID_P and a simple statecharts editor. An interpreter for this part is also under development. A domain object editor will be implemented in a future project.

Another area of further work in the approach is that of implementing the integration of the frameworks within the PPP-toolset, so that UID will be able to use similar tools to perform validation through larger case-studies. Further work in the approach will be to investigate the inclusion of user-modeling facilities. Currently, the explanation generation tool applies user-modeling. An integration of this with the actor-modeling language is proposed in Krogstie [Kro95a], and a natural extension is to investigate the application of this for the generation of adaptive user interfaces as part of the generation of computerized information systems from the combined conceptual models and user interface description. Some preliminary work has already been done in this direction [Ols93], but this needs to be extended.

References

[ABS91] R. Andersen, J. A. Bubenko jr., and A. Sølvberg, editors. *Proceedings of the Third International Conference on Advanced Information Systems Engineering (CAiSE'91)*, Trondheim, Norway, May 1991. Springer-Verlag.

[BRSS94] Richard Bentley, Tom Rodden, Pete Sawyer, and Ian Sommerville. Architectural Support for Cooperative Multiuser Interfaces. *IEEE Computer*, May 1994.

[Dav95] A. M. Davis. Object-oriented requirements to object-oriented design: An easy transition? *Journal of Sytsems and Software*, 30(1/2):151–159, July/August 1995.

[Far94] Babak Amin Farshchian. Construction of the User Interface Description Language. Technical report, Institute of Electrical Engineering and Computer Science, Norwegian Institute of Technology., 1994.

[Far95] Babak Amin Farshchian. UID, An Integrated User Interface Management System for the PPP ICASE Tool. Master's thesis, Norwegian Institute of Technology, April 1995.

[GLW91] J. A. Gulla, O. I. Lindland, and G. Willumsen. PPP - An integrated CASE environment. In Andersen et al. [ABS91], pages 194–221.

[Gre86] Mark Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, July 1986.

[Har86] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8.1987*, 1986.

[HBR+94] Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F Patterson, and Wayne Wilner. The Rendezvous Architecture and Language for Constructing Multiuser Application. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125, June 1994.

[HLN+90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[HS93] G. M. Høydalsvik and G. Sindre. On the purpose of object-oriented analysis. In A. Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, pages 240–255. ACM Press, September 1993.

[KMOS91] J. Krogstie, P. McBrien, R. Owens, and A. H. Seltveit. Information systems development using a combination of process and rule based approaches. In Andersen et al. [ABS91], pages 319–335.

[Kro95a] J. Krogstie. *Conceptual Modeling for Computerized Information Systems Support in Organizations*. PhD thesis, IDT, NTH, Trondheim, Norway, 1995. Forthcomming.

[Kro95b] J. Krogstie. Goal-oriented modeling of information systems. In *Proceedings of the Seventh International Conference on Computing and Information (ICCI'95)*, pages 983–1007, Peterborough, Canada, July 5–8 1995.

[KS94] J. Krogstie and G. Sindre. Extending a temporal rule language with deontic operators. In *Proceedings from the 6th International Conference on Software Engineering and Knowledge Engineering (SEKE'94)*, pages 314–321. IEEE, June 21-23 1994.

[Ols93] K. O. Olsen. User-modeling in ppp. Technical report, IDT, NTH, Trondheim, Norway, 1993.

[Pfa85] G. Pfaff, editor. *User Interface Management Systems*. Springer-Verlag, 1985.

[Raa93] M. R. Raabel. User interface specification language for PPP. Master's thesis, IDT, NTH, Trondheim, Norway, 1993.

[WMW89] R. J. Wieringa, J-J. C. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.

[Yan93] M. Yang. *COMIS - A Conceptual Model for Information Systems*. PhD thesis, IDT, NTH, Trondheim, Norway, 1993.

[YSS95] M. Yang, A. H. Seltveit, and A. Sølvberg. A distributed repository management system for formal conceptual IS models. In G. Grosz, editor, *Proceedings of the Sixth Workshop on The Next Generation of CASE Tools*, pages 195–213, Jyväskylä, Finland, 1995.

[ZM90] Brad Vander Zanden and Brad A. Myers. Automatic Look-and-Feel Independent Dialog Creation for Graphical User Interfaces. In Jane Carrasco Chew and John Whiteside, editors, *CHI '90 Conference Proceedings*, pages 27–34. ACM, Addison-Wesley, 1990.