Model-Based Automated Generation of User Interfaces

Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen

Medical Computer Science Group
Knowledge Systems Laboratory
Departments of Medicine and Computer Science
Stanford University
Stanford, CA, 94305-5479
{puerta,eriksson,gennari,musen}@camis.stanford.edu

ABSTRACT[1]

User interface design and development for knowledge-based systems and most other types of applications is a resource-consuming activity. Thus, many attempts have been made to automate, to certain degrees, the construction of user interfaces. Current tools for automated design of user interfaces are able to generate the static layout of an interface from the application's data model using an intelligent program that applies design rules. These tools, however, are not capable of generating the dynamic behavior of the interface, which must be specified programmatically, and which constitutes most of the effort of interface construction. Mecano is a model-based user-interface development environment that uses a domain model to generate both the static layout and the dynamic behavior of an interface. A knowledge-based system applies sets of dialog design and layout rules to produce interfaces from the domain model. Mecano has been used successfully to completely generate the layout and the dynamic behavior of relatively large and complex, domain-specific, form- and graph-based interfaces for applications in medicine and several other domains.

INTRODUCTION

In recent years there has been significant progress in providing automated assistance to user-interface developers. Commercially available interface builders, user-interface management systems, and interface toolkits provide considerable savings to developers in time and in effort needed to produce a new interface (deBaar, Foley, & Mullet 1992).

Even with these commercial tools present, the amount of effort and low-level detail involved in constructing interfaces is substantial. Therefore, researchers are investigating techniques to automate more portions of the interface design process. One promising area is that of model-based user-interface development (Puerta 1993; Szekely, Luo, & Neches 1993). In this approach, developers work with high-level specifications (models) of the interface to define dialog and layout characteristics. Model-based systems facilitate the automation of interface design tasks. A successful approach has been to use the application's data model to generate the static layout of an interface (deBaar, Foley, & Mullet 1992; Janssen, Weisbecker, & Ziegler 1993).

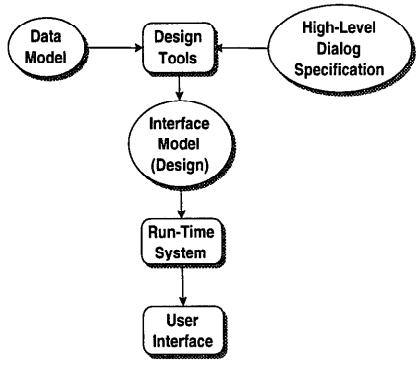

Figure 1. Generic framework for automated interface-generation environments that employ data models. The interface design is produced by tools that examine a data model and a dialog specification. The design may be represented implicitly or explicitly (as an interface model). The run-time system implements the design.

Figure 1 shows a generic framework for automated interface generation environments that employ data models. An intelligent design tool examines the data model and applies a set of design rules to produce a static design of an interface. Because the data model is shared between the interface design and the target application design, both designs can be coupled, and changes to the application design can be propagated easily to the interface design. The dynamic behavior of the interface, however, must be specified separately. This process can take many forms such as using a graphical editor to construct dialog Petri nets (Janssen, Weisbecker, & Ziegler 1993), to assigning sets of pre- and postconditions to each interface object (Gieskens & Foley 1992). Although working with

1. This work has been supported in part by grants LM05157 and LM05305 from the National Library of Medicine, and by gifts from Digital Equipment Corporation. Dr. Musen is recipient of NSF Young Investigator Award IRI-9257578.

high-level dialog specifications is helpful to interface developers, it does not automate the design of dynamic behavior. For large interfaces, editing the dialog specifications is still a time-consuming task involving the definition of hundreds of actions and conditions, some of which may conflict with each other.

The Mecano Approach

Current data-model approaches do not exploit the relationships among objects in the model to generate the dynamic behavior of an interface. In addition, a data model is application-specific. In the Mecano approach, we aim to use domain models from which dynamic interface behavior can be generated, and that are also sharable across a range of applications.

In this paper, we present Mecano, a model-based interface development environment that uses domain models instead of data models to generate interfaces. A domain model is a high-level knowledge representation that captures all the definitions and relationships of a given application domain. A domain model extends the data model for the application. By substituting the data model in Figure 1 for a domain model, Mecano does not require any dialog specification editing and can generate complete dynamic behavior specifications even for large interfaces with hundreds of components.

The rest of this paper is organized as follows. We first review related work and present an overview of Mecano, including a definition and illustration of domain models. Then, we show how various cases of dynamic behavior can be generated from domain models by using an example from the medical domain. Next, we explain how end users are able to participate in the layout design of interfaces generated in Mecano and how design revisions can be conducted. We conclude by analyzing this approach and summarizing the results.

RELATED WORK

There are three types of systems documented in literature that relate closely to the Mecano approach: (1) systems that use textual specifications to generate dialogs, (2) systems that combine the use of data models and high-level dialog specifications, and (3) systems that directly manipulate an interface model to produce an interface.

One of the earliest efforts to generate dialogs via textual descriptions is COUSIN (Hayes and Szekely 1992). It generates menus and fill-in forms from a specification of commands and their parameters. Mickey (Olsen 1989) uses an extended version of Pascal to describe contents, parameters, and behavior of direct-manipulation of interfaces. ITS (Wiecha et al. 1989) separates dialog and style into two different layers and allows the specification of the dialog layer through a command language and the definition of styles through a rule set. Given the textual description for a dialog, ITS reasons about the style rule set to generate the interface. The UofA* (Singh and Green 1991) system generates the presentation and dialog through a command language. These systems, in general, help the developer by providing tools to design dialogs at a high-level of abstraction, but they do not automate the design process beyond that point.

Among the first examples of the use of data models to derive static layouts for interfaces is HIGGENS (Hudson and King 1986). It allows a developer to view abstractly the interface by examining the data models, but it lacks an automatic generator for the actual interface.

The UIDE environment includes a tool for static layout generation from an extended data model (deBaar, Foley, & Mullet 1992). The specification of dynamic behavior, however, must be achieved by defining sets of pre- and postconditions (Gieskens and Foley 1992) for each one of the interface objects. The GENIUS environment (Janssen, Weisbecker, & Ziegler 1993) uses an entity–relationship data model along with a graphical editor for dialog specifications to generate interfaces. The data model, which can be edited graphically, provides the basis for the definition of the interface components and their layout. The graphical editor allows the review of *dialog nets,* a variation of Petri nets, that define the actions of the interface objects and the conditions that preclude or follow those actions.

Systems that employ data models have the advantage of sharing the data model with the target application, thus coupling the design of both. They cannot automate dynamic dialog design from the data model and have problems scaling up because of their approach to specifying dialogs. For example, the use of pre- and postconditions in large interfaces can cause conflicts among the conditions and may necessitate the development of conflict-resolution strategies.

Systems that generate interfaces by manipulating interface models include HUMANOID (Szekely, Luo, & Neches 1993) and DON (Kim and Foley 1993). HUMANOID defines an elaborate interface model that includes components for the application, the presentation, and the dialog. Developers construct application models and HUMANOID picks among a number of *templates* of interfaces to display the interface. The developer can then refine the behavior of the interface by editing the dialog model. HUMANOID assists, but does not automate, the generation of dynamic behavior specifications, and requires considerable additional developer effort to generate interfaces that do not conform to its templates, as is the case with most complex interfaces. DON uses a presentation model that allows developers to explore designs and that provides expert assistance in the generation of designs. DON does not have a dynamic behavior component for automatic generation of dialogs.

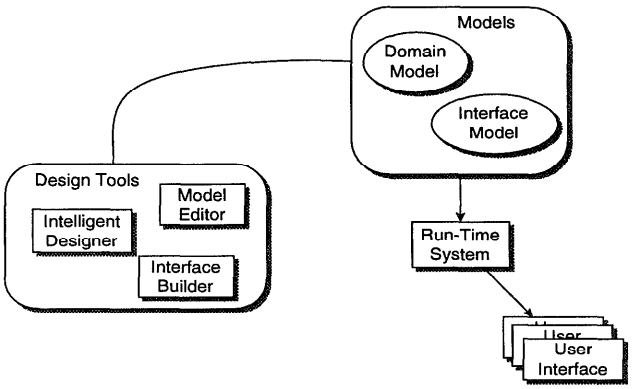

Figure 2. The main components of Mecano. The intelligent designer operates on a domain model, as opposed to a data model, to produce interface designs.

OVERVIEW OF MECANO

The main components of the Mecano environment are shown in Figure 2. Mecano follows the general architecture of Figure 1, replacing the data model with a domain model. The design tools include a model-editing tool, an intelligent designer tool, and an interface builder, which in our case is provided by the supporting platform, the NeXT environment.

The framework for user-interface development with Mecano calls for a developer to start by employing the model editor (Gennari 1993) to visualize and review a domain model (described later in this paper). The domain model is shared with the target application. Therefore, an interface developer need not build one for a given domain from scratch. Instead, the normal process is to revise an existing one. Once a domain model is deemed satisfactory, it is input to the intelligent designer (Eriksson, Puerta, & Musen 1994), a tool that produces a dynamic dialog specification and a preliminary layout for the interface. The layout can then be refined using NeXT's Interface Builder. Both the dialog and layout output by the intelligent designer are stored declaratively in an interface model. This model contains all facets of an interface design including interface objects, presentation, dialog, and behavior.

The design defined in the interface model is implemented by a run-time system. Mecano's run-time tools have the capability of implementing form- and graph-based interfaces with many types of objects, from simple ones, such as menus and push-buttons, to complex ones, such as list browsers and domain-specific graphical editors. The run-time tools implement the dynamic behavior of the interface according to the specifications in the interface model.

The overall design process in Mecano is iterative. The resulting interfaces may have deficiencies that require editing the domain model and regenerating the interface. In such cases, the intelligent designer keeps track of layout customizations that may have been made in the previous generation and reapplies these customizations as appropriate.

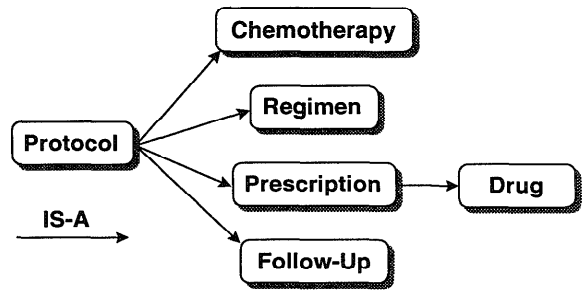

Figure 3. Partial view of a medical domain model for therapy (protocol) administration (IS-A view). The hierarchy of classes is used to generate the interface-navigation schema for windows and other objects.

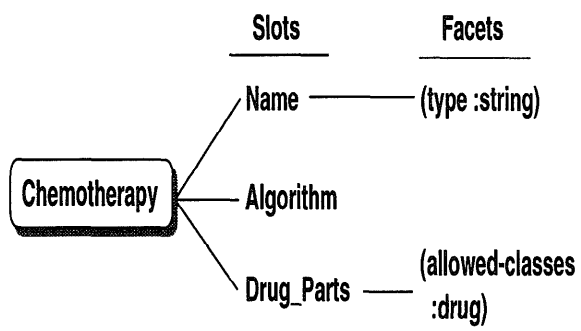

Figure 4. Partial view of the slots and facets (properties) for the *chemotherapy* class. Facets can define *allowed-classes* relationships among classes. These relationships are used to generate specifications for interface-object groupings in windows. Other facets like *type* are important to determine static layout (e.g., appropriate widget for a type *string* object)

Domain Models

A domain model is a representation of the objects in a domain and their interrelationships. Domain models in Mecano are constructed using a frame-based representation language that defines class hierarchies (Gennari 1993). Each *class* in the hierarchy can have a number of *slots* and each slot defines a number of properties (called *facets*) Figures 3 and 4 show partial views of a model for the medical domain of therapy administration (called protocol administration).

There are two important relationships in domain models. The *is-a* relationship (see Figure 3) determines the class hierarchy and is used by the intelligent designer in Mecano to specify the interface-navigation schema among windows and other objects. The *part-of* relationship (see Figure 4) is

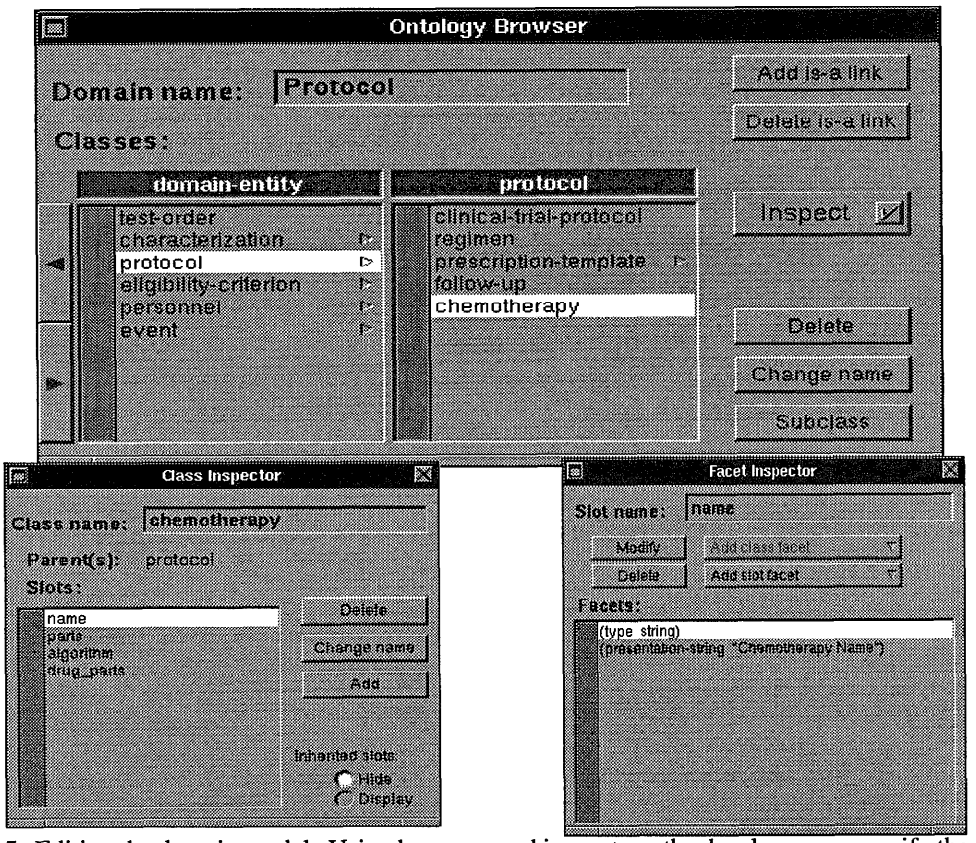

Figure 5. Editing the domain model. Using browsers and inspectors, the developer can specify the *class hierarchy* (top window) and the *slots* and *facets* (properties) of each class.

used to determine object groupings by windows. Other important facets include, for example, *type*, *cardinality*, *min* and *max* of a slot, which are used in the specification of the static layout (e.g., what widget should be used for the slot; size of a numeric input field). In fact, the application's data model is completely included in the domain model. Therefore, all the design rules of an intelligent design tool that may be applied to a data model can be applied to a domain model. In the next section, we illustrate the use of domain models to generate a therapy administration application.

GENERATION OF DIALOG SPECIFICATIONS FROM DOMAIN MODELS

Before dialogs can be generated, a domain model must be prepared with the model editor shown in Figure 5. The domain model is shared with the target application. Thus, a coupling of application design and interface design is established. Developers can build domain models incrementally, and can prototype interfaces early in the development process because Mecano supports iterative design. More importantly, it is not necessary to build domain models from scratch for every application. A domain model for medical therapy planning can be reused, with minor variations, in other applications. This is a significant advantage of Mecano over systems that design from data models because data models are difficult to reuse across applications.

Once edited, the domain model is used to generate dialog specifications. These specifications have two levels in Mecano:.

- High-level dialog defines all interface windows, assigns interface objects to windows, and specifies the navigation schema among windows in the interface.

- Low-level dialog defines specific dialog elements (widgets) to each interface object created at the high level and specifies how the standard behavior of the dialog element is modified for the given domain.

High-Level Dialog Generation

The elements of the high-level dialog specification are generated by examining the class hierarchy of the domain

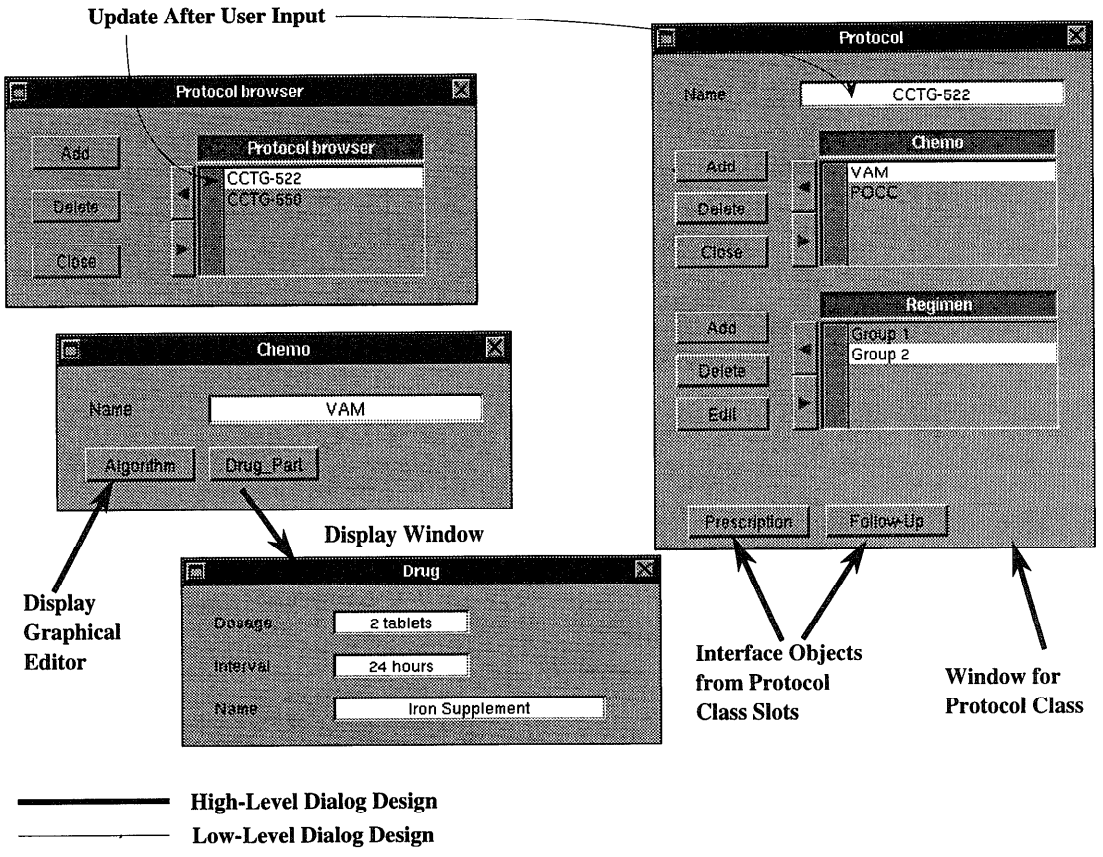

Figure 6. Interface generated from the partial domain model in Figures 3 and 4. Legends indicate generated dialog at high- and low-level design times. An interface generated from the full domain model for medical therapy contains over 60 windows and hundreds of dialog elements (widgets). The dynamic behavior of such interface can be generated automatically from a domain model.

model (see Figure 3) and the slots of each class (see Figure 4). Figure 6 shows an interface generated from the partial domain model shown in Figures 3, and 4. The complete medical domain model for therapy administration generates an interface with over 60 windows and hundreds of widgets. Note that the dialog for window navigation is established during high-level dialog design but that it can be refined, or augmented, at low-level dialog design time. The procedure to generate a high-level dialog design is as follows:

- Each class in the hierarchy is assigned a window.
- Window navigation is established by searching the class hierarchy for links indicated by the *allowed-classes* facet in the domain model. For example, the *Drug* window shown in Figure 6 is accessed from the *Chemo* window because the Drug class is an allowed class for the slot *Drug_Part*.
- Each window is assigned one *interface object* per slot in the class. After generation, the developer has the option of customizing the interface by splitting windows multiple objects into two or more windows. Interface objects are assigned actual widgets during low-level dialog design.

Low-Level Dialog Generation

Elements of the low-level dialog specification are generated by examining the facets (properties) defined for each slot in the domain model (see Figure 4). The process has these steps:

- Each interface object defined at high-level design time is assigned a dialog element (widget) by examining the facets of the corresponding slot in the domain model. For example an object of *type string* is assigned a text field, an object of *type Boolean* is assigned a check-box widget, and an object of *type*

string and *cardinality multiple* (i.e., the object can be multiple-valued) is assigned a list browser.

• Each dialog element may be assigned *actions* beyond the standard behavior of the dialog element by examining the facets of the corresponding slot in the domain model. Examples of dialog-element actions include disabling editing in other dialog elements, and updating values in other dialog elements after a user input action (see Figure 6).

Note that the specification of dialog-element actions is one of the important operations that cannot be automated in systems that rely on data models for interface generation.

GENERATION OF DOMAIN-SPECIFIC GRAPHICAL EDITORS

One of the important capabilities in Mecano is the generation from domain models of domain-specific, nodes-and-links graphical editors useful to describe procedures such as flowcharts. Consider the following slot information for the class *Protocol*:

(slot algorithm
(type :procedure)
(allowed-classes :xrt :chemotherapy :drug))

When the intelligent dialog designer examines this slot during low-level dialog design, it assigns a graphical editor as the dialog element for that slot due to the type *procedure* defined for that slot. It also defines three graphical objects to be used during editing, one for *x-ray therapies* (xrt), one for *chemotherapies (chemo),* and one for *drugs.* Figure 7 shows a graphical editor generated from the above slot definition.

PARTICIPATORY LAYOUT DESIGN AND DESIGN REVISION

A crucial concern with any system that automatically generates interfaces is how it allows the developer to review and change the generated design. In Mecano, there are two types of revisions: layout and dialog.

The intelligent designer tool uses a layout algorithm to produce a *preliminary* layout of the interface objects. The philosophy in Mecano is to be able to involve the end user in the process of custom-tailoring a layout. For example, for the medical treatment application shown in this paper, the interface developer works together with a physician to review and custom tailor the preliminary layout with an interface builder (see Figure 2). Our experience is that this revision—in the case of the interface derived from the full model—may take between two and a half to four hours for the 65 windows included in that application (including layout and dialog revisions, and needed interface regenerations). Custom-tailoring information is kept on a database so that if the interface needs to be regenerated because of incremental changes to the domain model (as it

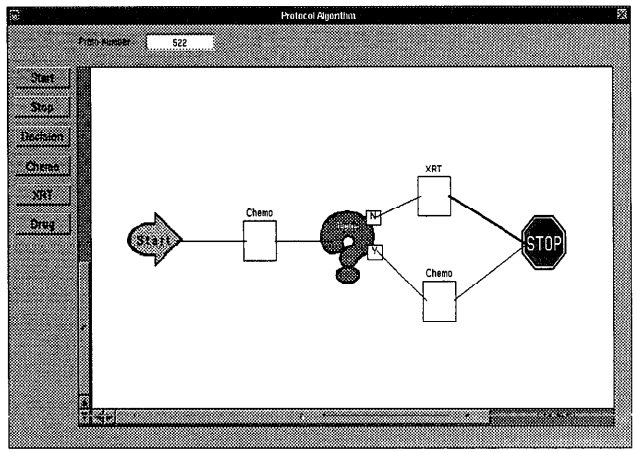

Figure 7. A graphical editor to draw medical treatments generated from a domain model. Both the drawing objects and their connectivity behavior are determined by the intelligent designer tool in Mecano.

is often the case), the customizations can be reapplied to the newly generated interface. Substantial revisions of the domain model, however, invalidate the information on the customization database.

The working sessions with the end user—in this paper's example, a physician—are also used to discover difficulties with the dialog design and incompleteness in the information displayed in the interface. Dialog design customizations can be made by editing directly the interface model (see Figure 2) and do not require a regeneration of the interface. On the other hand, for the interface to be able to display additional dialog elements, changes must be made to the domain model to define needed slots or classes. Such changes do require the interface be regenerated. Overall, the Mecano policy is to understand the interface design process as iterative and to support the introduction of custom changes without creating duplicate work.

ANALYSIS AND CONCLUSIONS

We have described a user-interface development environment that generates automatically presentation and dialog specifications for domain-specific, form- and graph-based interfaces. The strong points of this system are:

• Generation of both the static layout and the dynamic behavior of domain-specific, form- and graph-based interfaces, including relatively large and complex ones, for multiple domains (e.g., medical treatment, elevator configuration).

• Use of the application's domain models, which includes the application's data model, for interface

generation considerably augments automation capabilities over systems utilizing only a data model.

- Support of participatory layout design involving end users of the applications, and support for iterative design without duplication of work.

Mecano has the same central weakness that other model-based systems have: the system is as good as the expressiveness of its underlying models. We continue researching extensions to our frame-based representation language for domain models and interface models in order to be able to automate more types of dialog actions. In particular, we are concerned with how to generate complex sequences of actions (commands) at low-level dialog design time. We are also working on the run-time system of Mecano to implement new types of widgets. Furthermore, the interface generation approach from domain models is most useful for domain-specific interfaces with a relatively fixed user dialogue (such as the medical forms shown in the figures in this paper). For other types of interfaces, it will be necessary to examine other types of models (such as a model of the user's task) to be able to generate automatically interface specifications. We are currently working on developing such task models as components of our generic interface model.

Overall, Mecano provides a framework for assisting the development of interfaces and for the study of interface models and the relationships between domain characteristics and user interface presentation and dialog.

ACKNOWLEDGMENTS

We wish to thank Tom Gruber for his helpful comments.

REFERENCES

de Baar, D.J.M.J., Foley, J.D. and Mullet, K.E. 1992. Coupling Application Design and User Interface Design. In *Proceedings of Human Factors in Computing Systems, CHI'92*. Monterey, California, May 1992, pp. 259–266.

Eriksson, H., Puerta, A.R. and Musen, M.A. 1994. Generation of Knowledge-Acquisition Tools from Domain Ontologies. In *Proceedings of the Eighth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop.* Banff, Alberta, Canada. pp. 7.1–7.20.

Gennari, J.H. 1993. *A Brief Guide to Maître and MODEL: An Ontology Editor and a Frame-Based Knowledge Representation Language.* Stanford University, Knowledge Systems Laboratory, Report KSL-93-46, Stanford, California. June 1993.

Gieskens, D.F. and Foley, J.D. 1992. Controlling User Interface Objects through Pre- and Postconditions. In *Proceedings of Human Factors in Computing Systems, CHI'92*. Monterey, California, May 1992, pp. 189–194.

Hayes, P. and Szekely, P. 1992. Graceful Interaction through the {COUSIN} Command Interface. *International Journal of Man–Machine Studies*, **19**(3), pp. 285–305.

Hudson, S.E. and King, R. 1986 A Generator of Direct Manipulation Office Systems. *ACM Transactions on Information Systems,* **4**(2), pp. 132–163.

Janssen, C., Weisbecker A. and Ziegler J. 1993. Generating User Interfaces from Data Models and dialog Net Specifications. In *Proceedings of Human Factors in Computing Systems, INTERCHI'93.* Amsterdam, The Netherlands, April 1993, pp. 418–423.

Kim, W.C. and Foley, J.D. 1993. Providing High-Level Control and Expert Assistance in the User Interface Presentation Design. In *Proceedings of Human Factors in Computing Systems, INTERCHI'93.* Amsterdam, The Netherlands, April 1993, pp. 430–437.

Olsen, D.R. 1989. A Programming Language Basis for User Interface Management. In *Proceedings of Human Factors in Computing Systems, CHI'89.* Austin, Texas, May 1989, pp. 171–176.

Puerta A.R. 1993. The Study of Models of Intelligent Interfaces. In Proceedings of the 1993 International Workshop on Intelligent User Interfaces. Orlando, Florida, January 1993, pp. 71–80.

Singh, G. and Green, M. 1991. Automating the Lexical and Syntactic Design of Graphical User Interfaces: The UofA* UIMS. *ACM Transactions on Graphics,* **10**(3), pp. 213–254.

Szekely, P., Luo, P. and Neches, R. 1993. Beyond Interface Builders: Model-Based Interface Tools. In *Proceedings of Human Factors in Computing Systems, INTERCHI'93.* Amsterdam, The Netherlands, April 1993, pp. 383–390.

Wiecha, C., Bennett, W., Boies, S., Gould, J. and Greene, S. 1989. ITS: A Tool for Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems,* **8**(3), pp. 204–236.