



Repositorio Institucional de la Universidad Autónoma de Madrid

<https://repositorio.uam.es>

Esta es la **versión de autor** de la comunicación de congreso publicada en:
This is an **author produced version** of a paper published in:

1998 Australasian Computer Human Interaction
Conference, Proceedings. IEEE, 1998. 306-313

DOI: <http://dx.doi.org/10.1109/OZCHI.1998.732229>

Copyright: © 1998 IEEE

El acceso a la versión del editor puede requerir la suscripción del recurso
Access to the published version may require subscription

KIISS: a System for Visual Specification of Model-based User Interfaces

Francisco Saiz*

Javier Contreras*[†]

Roberto Moriyón*

*Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Cantoblanco 28049, Madrid, SPAIN
{francisco.saiz, javier.contreras,
roberto.moriyon}@ii.uam.es
Fax: + 34- 1- 397- 5277

[†] LIAP-5
Université René-Descartes
7505 Paris, FRANCE
conj@descartes.math-info.univ-paris5.fr

Abstract

The appearance of model-based techniques for interface development has simplified the design of complex interactive applications. But this approach still requires from the designer a high knowledge level about the textual specification required. This paper presents a system, KIISS, which allows the designer of an application to interactively define the model of its interface through visual specifications on an application example. Thus, the system enhances the model by allowing its use by designers who are not quite familiar with the textual specifications required for a user interface development. Moreover, reusability is preserved, since parts of existing applications can be interactively both exchanged and modified.

Keywords

Interactive User Interface Design, Model-based Specification, User Interface Technique

1: Introduction

The aim of this paper is to show how the model-based paradigm for the construction of user interfaces (UIs) allows the interface development not only through textual definitions, but also by means of visual, interactive, and more intuitive specifications on an application example. We shall do this through a description of some of the most relevant aspects of KIISS¹, an editor for the interactive development of UIs that can include context sensitive presentations.

When compared with the most advanced model-based tools for the design of UIs, one of the main and novel features of KIISS, is that the developer of the interface does not need to rely on all the details about the textual and formal model that represents it. Instead, relations among parts are expressed graphically, and every operation enabled in the textual model can be performed interactively as well.

KIISS allows the interactive specification of the main aspects of the interface, including presentation and user interaction. Since different parts of the window where the work is going on can correspond to the same part of the model under modification, it is necessary to specify which of these parts should be modified. KIISS gives the designer all the information and mechanisms necessary to

¹ KIISS stands for Knowledge-based Interactive Interface Surgery System.

decide where the actions are to take effect. KISS gets rid of ambiguities through dialogues that specify *presentation patterns*. These patterns determine sets of widgets by specifying properties that they must satisfy.

The need to explore the possibilities of interactive UI building is clear, since the tools that are used nowadays for this task have very limited capacity. The most common tools, interface builders like [10], [9], can only build part of the static components of the interface. More advanced development tools, like DRUID, [3], incorporate also the ability to specify some constraints between parts of the display, but the kind of constraints that can be specified in this way is very limited. As a matter of fact, the interactive specification of the possibilities of interaction for an application under development is a field that is still a matter of research.

Tools developed as the result of research efforts have achieved some success in this direction; the set of tools built on top of GARNET, [4], are especially remarkable: both LAPIDARY, [6], an advanced editor of widgets that allows the definition of geometric constraints, and C32, [5], an editor for generic constraints, simplify considerably the development of complex interfaces with this kind of constraints. KISS applies techniques similar to those developed in C32, and extends them to cover most aspects of the development of model-based interactive applications. MARQUISE, [7], represents an interesting attempt to incorporate sophisticated specification by example of constraints in this context, but the field covered by these techniques is relatively small.

Finally, the model-based approach has made remarkable contributions in this direction, which can be best exemplified by HUMANOID, [12], and UIDE, [2], developed at ISI/USC and Georgia Tech respectively. More recently, a more powerful model-based tool for the design of interactive applications, MASTERMIND, [14], is being developed as a joint effort between the institutions cited above. KISS is built on top of HUMANOID.

But, as Brad Myers has pointed out in [8], one of the biggest problems of model-based systems is that they are not easy to use. In fact, the model-based approach has succeeded incorporating complex frameworks that cover more and more aspects of the overall application. But it has failed when trying to allow the interactive specification of complex interfaces by non-programmers. On the other hand, the dual approach (textual and visual) for building UIs has been already considered in systems like XXL, [15], but not within the model-based approach.

For example, HUMANOID incorporates editors for templates (presentation models), application and command models, [13], that are useful in the application

development, but still require a lot of knowledge from the user about the structure of the model. More specifically, editing of the interface is done in HUMANOID through a mechanism that gives the designer a view of the model of the interface and a view of an example of the interface itself, but the editing is done all the time on the model.

KISS allows the designer to edit the interface directly on an example, simplifying very much the editing process, and the amount of knowledge required about the model. This also enhances reusability of UIs components, since the editing of existing applications is done in a simple way, and it does not require a deep knowledge of the underlying model. This is achieved by an extension of the HUMANOID model that includes *Virtual Slots* [11], which allows the use of graphical objects that act on other objects to which they can be attached, like rulers for defining lengths. Some techniques to control the sequencing of the application have been also developed, which are described elsewhere, [1].

Although KISS can be considered from an abstract point of view as a modeling system completely independent of other models like the one underlying HUMANOID, in practice it uses extensively a feature of HUMANOID's model that is not present in others, except for MASTERMIND, namely the consideration of presentations with conditional appearance and behavior. As a consequence of this, it could be implemented on top of MASTERMIND with an effort similar to the one spent to do it on top of HUMANOID, but it could hardly be implemented on top of other model-based systems without major additions to them. Finally, let us mention that, while MASTERMIND addresses hard design issues such as adding power to the interactive design of constraints, and many others, it has essentially the same capacities and limitations as HUMANOID in the main aspects of user interface design that are addressed by KISS.

The rest of the paper is organized as follows: we begin with an example of an application to be built interactively using the KISS editor, followed by an explanation along subsequent sections of how KISS works based on this example. This will illustrate our claims about the simplification of the design process achieved by the use of KISS. The last section is devoted to Presentation Patterns, one of the main features of the editor, and a description of its architecture.

2: An example

The goal of this section is to show in some detail the kind of interfaces that KISS is able to produce. We shall

show two different stages of the construction of a simple but still representative interface. As we present both stages, we shall introduce the model that lies behind them when they are built using a model-based system like HUMANOID. This information will be used in next section, where the most relevant aspects of the editing process that allows to construct the last interface from the simpler one will be described.

Our starting point will be a *simple folder browser*, shown in Figure 1, which just displays the names of the files in a folder that can be specified by the user. The only functionality we shall assume it has corresponds to clicking on the buttons *quit* and *refresh*, and typing on the *folder* field after clicking on it with the mouse. The effect of these three actions is in each case the obvious one.



Fig. 1: In this figure a simple folder browser application is displayed. The *folder* is the input of the application, and *quit* and *refresh* its corresponding commands.

The final application to be built is *the semantic folder browser* shown in Figure 2, that displays the relevant information about the files in a folder, the kind of information depending on the type of file. For example, we might want to be able to see for each bitmap file in the folder an icon that is a reduced copy of the bitmap it represents. This might be useful when looking for specific bitmaps in a folder with many files of this type. In the example we give, the user can decide whether

these files should be seen by name or by icon by using the subitems of the menu *appearance*, while other fields will always be shown by name. Another possible application of a semantic browser like this would be to show files of type agenda by inserting a list with the names, phone numbers, etc., according to the criteria specified by means of the subitems of the *sort* menu, and a small icon showing a picture of each person in the agenda. Moreover, the user can filter the types of files to be shown. Finally, we shall assume that the user can sort the list of files by size, date, etc., and that it is also possible to drag a file into the *Folder* editing field and, if it is a folder, it will be browsed. What we want to stress here is that the *semantic file browser*, which is clearly a more powerful application than the initial *file browser*, can be developed interactively with KIISS.

Let us examine in some detail some of the features in the underlying model of these two applications. The following explanations are “textual” specifications required for the definition of the interfaces in HUMANOID. We must stress that, as we shall see, the complexity of these definitions is eased when they are specified in KIISS.

As for the interface represented in Figure 1, the presentation model consists of a *window template* that has three *parts*: an *input panel template* (modeling the upper part of the window), a *column template* (modeling the window body), and a *command panel template* (containing the buttons). The second *part* has a *subpart*, a *label template*, which is replicated, i.e. there are as many widgets in this part as *files* in the *folder* being browsed. Here, the list of *files* constitutes a data associated with the *body* of the window. The *folder browser application* consists of a description of the *inputs* (parameters) and the *commands* (in this case there is just one *input*, the *folder* mentioned above, and the *commands* *quit* and *refresh*).

Finally, the interaction model for Figure 1 specifies essentially the possibilities to click with the mouse on the lower buttons, and also to type in the upper editing field, and the corresponding effects. Each interaction possibility is modeled through an object called *interaction template*, which includes several *events*, like the *start event*, and the *stop event*, several *wheres*, like a *start where* and a *stop where*, and an *action*.

The application depicted in Figure 2 has some major differences with the original one (apart from other simpler ones such as a new menu bar, and a new *type editing field* in the *input panel template*). For instance, the body of the window has now two parts, a *row template* that includes the headings for the columns, and a *column template* as before. But now the replicated *part* is not a *label template* any more, but another *row template*.

This means that there are new *data* corresponding to the different new *labels* that appear. These new *data* compute their values from the corresponding *files* in the surrounding *row templates*.

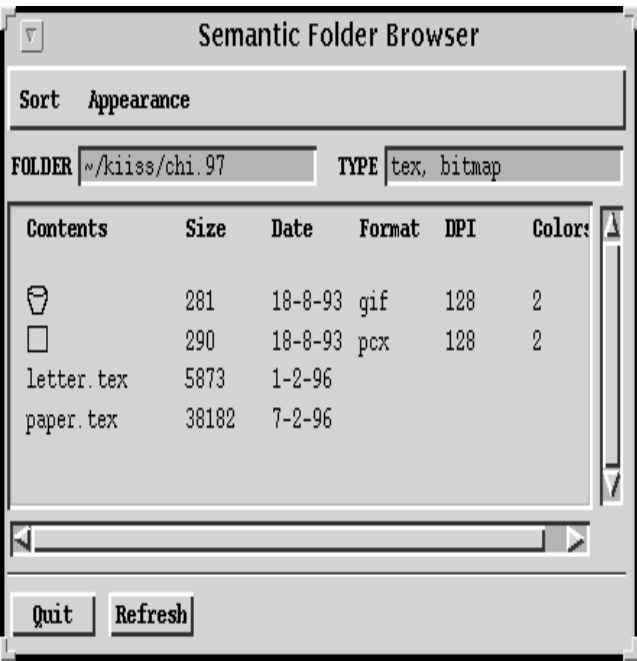


Fig.2: This figure displays the semantic folder browser application. It includes additional attributes (*size, date, etc.*) of files, as well as new commands in the application (*sort by size, sort by date and graphics*) and the input type.

The last four parts in the *row template* that shows the *files* are made up of *label templates*. But its first part can be either a *label template* or a *bitmap template* on different instantiations or even at different moments. This is a new kind of *presentation template*, a *substitution template* that models this kind of situation. Actually, the *row template*, mentioned in the previous paragraph is another *substitution template* that is shown only in case its corresponding file is of one of the desired types. Finally, in the application semantics model, there is one more *input*, the list of *types* of the files to be shown.

There are also new *interaction templates* associated to the menus, the new editor in the *input panel*, and the dragging action. The last one is a *dragging interaction template*, associated to the first *part* in the replicated *row template*. Its corresponding *action* is a *set input value action*, the *start where* function returns the widget that contains it generated by the *substitution template*

mentioned above (in case there is one and it corresponds to a folder), and the *stop where* function returns the *folder* editing field.

2: Editable dimensions in KIISS

There are four aspects of an application that KIISS can visualize and modify. All of them are attached to a part of the window selected before any visualization or editing takes place. Editing in general can modify, create or destroy specific features. Modification and creation can be done by direct specification of the new properties or by importing them from another object, or even from another application. *Presentation patterns* (described in section 4) will serve as a means of interactive non-ambiguous specification for properties on imported objects. Let us examine these aspects:

- *Visual aspects* considered by KIISS can be either the *presentation template* to be used, or parameters that determine the geometric properties of the widgets to be generated and their appearance, like the *top, height, vertical interspacing* (if appropriate), *color*, etc. Since they correspond to generic descriptions of these attributes in a model of presentation, very often their values are not just constants, but formulae that will be evaluated for each instance of the *template* that is created.
- *Data* are generic parameters of the generation of widgets like bitmaps, text (such as the name of the file where the bitmap is stored), etc. *Application inputs* and *commands* are also included in this design dimension.
- *Sensitivity* refers to the different amounts and kinds of graphical objects that are generated from a *template* when it is instantiated. For example, some *templates* are replicated at instantiation, while some others give rise to different types of graphical objects depending on specific conditions. Others can appear or not depending also on some conditions. In the example from the previous subsection, we can have a bitmap or a label depending on the type of file associated to the row.
- *Interaction*. Finally, KIISS allows the user to edit interactively the *events, where places* and *actions* associated to a given *template*. The mechanism of *presentation patterns*, which will be described below, reduces the task of specifying a *where value* to the definition of links between some components. Similarly, specifying an *action* can also be reduced to the specification of the type of *action*, and the specification of the information associated to it. Both tasks can be done in a way similar to the editing of visualization aspects or data described above.

3: Information Visualization

KIISS allows the visualization of the information related to the four aspects just mentioned, in the same window of the interface, highlighting graphically the

references to other parts of the interface. Information about the application inputs and commands can be displayed similarly. This information is given usually by complex constraints, which are enforced for each instance of the application. The designer can hide part of the information shown about each object to avoid cumbering the screen.

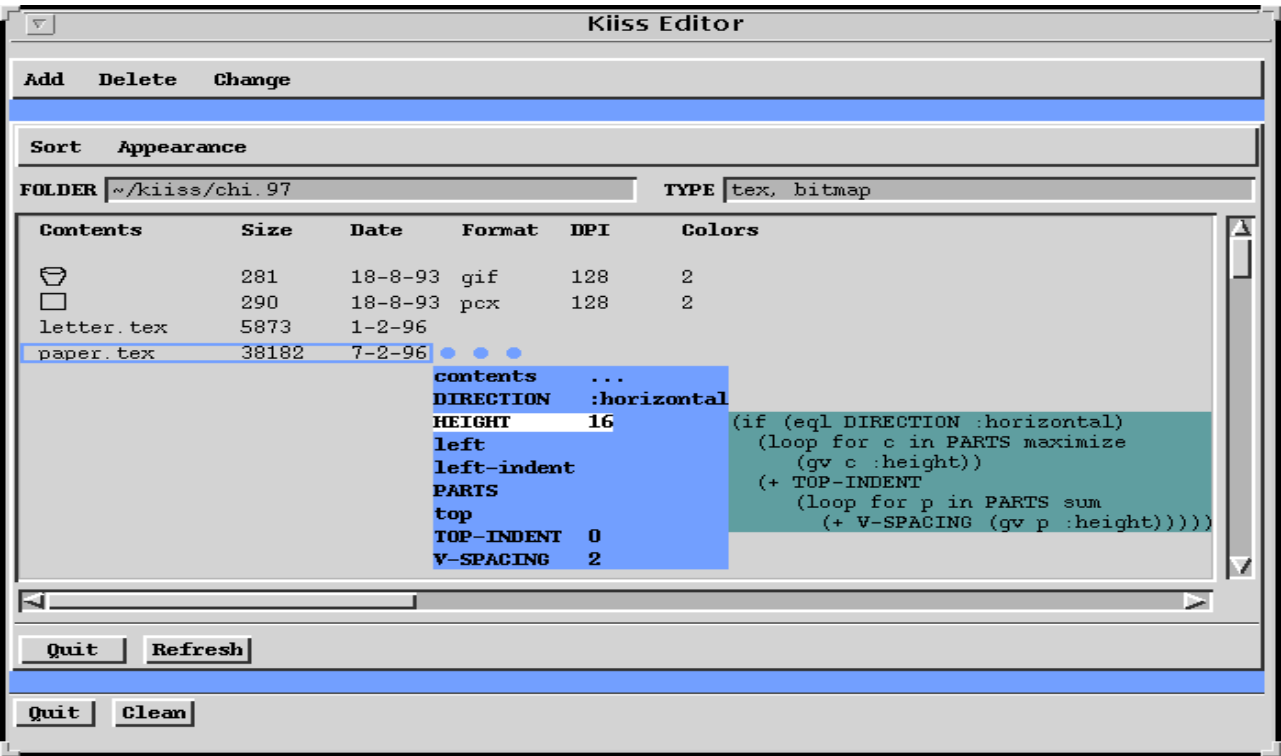


Fig. 3: The Semantic Object Browser is displayed, where a replicated row is selected. An information sheet describing its attributes appears next to it, where the height slot is being edited.

Figure 3 shows a visualization of geometric aspects of the highlighted part in the *semantic folder browser*. Data in the formula that appear on the right referring to information contained in the sheet on the left side are highlighted in the same color. In general, the designer can visualize information about several dimensions and objects at the same time.

It is crucial for the understanding of editing in KIISS, to notice that the designer selects on the window some information related to specific widgets appearing on it, but the modifications he indicates are performed on *templates*, abstract objects that represent families of widgets to be generated by instantiation. The designer also specifies or modifies formulae that link together several widgets, but again what KIISS does is annotating those links in their corresponding abstract templates.

Figure 4 shows a step of the editing and visualization of the final action of the *drag and drop interaction template*. After having entered a *set input value action* into the part marked *final action*, the designer must drop over there the data *filename* corresponding to the label of the left, and the input *folder* of the application. This instantiates the input and value that appear in the editing field on the right. The specification of the event that activates the *drag and drop interaction template* is done by demonstration, while the specification of the *start* and *stop wheres* is achieved by means of *presentation patterns*, as explained in the next section.

4: Presentation patterns

Whenever the designer specifies a modification associated to a *template*, KIISS asks whether that modification should take effect on all graphical instances of the *template* or only on some of them. In case the user wants to reduce the extent of the modification being specified, this is done through the mechanism of *presentation patterns*. This mechanism can also be used

to define sensitive presentations, as it happens in the case of the first column in the rows representing files in the example from the previous section. In this case, the designer will specify the condition for the substitution to take place by indicating interactively that the type of the file associated to the row must be bitmap. Finally, it also allows the interactive definition of the *start where*, and *stop where* slots associated to *interaction templates*.

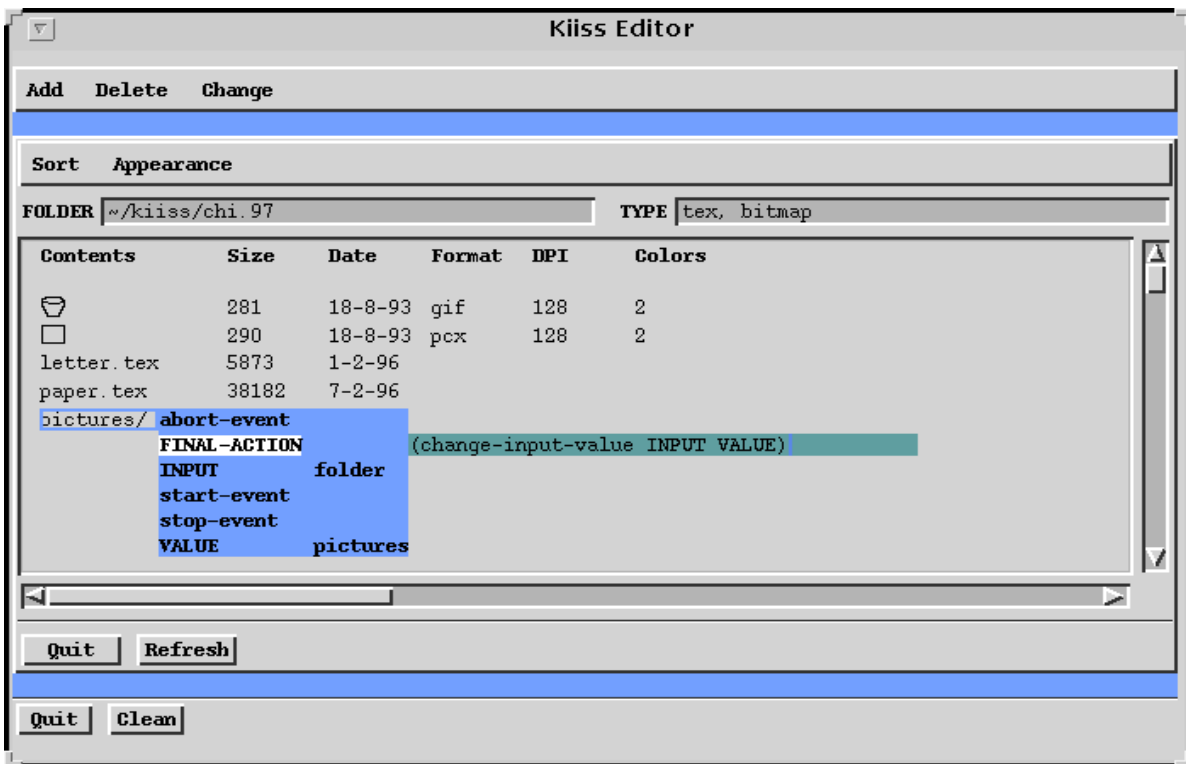


Fig.4: The Semantic Folder Browser is displayed, together with the interaction information sheet for a directory row.

Presentation patterns are objects that represent patterns of widgets defined by properties they must satisfy. The following parameters define *presentation patterns*: a) properties of the location where the widget is displayed, like being contained in some given part of the window; b) properties of its associated *presentation template*, like being an instance of a given one; c) properties of some data associated to the widget.

An example of a *presentation pattern* related to the interface described in the previous sections is the one that matches all the icons that represent image files. This *presentation pattern* is defined by requiring the corresponding template to be an instance of the *image file template* from the file presentation template library.

A *presentation pattern* contains a predicate on widgets, and hence it also represents a set of such widgets. Actually, *presentation patterns* can also be built from simpler ones by their conjugation (in which case they represent the conjugation of the corresponding predicates), disjunction, or negation.

Presentation patterns also allow the definition of conditions in *substitution templates*, which are defined by means of a list of pairs, each formed by a condition for the substitution to hold and a *template* that substitutes the one under construction in case the corresponding condition is satisfied. For example, the *template* used to present a file that was mentioned in the previous paragraph is a *substitution template* that becomes an *image file template* in case its corresponding *presentation*

pattern matches the widget under consideration. This matching holds whenever the name of the file associated to the widget has the right termination, like “.bitmap”.

5: Architecture and implementation

KIISS is a model-based application built on top of HUMANOID. The model it uses is an extension of that of HUMANOID. Hence, it models the application being designed according to its presentation, interaction, and navigation aspects. It has a specific model for the interaction with the user that allows the evolution of the design; the main ingredients of this model have been described in the previous section through an example, i.e. the *presentation patterns* and the *information sheets*. These components of the system use extensively the possibilities of HUMANOID’s model to specify presentations whose visibility and appearance depend on specific conditions.

The main input of the KIISS editor is the model of the application being edited, such as the *simple folder browser* in our example. KIISS first generates a copy of the application it is editing, and adds interactive functionality to it that is useful for its editing. Then the application model is changed successively according to the user’s actions. Finally, when the designer of the application saves the design, the application is first

enabling Runtime System, and from the editing request as input, modifications are performed in the application model, which are updated in the example application window.

Figure 5 shows the data flow during an interactive session with KIISS. The input to the system is an editing request, which is accomplished either on a region of the application display, in case of parts editing, or on the sheet that represents a specific kind of information associated to such a region, in case of the editing of other types of information. Most inputs affect the model of the edited application, either directly or indirectly, and the KIISS windows where the main attributes appear.

The runtime system of HUMANOID interprets the editing request from the user and creates and modifies the corresponding window. KIISS application model serves as the starting point for the generation of all the KIISS information sheets appearing along the editing session.

In the figure, we can see that KIISS uses data from the application, and modifies them. On the other hand, the arrows on the left-hand side of the figure show how the application receives events and reacts to them as if it was working in a standalone regime. This is an important fact that allows the user not only to specify interactively the reaction of the application to events, as we have explained in previous sections, but also to experiment with them and correct errors on the fly.

6: Conclusions

This paper has shown the main features of KIISS, which accomplishes interactive modifications on the different components of a given model-based interface. We have proven that the model-based paradigm for the construction of user interfaces can be enhanced allowing the development of more dimensions of complex interfaces in an interactive way. The dimensions covered by KIISS include the presentation, interaction and application models. Getting rid of ambiguities is achieved by using *presentation patterns* that define a set of widgets by certain properties. The above facts have been introduced through an example where we have seen how KIISS allows the designer of the user interface to specify the extents of changes in presentation and interaction with the user that previous systems like HUMANOID did not allow.

We have developed a prototype that includes essential aspects of the functionality described in the

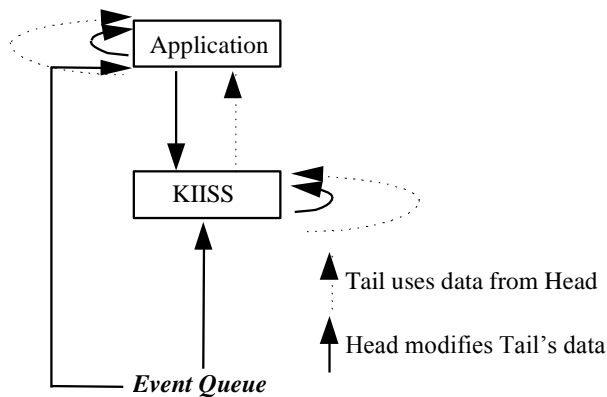


Fig. 5: Architecture of KIISS. HUMANOID provides the

example through this paper, but it still has important limitations, especially about the location of the information sheets when the application has superimposed windows, like scrolling windows. In spite of these limitations, the most remarkable advance obtained from the use of the system is the proved fact that visual specifications on an application example turn out to be much more flexible and easy to use than textual definitions, which are the classical starting point in model-based interfaces.

Acknowledgements

KIIS is partially supported by the Plan Nacional de Investigación, Programa Nacional de Tecnología de Información y de las Comunicaciones, Spain, project number TIC93-0268, and a special grant from Comunidad de Madrid (Acción Especial KIIS).

References

1. Contreras, J., and Saiz, F.. A Framework for the Automatic Generation of Software Tutoring. In *Proceedings of CADUI'96*, EUROGRAPHICS, Presses Universitaires de Namur, 1996, pp. 171-182.
2. Foley, J., Kim W., Kovacevic S., and Murray, K. *Uide: An Intelligent User Interface Design Environment*, Addison-Wesley, Reading MA, 1991, pp. 339-384
3. Gurminder S., Hong C., and Ye T.: *Druid: A System for Demonstrational Rapid User Interface Development*. In *Proceedings of UIST'90*, ACM, 1990, pp. 167-177.
4. Myers, B.A., et. al.. GARNET: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. *IEEE Computer* 23(11), 1990, pp. 71-85.
5. Myers, B.A.. Graphical Techniques in a Spreadsheet for Specifying User Interfaces. In *Proceedings of CHI'91*, ACM, 1991, pp. 243-256.
6. Myers, B. A.. Lapidary. *Watch What I do: Programming by Demonstration*. Allen Cypher (ed.), The MIT Press, Cambridge, 1993.
7. Myers, B.A., McDaniel, R.G, and Kosbie, D.S.. *Marquise: Creating Complete User Interfaces by Demonstration*. *INTERCHI'93*, 1993, pp. 293-300.
8. Myers, B.A.. User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, Vol. 2, No. 1, pp. 64-103, March 1995.
9. Neuron Data, Inc.. *Open Interface Toolkit*. Palo Alto, CA, 1991.
10. NeXT, Inc.. *Interface Builder*. Palo Alto, CA, 1990.
11. Saiz, F., Contreras, J., and Moriyon, R.. Virtual Slots: Increasing Power and Reusability for User Interface Development Languages. In *Proceedings of CHI'95*, ACM, 1995, pp. 236-237.
12. Szekely, P., Luo, P., and Neches, R.. The HUMANOID Model of Interface Design. In *Proceedings of CHI'92*, ACM, 1992, pp. 507-514.
13. Szekely, P., Luo, P., and Neches, R.. Beyond Interface Builders: Model-Based Interface Tools. In *Proceedings of INTERCHI'93*, 1993, pp. 383-390.
14. Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasany, J. and Salcher, E.: *Declarative Interface Models for User Interface Construction Tools: The Mastermind Approach*. In *Engineering for Human- Computer Interaction*, L. Bass and C. Unger, Eds. Chapman & Hall, 1996.
15. Lecolinet E.: XXL: A Dual Approach for Building User Interfaces. In *Proceedings of UIST'96*, ACM, 1996, pp. 99-108.