Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2002), *16*, 173–184. Printed in the USA.
Copyright © 2002 Cambridge University Press 0890-0604/02 $12.50
DOI: 10.1017.S0890060402163049

Creation of flexible graphical user interfaces through model composition

B. RAPHAEL,[1] G. BHATNAGAR,[2] AND I.F.C. SMITH[1]
[1]IMAC, Applied Computing and Mechanics Laboratory, Swiss Federal Institute of Technology, (EPFL), Lausanne, Switzerland
[2]Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi 110 016, India

(RECEIVED September 27, 2001; ACCEPTED March 28, 2002)

Abstract

Nearly all software products have rigid and predefined interfaces. Users are usually unable to modify or customize features beyond cosmetic aspects. Interface adaptability is important because aspects such as user preferences and task sequences vary widely in engineering, even within specialized domains. A methodology for the creation of adaptable user interfaces using model composition is presented in this paper. User interfaces are generated dynamically through the composition of model fragments that are stored in a fragment library. When fragments are linked to models of physical behavior, interface model composition applications are likely to be easier to extend and maintain than traditional graphical user interfaces. A prototype system within the domain of bridge diagnosis illustrates the potential for practical applications.

Keywords: Graphical User Interfaces; Model Composition; Model-Based User Interfaces

1. INTRODUCTION

The field of user interface (UI) design has been identified as one of the six core subjects of computer science (Hartmanis & Lin, 1992). Moreover, it is considered to be the most critical for the success of applications within companies (Grover & Goslar, 1993). Well-designed interfaces improve the performance of users and thus result in significant cost savings. However, UIs are hard to design. The evaluation of quality inevitably involves subjective matters such as appearance and ease of use. Several guidelines and methodologies have been published for good UIs (Schneiderman, 1997). For example, Smith and Mosier (1986) compiled 944 guidelines in a 478 page report.

There is a general consensus that the design of UIs should account for specific characteristics of expected users. This is a challenge, even in specialized domains such as engineering, because users often manifest a wide range of preferences and approaches when confronted with similar tasks. Interface adaptability is important for software systems to be effective across a wide variety of users (Boulanger & Smith, 2001). Most software packages have rigid and predefined interfaces that make it impossible to customize features beyond cosmetic aspects. Navigational structures, as well as the look and feel, are usually hard coded into programs. Programs that permit users to change the UI allow only a choice of predefined sets of alternatives. For example, once familiarity is reached, users might be able to shift from "novice" mode to "expert" mode.

Computer tools that support complex tasks are often difficult to use. Several techniques, such as the use of intelligent assistants and wizards, have been attempted to support complex tasks. These attempts have not always provided the support that was intended (see Section 2). The objective of this work is to improve the effectiveness of computer tools that support engineering tasks through adaptable UIs. More specifically, we developed an approach to adapt the UIs of decision support systems. The approach is based on model composition. The graphical UI (GUI) is constructed dynamically through explicit representations of parts of the GUI in the form of model fragments.

Difficulties in developing and evaluating GUIs in general, and engineering applications in particular are discussed in the next section. This is followed in Section 3 by

Reprint requests to: B. Raphael, IMAC–Applied Computing and Mechanics Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland. E-mail: Benny.Raphael@epfl.ch a description of the use of model composition techniques for the development of a system that supports bridge diagnosis tasks. The paper ends with a discussion of limitations of the approach.

2. DESIGN AND IMPLEMENTATION OF UIs

2.1. Simplifying interfaces for complex tasks

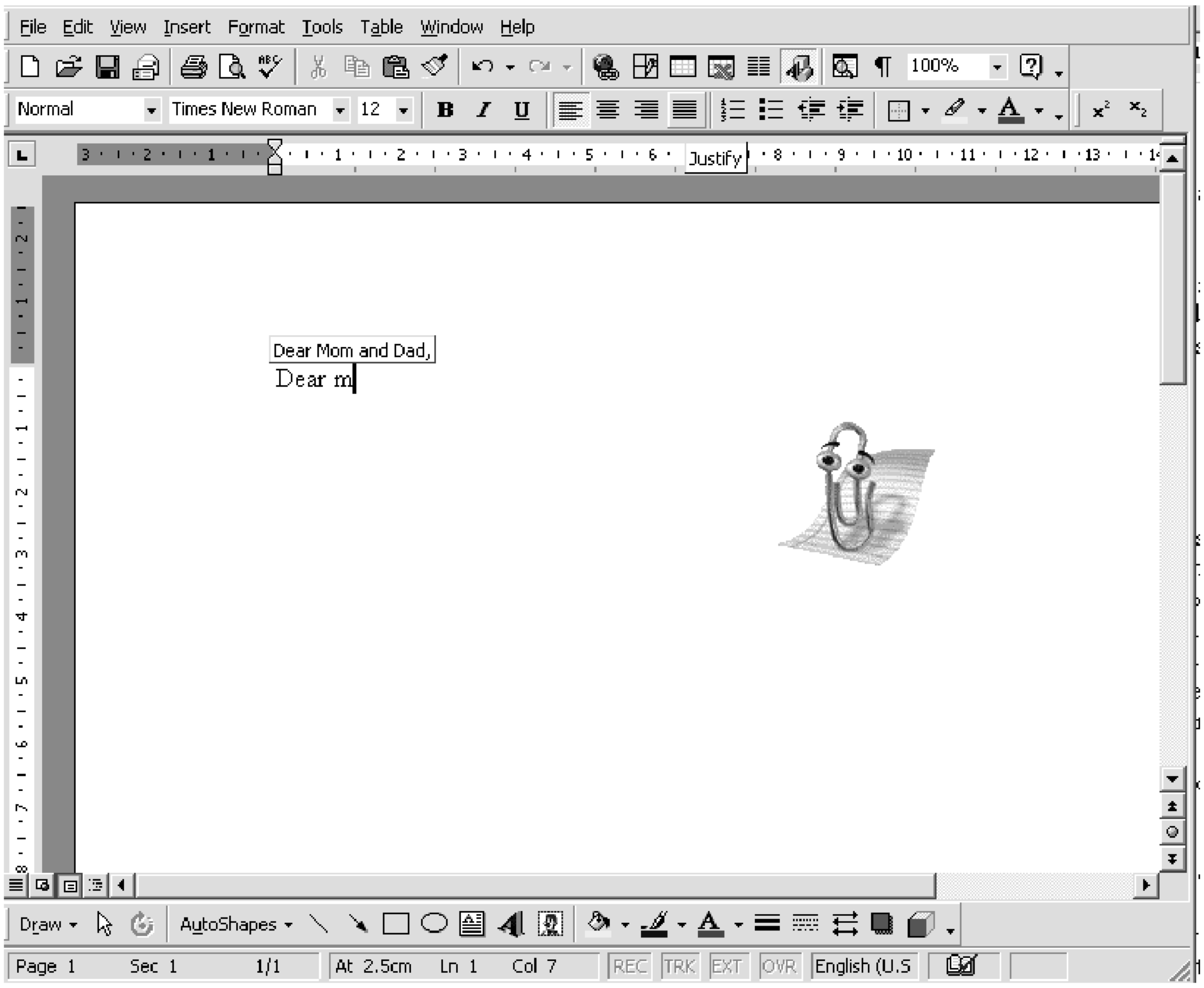
The main objective of UI research is to improve support for tasks. Current UIs do not provide appropriate support (Colvin, 2001), even after decades of work and huge investments. For example, modern word processors contain office assistants, which are meant to provide proactive support. However, some users have reported that suggestions are irrelevant and the overall behavior is intrusive. For example, as soon as users start typing in “Dear m,” a word processor may suggest the phrase, “Dear Mom and Dad” (Figure 1). The application often makes incorrect assumptions regarding user characteristics. All users typing “Dear m” are assumed to be children who are writing letters to their parents. Incorrect assumptions lead the assistant to interfere with the activities of users. Furthermore, queries posed to the assistant often retrieve irrelevant topics.

This example illustrates the following:

- several software companies have begun to realize the importance of proactive support;
- proactive support in the absence of accurate knowledge is likely to be more irritating than helpful; and
- commercial technology has not matured enough to provide intelligent support.

This leads to the following questions:

- Should engineering software attempt to guess the intentions of users and provide suggestions?


File Edit View Insert Format Tools Table Window Help
100%
Normal
Times New Roman
12
Justify
Draw
AutoShapes
Page 1 Sec 1 1/1 At 2.5cm Ln 1 Col 7 REC TRK EXT OVR English (U.S

Dear Mom and Dad,

Dear m

Fig. 1. An example of a word processing assistant. Proactive support often involves wrong assumptions about user characteristics.

- Is it possible to make reliable assumptions about user characteristics?
- What do engineers expect from UIs?

Here, a difference is drawn between proactive support and active support. While proactive support involves presumptions regarding the task of the user, active support (Smith, 1996) provides only updated information related to aspects of tasks such as the characteristics of current solution spaces. Active support is intended to provide help to engineers prior to decision making; assumptions related to user characteristics are not made. Such support is generally preferable to passive support where, for example, decisions are critiqued. Our experience is that practicing engineers often react negatively to passive support and positively to active support.

2.2. UI design

Myers (1993) describes challenges related to UI design and implementation, including

- lack of knowledge of characteristics of tasks and users;
- complexity of tasks;
- trade-offs between standards, presentation quality, and performance; and
- difficulty carrying out iterative design due to time and cost constraints.

Users are extremely diverse. It is often a challenge to formulate generalizations related to the way people carry out tasks. Interfaces should match the skills, expectations, and needs of users. Another complication is that the modularity and structure of non-UI code are often incompatible with requirements for good human–computer interaction (HCI). For example, highly modular programs may contain black box functions. Each function produces an output for a given input using a well-defined algorithm. If users need to determine the status of computation at regular intervals, the relevant function needs to contain code to communicate with the user interface. This clutters the code with operations that are not related to the main objective of the function, and it violates the principle of performing one well-defined operation per function.

The complexity of the task is usually comparable to the complexity of UI design. For example, a reasonable word processing application contains more than 300 different operations. Allowing users to access all operations while avoiding an excessively complex interface requires careful design.

The UI design also requires consideration of code maintenance and updating. The model–view–controller (MVC) paradigm (Buschmann et al., 1996) is useful for application development because it leads to increased reliability during code maintenance. According to this architecture (Figure 2), the application is split into three separate layers that interact with each other using simple interfaces. The *model* encapsulates application logic. It contains the structure of data and the logic for data manipulations. The *view* is concerned with displaying the model accurately to the user. Displays take the form of text and graphics. The *control* is responsible for interacting with the user in order to modify the model. Instead of modifying the model directly, the control sends messages to the model in order to carry out these changes. The model sends messages to the view to refresh the presentation. The advantage of MVC is that if the presentation platform is changed (e.g., from Visual Basic to Java), code that is within the application layer need not be modified.

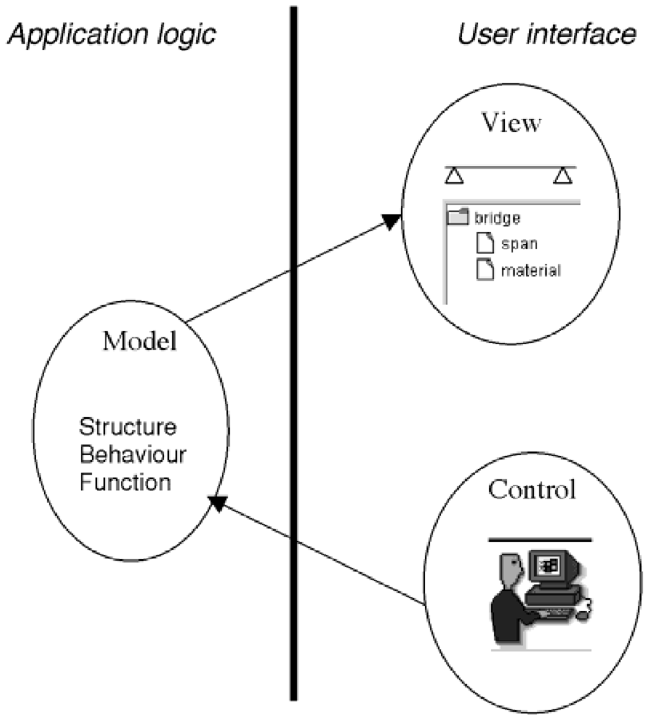


Fig. 2. The model–view–control (MVC) paradigm improves maintenance reliability. The model contains the representation of domain knowledge, which is used to generate views for users. Users interact with the model through the control module.

The MVC paradigm provides insights into the complexity of UIs. Objects in the application logic layer and objects in the presentation layer (user interface) are linked. Hence, the presentation layer is as complex as the structure of the data model. In addition, any modification to data is effected by sending messages from the presentation layer to the application logic layer. Therefore, there must be as many message passing links between the two layers as is required for all possible changes to the model.

2.3. UI design for engineering applications

Guidelines that have been generated through UI research are not necessarily applicable to engineering software because of unique characteristics of engineering users (Smith, 1999; Stalker & Smith, 1999). These characteristics include the following:

- Users are professionally qualified to perform the task that is being supported.
- Engineers are legally responsible for their actions, and they need to know the details of all key operations that are performed by computers. UIs should be capable of providing the necessary details.
- Engineers employ mathematical constructs for describing physical artifacts and processes. Engineers in each discipline use specialized graphical and symbolic languages to communicate without ambiguity. Although such information may be daunting to other users, engineering software should use this language in its UIs.
- Engineers are usually comfortable with intensive computer use as a result of their engineering education.

A challenge of UI research is that evaluations are often subjective. They involve qualitative matters such as appearance and ease of use. Although these aspects may be evaluated scientifically through, for example, statistical analysis of results obtained from questionnaires, results are subject to different interpretations. Nevertheless, the following consensus seems to have emerged for engineering applications:

- UI design should include a consideration of engineering characteristics;
- interfaces need to be adaptable;
- there needs to be a clear separation between presentation and data, even though data structures need to be compatible with interface characteristics.

Accepting these principles makes it possible to evaluate engineering UIs using less subjective criteria. For example, it is possible to analyze and determine whether a particular UI has accounted for engineering characteristics and to what degree it is adaptable. In this paper, the principal focus is on developing adaptable GUIs.

2.4. Need for developing adaptable interfaces

The strong relationship between data and process models and UIs was discussed in Section 2.2. Most UI elements are provided to create data, modify data parts, process data, and view results. These links between the UI and data models do not create difficulties for applications when data structures are well defined. This is not the case in many engineering applications. The structure of engineering objects are difficult to define *a priori*. Furthermore, they may change during the useful life of the software. Data model complexity creates challenges for UI development. The methods used to present the data to users, as well as the sequence of operations performed to modify the data, are context dependent. In the MVC paradigm, the view layer, which visually represents the data model, and the control layer, which modifies the data, are dynamic.

2.5. Model-based GUIs

A methodology for the creation of adaptable UIs is to make use of explicit models. The idea of model-based UIs is not new. Proposals have been made for more than a decade (Wiecha & Boies, 1990), and most originate from research into UI management systems (UIMS, Myers, 1995). UIMS help users write specifications of UIs in a specialized high-level specification language. These are automatically translated into executable programs or interpreted at run time to generate appropriate interfaces. Their main goal is to allow interface designers and end users to design and modify the interface quickly without requiring extensive programming skills.

The specification languages of UIMS require detailed descriptions of the format, placement, and functionality of UI elements (widgets). The technology for model-based UI development (Foley et al., 1989), on the other hand, allows developers to provide a higher level description of the objects and functionality of an application. An example of a system utilizing this approach is HUMANOID (Szekely et al., 1992). HUMANOID's design model captures information related to an application's functionality in five dimensions:

- application semantics design represents the operations and objects,
- presentation defines the visual appearance of the interface,
- manipulation represents gestures that can be applied to the objects,
- sequencing defines the order in which manipulations are assembled, and
- action side effects declares actions that are performed automatically after a manipulation.

The UI is generated automatically using this information.

Another approach to model-based UI development involves the automatic generation of UIs from declarative definitions of task and user models. Task models describe possible tasks that users perform during interaction with the application (Pinheiro da Silva, 2000). User models provide a way to model UI preferences for specific users or groups of users. Most systems do not have explicit user models; instead, they contain options for user preferences for a group of users.

Paterno and Mancini (2000) use an automatic tool-supported notation called ConcurTaskTrees (Puerta et al., 1999) for specifying task models in a hierarchical way. Task models were created by interviewing HCI experts and possible end users. Users were classified into different types depending on their knowledge in the domain, and navigation structures were designed accordingly.

Recently, there has been considerable interest in the application of Unified Modeling Language (UML) to interface design (Van Harmelen et al., 1997). UML is the current industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. Activities, sequences, components, and states can be represented diagrammatically using this language. Tools already exist for developing and editing UML models (e.g., http://www.rational.com). It is expected that commercial software packages will soon be available that generate UI code using UML models as input.

3. ADAPTIVE UIs THROUGH MODEL COMPOSITION

Compositional modeling (Falkenhainer & Forbus, 1991) is a framework for constructing adequate device models by composing model fragments selected from a model fragment library. Model fragments describe components and physical phenomena. Model construction is a search task in which the goal is to select a model from the space of possible models defined by the model fragment library while satisfying all constraints. The research in model composition concentrates on providing causal explanations for the behavior of small devices. Its application to construction of GUIs for full-scale engineering applications has not been assessed previously. In this work, model composition is used to dynamically create and instantiate models of UIs.

The following steps are involved in the creation of a GUI using model composition (Figure 3):

- Users define model fragments that represent parts of the GUI. Each fragment represents a particular aspect of the GUI such as the view for a data item or the control for allowing users to interact with it.
- Fragments are organized into one or more fragment libraries.
- The model composition module constructs a model of the UI by composing compatible fragments.

Model fragments involve a decomposition hierarchy consisting of attributes and sub-attributes. The structure is similar to the decomposition of objects in object oriented programming. The MVC paradigm (Section 2.2) is used to organize the information related to the GUI. Users specify the view and the control for each fragment. The view refers to the visual representation of a fragment. Views are defined in terms of standard UI elements such as text boxes, tables, buttons, and images. The control refers to the modules that manage the user interaction for each fragment. For example, a particular control module checks if the user input is within the allowable range.

Users are able to adapt the UI by adding new fragments and modifying the visual representations of attributes of fragments. The controls that are displayed in each window and the sequencing of windows are adapted through modifying the decomposition hierarchy.

This approach is particularly attractive to engineering applications that contain models of physical phenomena. Fragments of models that represent physical phenomena may be directly linked to fragments of the UI. This avoids the gradual degradation of the UI as models evolve through changes. The correspondence between model fragments of physical phenomena and fragments of the GUI is illustrated in Figure 4.

4. BRIDGE DIAGNOSIS USING MULTIPLE MODELS

The framework for creating adaptive UIs through model composition has been applied to bridge diagnosis decision support in order to illustrate the approach described in the previous section. Model-based bridge diagnosis involves finding causes for observations through reference to declarative models. Thousands of models may be possible, even for simple bridges. Identifying a good model is a nontrivial task. Multiple models can be created by making different combinations of assumptions related to material properties, geometric properties, support conditions, and loading. The case study of the Lutrive Highway Bridge in Switzerland illustrates this point.

4.1. Lutrive Bridge case study

The Lutrive Highway Bridge (Figure 5) was constructed in 1972 using the cantilever method with central hinges. Two

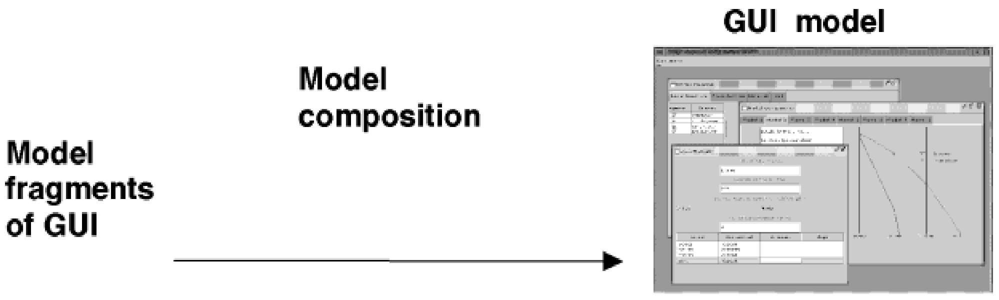


Fig. 3. The construction of a graphical user interface using model composition.

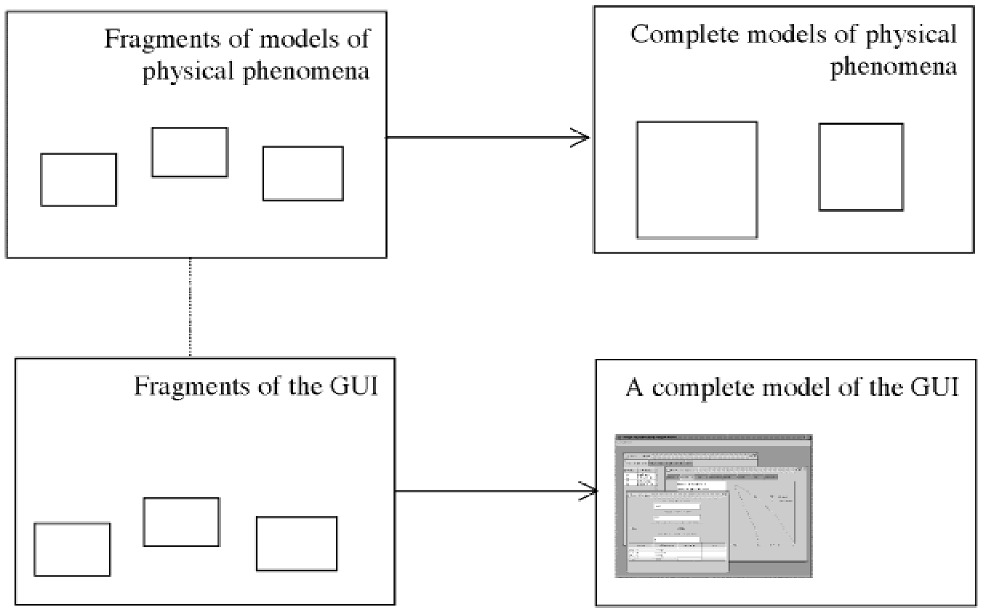

Fragments of models of physical phenomena
Complete models of physical phenomena
Fragments of the GUI
A complete model of the GUI

Fig. 4. The link between fragments of the GUI and fragments of models of physical phenomena.

bridges, the North and South Lutrive Bridges, were built (one for each direction of traffic) with a length of 395 m each and a maximum span of approximately 130 m. Both bridges were constructed with the same material and technology. However, one bridge continues to deflect considerably, even after 25 years of construction. The deflection is on the order of several centimeters (Robert-Nicoud et al., 2000). In order to understand the cause of abnormal deflection it is important to identify models that reasonably explain observations and measurements.

Examples of complete models that have been created for the Lutrive Bridge (Robert-Nicoud et al., 2000) are shown in Figure 6. Each model has a different set of attributes and values. This information is not completely known *a priori*.

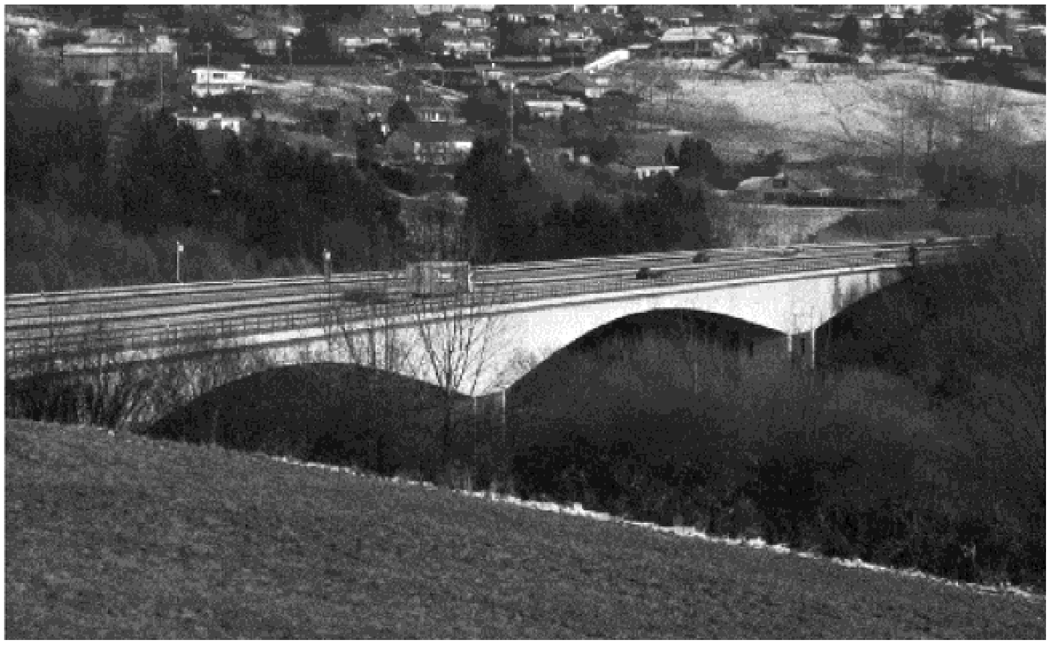


Fig. 5. The Lutrive Bridge in Switzerland.

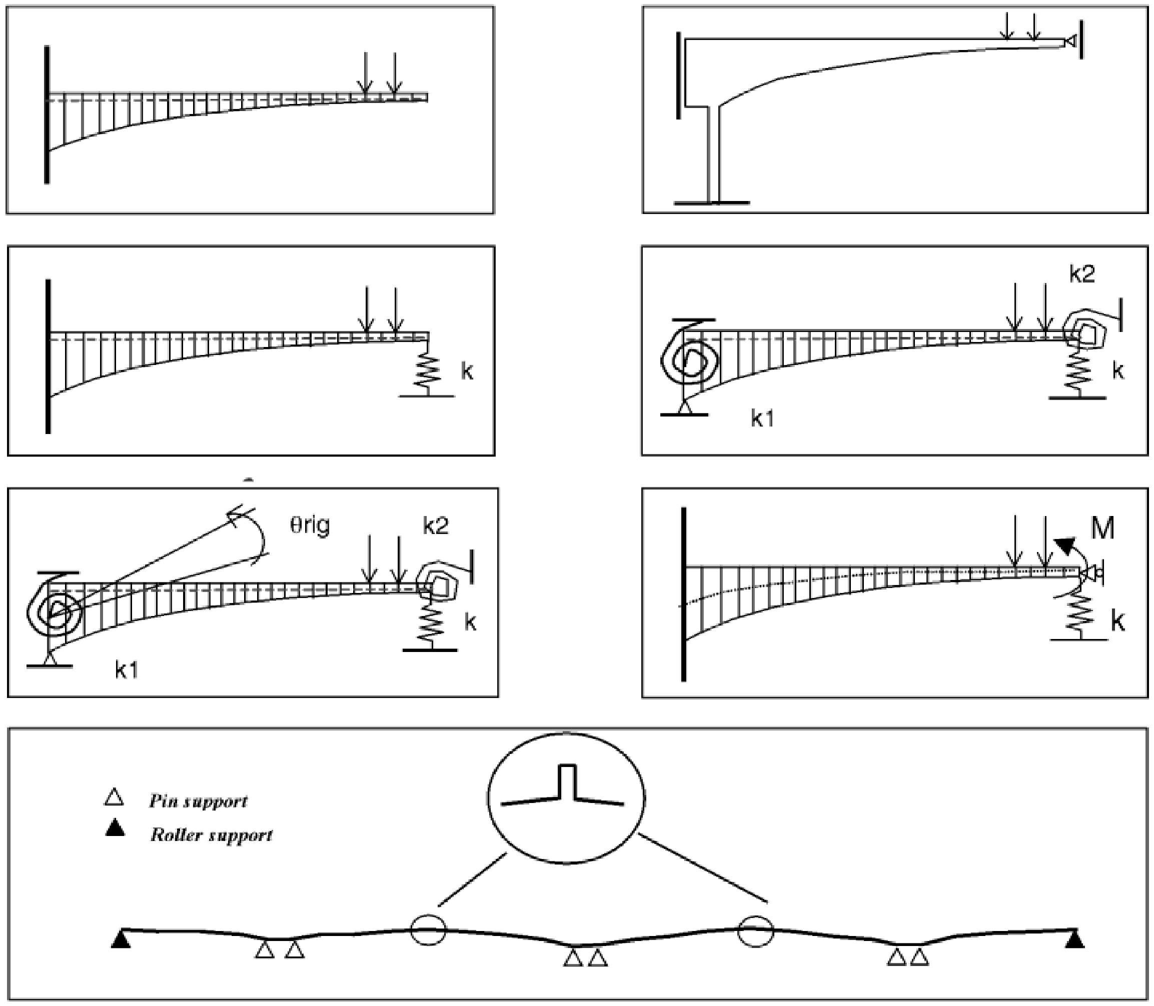


Fig. 6. Multiple ways of modeling the same bridge. Each model has different sets of attributes; hence, a GUI cannot be built using an assumed predefined data structure (Robert-Nicoud et al., 2000).

4.2. Model composition for bridge diagnosis

The technique of model composition has already been suggested for systematic evaluations of modeling possibilities for bridges (Raphael & Smith, 1998). Complete models are constructed by composing fragments that model different aspects of the bridge. Model composition in general is intractable because of the exponentially large number of combinations of fragments. Finding good combinations requires good user interaction. Well-designed UIs are necessary to provide active support.

As mentioned earlier, a difficulty associated with this application is that the data structure is dynamic. Users add model fragments and models at run time; therefore, the structure of data is not known *a priori*. Most programs operate on fixed data structures, but this is not possible here. Because the UI elements such as screens, buttons, and text boxes depend on the number and type of fragments in the model fragment library, a static predefined UI cannot be developed.

4.3. A model-based UI for bridge diagnosis

A model-based UI was developed for the bridge diagnosis support system that is described in the previous section. The program helps users create multiple models by repeatedly creating sets of feasible fragments from a model fragment library. These models are then analyzed using external structural analysis programs, and the results are compared with data collected from on-site measurements. This comparison results in the identification of good behavioral models.

4.3.1. The scenario

Engineers start with simple models in order to explain observations, and only when simple models fail to provide satisfactory explanations do they consider greater complexity. For example, in the case of Lutrive Bridge, a simple cantilever model (shown on the top left of Figure 6) was initially used. When measurement data did not match theoretical predictions, more complex models were explored.

Whereas simple models could be evaluated using algebraic expressions and simple procedures, more complex models require external programs such as those that provide finite element analyses. It is necessary to evaluate several models that are derived from different assumptions. This is done by composing model fragments that involve a unique set of assumptions. Model fragments are grouped into four categories (libraries) that represent different aspects of the bridge. The four libraries are basic structure, cross section, material, and load.

Three categories of personnel are involved in bridge diagnosis using this system.

1. The programmer defines general classes of model fragments and develops code for interfacing them with analysis modules.
2. The engineer responsible for knowledge formulation and maintenance defines specific model fragments that are applicable to specific bridges.
3. The engineer responsible for bridge monitoring inputs measurement data, selects model fragments, creates complete models, and evaluates results.

Four top level tasks have been identified: data management, model creation, model use, and model evaluation. For illustration, consider the task of model creation. Model creation involves selecting appropriate fragments from the fragment libraries and instantiating them. The sequence of windows (screens) that are shown to the user for instantiating each fragment is different. For a fragment called "uniform cross section," the user needs to input a single set of cross section properties. For another fragment, "parabolic variation of cross section," the user needs to input the cross section properties at three different points in order to completely define the longitudinal profile of the bridge. Users interact differently with these two fragments. The visual representations of the attributes of the fragments are also not similar.

The fragment libraries are currently implemented as text files describing the attributes and structure of individual fragments (decomposition hierarchy). Although the libraries storing model fragments of bridges and model fragments of the GUI are different, they are linked. Model fragment libraries of the GUI contain the decomposition and visual representation of each fragment. Model fragment libraries of the bridge are more complex than those associated with the GUI; they contain information related to the compatibility of fragments. (For more details, see Raphael & Smith, 1998.)

In general, the sequence of windows for each task and the controls that are displayed in each window are dependent on the model fragments that are currently present in the fragment libraries, as well as the fragments that are currently selected. The engineer responsible for knowledge maintenance defines model fragments and their visual representations.

Consider the initial stage of diagnosis in which only a single fragment, namely, the cantilever, was defined in the fragment library "Basic structure." At this stage, the table on the left side of Figure 7 contains one row displaying this fragment. Upon selecting this fragment, its attributes are shown on the right panel. The representation chosen for this fragment consists of two text boxes and two radio buttons. Relevant parts of the fragment library of the GUI representing this fragment follow:

```
Fragment cantilever {
  Attribute starting_position {
    Description "starting_position"
    View textbox
  }
  Attribute span_length {
    Description "starting_position"
    View textbox
  }
  Attribute support_location {
    Description "support_location"
    View checkbox
    Options { "left", "right" }
  }
}
```

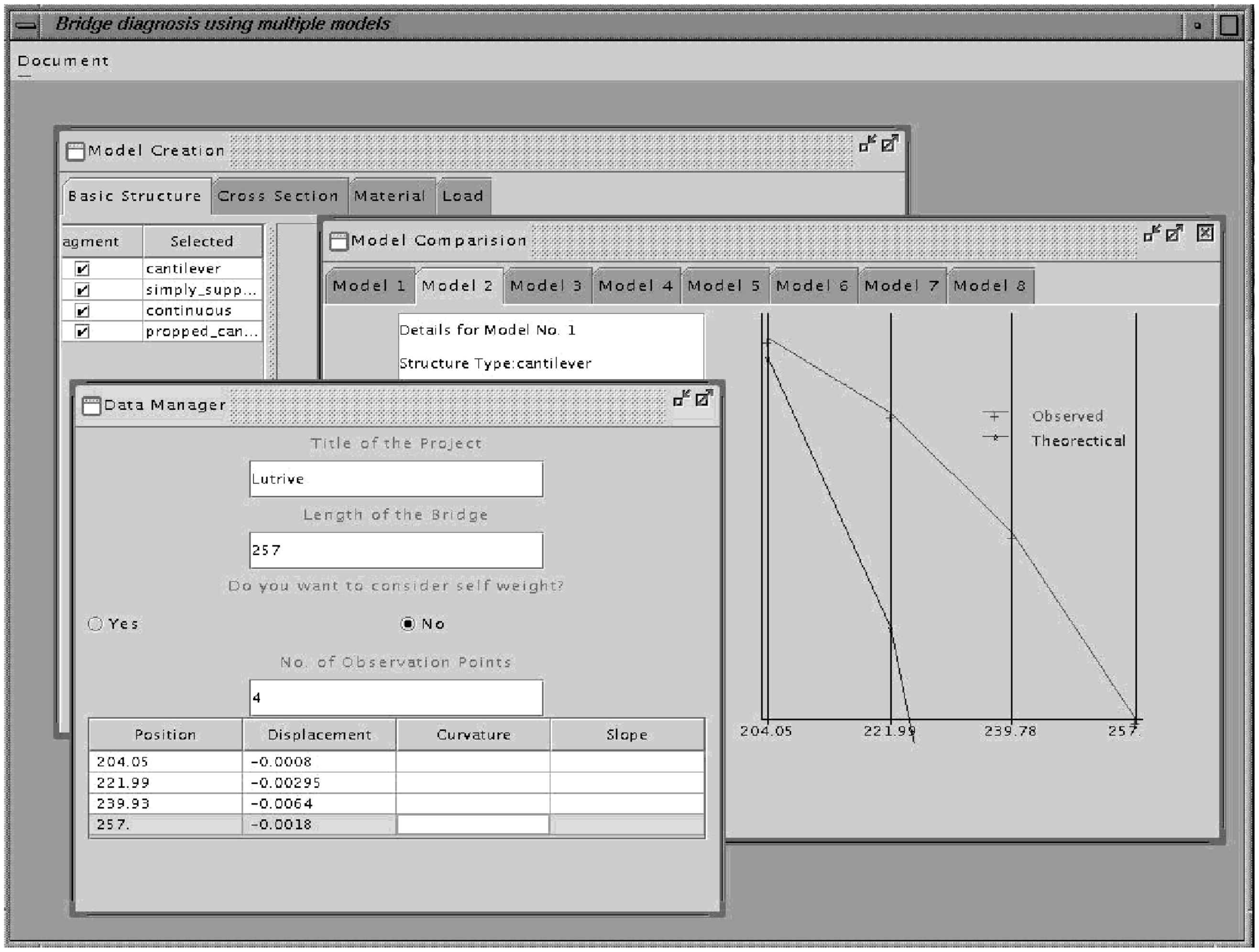
When additional fragments are added, they are visible in the table on the left and are edited by the end user through the right panel. For example, upon selecting the fragment, "continuous beams," a different set of controls specified by the engineer are displayed on the right panel. This fragment contains a button "from file" for opening a dialog box to import table data from a file. Thus, the engineer is able to define how end users interact with the system. Without an explicit representation of the model fragments of the GUI, the type of interaction would be hard coded, which would require recompilation of the program for each change.

4.3.2. Implementation details

The software was programmed in Java for portability and ease of development. Objects representing model fragments are created by reading model fragment libraries. Figure 8 shows various screens generated by the system for a sample model fragment library.

No explicit task model or user model was incorporated in the system. Users are allowed to interact freely with the system without a rigid task structure. Recent research in HCI (Boulanger & Smith, 2001) indicates that every engineer has a unique way of performing tasks, so it is not possible to use a predefined sequence of activities for complex engineering tasks.

In this application, tasks are organized into four categories, namely, data management, model creation, model use, and model evaluation. These categories correspond to those identified by work on HCI (Stalker, 2000). Each task category is organized in a separate internal frame. Each fragment library is displayed in a separate tabbed pane within



agment	Selected
✓	cantilever
✓	simply_supp...
✓	continuous
✓	propped_can...

Details for Model No. 1

Structure Type:cantilever

Title of the Project: Lutrive

Length of the Bridge: 257

Do you want to consider self weight? Yes [✓ No]

No. of Observation Points: 4

Position	Displacement	Curvature	Slope
204.05	-0.0008		
221.99	-0.00295		
239.93	-0.0064		
257.	-0.0018		

Fig. 7. A screen from the bridge diagnosis program. All the windows were generated automatically through reading model fragment libraries.

the internal frame representing the model creation task. Model fragments are displayed in panels within internal frames. Attributes of fragments are represented by UI elements, such as, text boxes, check boxes, radio buttons, and images. This organization is summarized in Table 1.

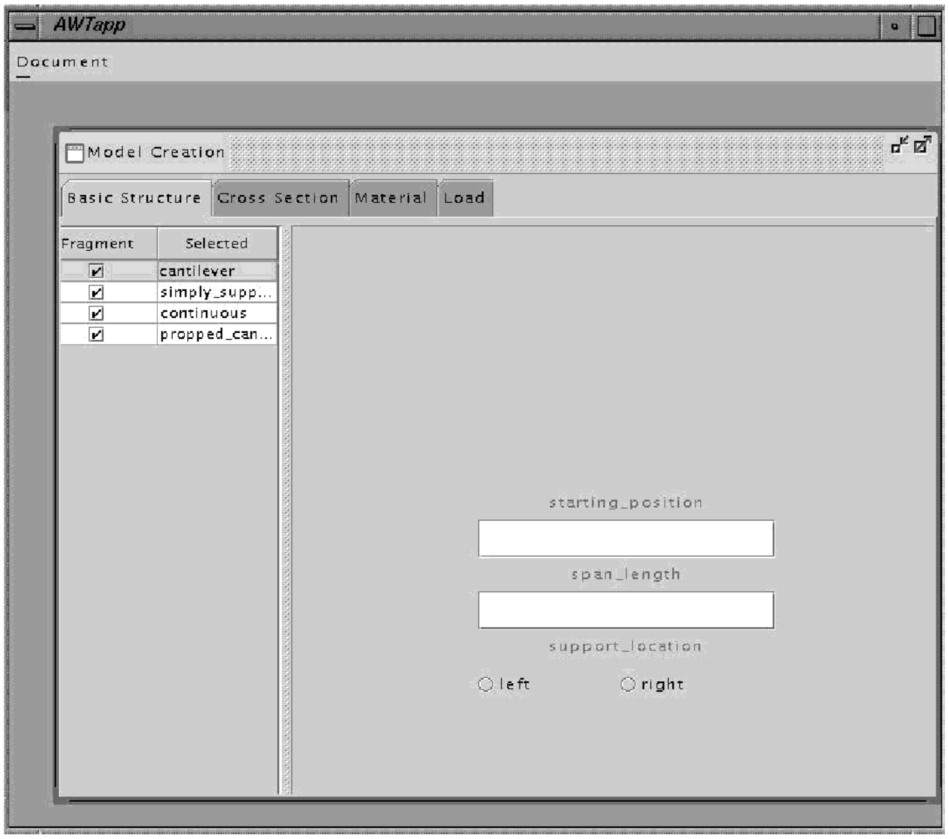
This predefined free-form structure of the UI avoids difficulties associated with developing a complex specification language for describing details of the interface. The user needs only to define representations for individual fragments in order for them to be automatically laid out by the system.

4.3.3. *Related work*

The present work differs from previous work in the area of model based UIs, for example, Foley et al. (1989), in important ways. The same process of model composition that is used to construct models of physical behavior is used to construct models of the UI. The model fragment library is used at execution to create the UI.

This approach offers several advantages:

- Because the model creation methodology that is applied to diagnosis is reused, the GUI creation module is more compatible with the application software than traditional approaches.
- Adding new fragments to the model library and changing types of attributes only require changes to the model files. There is no need to change the program. For example, an attribute that allows multiple values can be changed to a single-valued attribute through modifying model data files.
- Users may specify the GUI components for new attributes. Hence, the program allows for great flexibility. For example, a pinned roller support can be represented by an image containing the usual symbolic notation used by engineers.
- Users may change the look and feel of the interface as needs change without modifying the program. For ex-



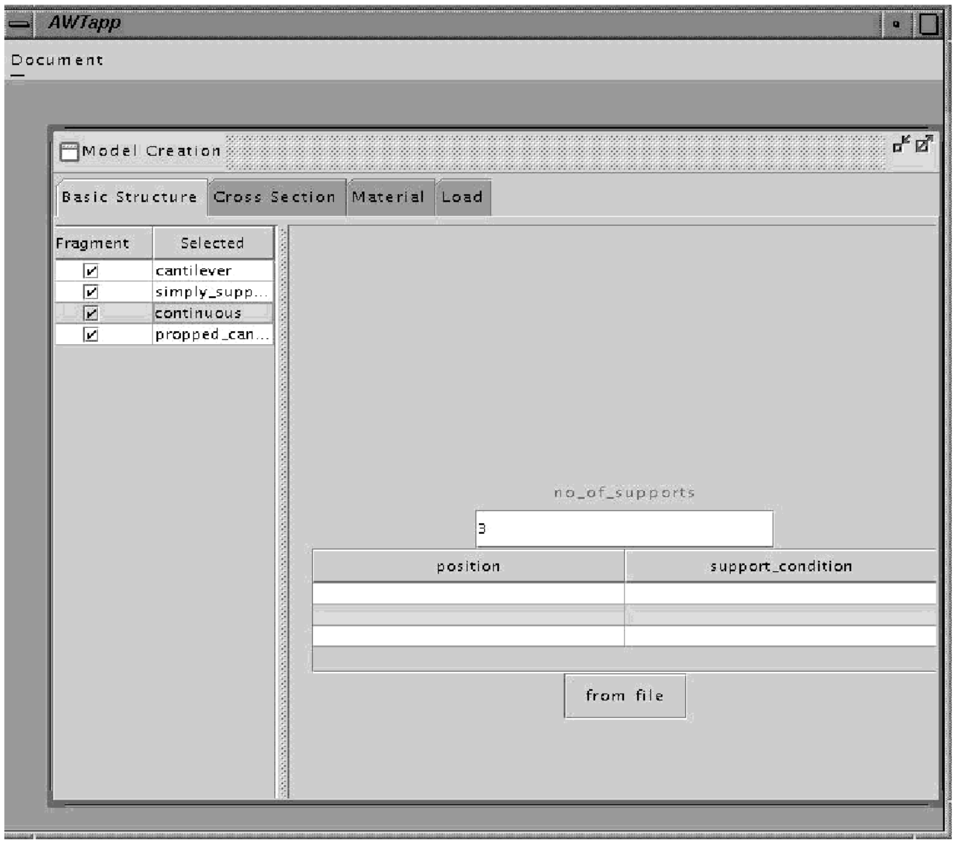


Fig. 8. Users specify visual representations of attributes of fragments. The first screen contains text boxes and radio buttons for representing simple attributes. The second screen contains a table for inputting an array of values for an attribute.

Table 1. *Predefined representations for elements of application*

Application Element	GUI Representation	Java Class
Task category	Internal frame	JinternalFrame
Fragment library	Tabbed pane	JTabbedPane
Model fragment	Panel	Jpanel
Fragment attribute	Text box, text area, check box, radio button, image	JTextArea, JRadioButton, etc.

ample, an attribute that takes multiple values can be represented by a list box or a group of check boxes.

There are currently systems that utilize model composition for creating UIs (Browne et al., 1997; Stirewalt & Rugaber, 1998). Such work is concerned primarily with generating correct and efficient UI code from declarative models. The objective of present work is not to generate code, and our software is not meant to be a programmer's tool. Instead, the software has been developed for engineers to create the most compatible GUI at execution time. This is possible through storing the decomposition and the navigational structure of the interface in simple formats that are easily understood by engineers.

4.3.4. *Limitations and possible extensions*

This research evolved from work that applied model composition to diagnostic tasks. Limitations identified in this work include the following:

- The free-form structure of interaction with the system may not be suitable for novice engineers who would like to be guided when carrying out tasks.
- Few constraints are applied during GUI composition because fragments are generally compatible with each other. An extension to the current procedure would involve a language to explicitly specify compatibility constraints.

In order to explain the second point, consider a model fragment representing a propped cantilever beam. Attributes of this fragment include the type of left and right supports. If the user decides that the left support should be fixed, a fixed support at the right is not allowed, because by definition, a propped cantilever is fixed at one end and supported on rollers at the other end. These constraints are easily enforceable if the UI is static. However, in the current application, GUI elements corresponding to fragments are created at run time. Therefore, enforcing such constraints would require the activation of a constraint propagation module.

The enforcement of a range of integrity constraints would involve the development of a comprehensive constraint specification language in order to check the compatibility of model fragments and the values of their attributes. For example, the syntax of a possible language is as follows:

```
Fragment proppedCantilever
    constraintType attributeValues
        NOT (leftSupport == fixed AND rightSupport ==
             fixed)
```

Such constraint languages tend to be complex; as a result, engineers may encounter difficulties during use. A graphical language for constraint specification might provide assistance. These issues are being considered in current research.

To a certain extent, the visual representation of a fragment can be automatically determined from the data types of its attributes. This has not been done in the current implementation because of the following reasons:

- The engineer usually knows the best visual representation for each attribute.
- Not all attributes of a fragment need to be displayed to users. Certain attributes are computed using other attributes, and the engineer can decide what should be visible and what should be hidden.

5. CONCLUDING REMARKS

UI technologies have made much progress in the recent decades. This is partly due to research into direct manipulation interfaces, whereby users directly point at objects on the screen and manipulate them (Schneiderman, 1983). However, several difficulties remain for the development of good interfaces, especially for engineering applications. Most current designs result in interfaces that are difficult to adapt. This paper proposes the use of explicitly defined model fragments that, when composed, provide UIs for engineers that are compatible with needs. The prototype system described in this paper demonstrates the application of model composition to the development of GUI that can be customized by end users. Such model-based approaches show much potential for creating adaptable graphical engineering interfaces. Adaptability is expected to increase the quality of computer support and thereby encourage greater degrees of creativity and efficiency. Work is in progress to extend software for tests on tasks associated with the monitoring and diagnosis of full-scale structures.

ACKNOWLEDGMENTS

This research is funded by the Swiss National Science Foundation, the commission for technology and innovation, and the Alliance for Global Sustainability. G. Bhatnagar, from the Indian Institute of Technology was funded by the Centre for International Relations, EPFL. We would like to thank Mr. Arun Kumar and the late Prof. C.S. Krishnamoorthy of IIT Madras for their contributions in the implementation of the initial version of the model composition module and its application to Arur Kaayal Bridge in

India. We are also grateful to Mr. Yvan Robert-Nicoud for the use of information related to the Lutrive Bridge.

REFERENCES

Boulanger, S., & Smith, I.F.C. (2001). Multi-strategy workspace navigation for design education, *Design Studies 22*, 111–140.

Browne, T.P., Davila, D., Rugaber, S., & Stirewalt, K. (1997). Using declarative descriptions to model user interfaces with MASTERMIND. In *Formal Methods in Human–Computer Interaction* (Paterno, F., & Palanque, P., Eds.). New York: Springer–Verlag.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. New York: John Wiley.

Colvin G. (2001, March 5). The kid in my computer. *Fortune* 26.

Falkenhainer, B., & Forbus, K.D. (1991). Compositional modelling: Finding the right model for the job, *Artificial Intelligence 51*, 95–143.

Foley, J., Kim, W., Kovacevic, S., & Murray, K. (1989). Defining interfaces at a higher level of abstraction. *IEEE Software 6*, 25–32.

Grover, V., & Goslar, M. (1993). Information technologies for the 1990s: The executives' view. *Communications of the ACM 36*, 17–19.

Hartmanis, J., & Lin, H. (1992). Computing the future: A broader agenda for computer science and engineering. *Communications of the ACM 35*, 30–40.

Myers, B.A. (1993). *Why are human–computer interfaces difficult to design and implement?* Technical Report CMU-CS-93-183. Pittsburgh, PA: Carnegie Mellon University.

Myers, B.A. (1995). User interface software tools. *ACM Transactions on Human–Computer Interaction 2*, 64–103.

Paterno, F., & Mancini, C. (2000). Model-based design of interactive applications. *Intelligence 28*, 27–37.

Pinheiro da Silva, P. (2000). User interface declarative models and development environments: A survey. *Seventh Int. Workshop on Design, Specification and Verification of Interactive Systems*, Limerick, Ireland.

Puerta, A.R., Cheng, E., Ou, T., & Min, J. (1999). MOBILE: User-centered interface building. *Proc. CHI '99*, pp. 426–433.

Raphael, B., & Smith, I. (1998). Finding the right model for bridge diagnosis. In *Artificial Intelligence in Structural Engineering*, *Computer Science*, *Lecture Notes in Artificial Intelligence*, Vol. 1454 (Smith, I.F.C., Ed.), pp. 308–319. Heidelberg, Germany: Springer.

Robert-Nicoud, Y., Raphael, B., & Smith, I.F.C (2000). Decision support through multiple models and probabilistic search. *Proc. Construction Information Technology 2000*, Iceland Building Research Institute.

Schneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer 16*, 57–69.

Schneiderman, B. (1997), *Designing the User Interface: Strategies for Effective Human–Computer Interaction*, Reading, MA: Addison–Wesley.

Smith, I.F.C (1996). Interactive design. In *Information Processing in Civil and Structural Engineering Design* (Kumar, B., Ed.), pp. 23–30. Edinburgh: CIVIL-COMP Ltd.

Smith, I.F.C (1999). Designers like designing. In *Bridging the Generations: The Future of Computer Aided Engineering* (Garrett, J.H., & Rehak, D.R., Eds.), pp. 105–109. Pittsburgh, PA: Carnegie Mellon University, Department of Civil Engineering.

Smith, S.L., & Mosier, J.N. (1986). *Guidelines for designing user interface software*. Technical Report ESD-TR-86-278. Bedford, MA: MITRE.

Stalker, R. (2000). *Engineer–computer interaction for structural monitoring*. PhD Thesis. Lausanne, Switzerland: IMAC-DGC, Swiss Federal Institute of Technology, EPFL.

Stalker, R., & Smith, I. (1999). An interactive toolkit for structural monitoring. In *Artificial Intelligence in Structural Engineering* (Borkowski, A., Ed.). Warsaw, Poland: Wydawnictwa Naukowo-Techniczne.

Stirewalt, R.E.K., & Rugaber, S. (1998). Automating UI generation by model composition. In *13th Conf. Automated Software Engineering, ASE'98*, Honolulu, HI. New York: IEEE.

Szekely, P., Luo, P., & Neches, R. (1992). Facilitating the exploration of interface design alternatives: The HUMANOID model of interface design. *Proc. CHI'92, CM Conf. Human Factors in Computing Systems*, pp. 107–114.

Van Harmelen, M., Artim, J., Butler, K., Henderson, A., Roberts, D., Rosson, M.B., Tarby, J.-C., & Wilson, S. (1997, October). Object-oriented models in user interface design. CHI 97 Workshop. *SIGCHI Bulletin*.

Wiecha, C., & Boies, S. (1990). Generating user interfaces, principles and use of ITS style rules. *Proc. UIST'90*, pp. 21–30.

Benny Raphael is a Research Associate at the Applied Computing and Mechanics Laboratory at the Swiss Federal Institute of Technology in Lausanne. He obtained his BS and MS degrees from the Indian Institute of Technology, Madras, in 1990 and 1992, respectively. He obtained his PhD from Strathclyde University, Glasgow, UK, in 1995. Dr. Raphael's interests include global search UIs, case-based reasoning, and other applications of artificial intelligence to engineering.

Gaurav Bhatnagar obtained his BS in computer science and engineering from the Indian Institute of Technology, New Delhi, in 2001. He is currently working as a software Design Engineer at Microsoft in Redmond, WA, USA. Gaurav's interests include algorithms, computer networks, and HCI.

Ian F.C. Smith is currently Professor and Head of the Applied Computing and Mechanics Laboratory at the Swiss Federal Institute of Technology in Lausanne. He earned his engineering degree at the University of Waterloo, Canada, in 1978 and his PhD at Cambridge University in 1982. Dr. Smith's interests include artificial intelligence applications, computer supported cooperative work, HCI, infrastructure monitoring and repair, intelligent structures, global sustainability, and intelligent CAD.