Virtual Slots: Increasing Power and Reusability for User Interface Development Languages

Francisco Saiz,† *Javier Contreras*,* *and Roberto Moriyon*†
† Instituto de Ingenieria del Conocimiento
Universidad Autonoma de Madrid, SPAIN
E-mail:<saiz, roberto>@lola.iic.uam.es
* LIAP-5, Universite Rene-Descartes, Paris, FRANCE
E-mail: conj@descartes.math-info.univ-paris5.fr

ABSTRACT
An extension to constraint-based user interface development languages is shown. It permits the abstract representation of constraints which must be applied to objects that are not accessible in the moment of the constraint definition. Using this mechanism, more modularity is achieved, as each part of information is stored where it is needed. Richer libraries of reusable objects can therefore be built in a natural way.

KEYWORDS Constraints, reusability, user interface implementation, libraries.

INTRODUCTION

In this paper we propose an extension for constraint-based languages for user interfaces (UI) development that brings the developer closer to the conceptual level of specification of the interface, and at the same time overcomes limitations for its reusability. The technique we propose, *Virtual Slots*, can be implemented as an extension to languages like the ones used in systems such as Garnet, [4], Rendezvous, [1], SkyBlue, [5], or ThingLab II, [2]. The use of Virtual Slots improves the language's power and expressivity, and makes some programming tasks easier.

As a simple example that will allow us to introduce both the limitations of present languages, and the kind of solution we propose, we consider the development of a folder manager that allows the dragging of objects (e.g. file icons) and notifies the user when they can be dropped on the one below them. This can be done by highlighting this last object, or by a spoken message.

One of the main features of Virtual Slots is that generic reusable interaction styles can be defined. In our example, the interaction style that allows the user to drag objects and at the same time highlights automatically some others can be stored in a library for its further reuse in other applications. Moreover, such objects can be defined in a simple way without doing general programming, i.e. just defining the corresponding objects by a declarative specification of their components. Also, Virtual Slots allow the partial specification of qualities of objects (constraints for their position, visibility, etc.) without making complete commitments about the kind of objects to be used. These achievements can be seen as complementary of model-based interface design tools like Humanoid, [6]. Finally, let us point out that current systems do not have the above capabilities.

VIRTUAL SLOTS

Constraint-Based UI Systems: Concepts and Drawbacks

In this section we illustrate both the aspects of current constraint systems for UIs, and their main drawbacks, that are relevant for the paper. We shall be concerned with Garnet, but our main conclusions are also valid for the rest of the systems mentioned above.

Garnet allows the definition of (uni-directional) constraints by assigning formulae as values of slots; these symbolic formulae are evaluated whenever the computation of a new value of the slot is needed. Garnet also includes a limited set of predefined interactors, objects that model interaction with the user at the level of mouse and keyboard events, and the objects present in the interface. The user can instantiate them according to his needs. But any behavior that can not be obtained through instantiation of one of the basic interactors requires a considerable amount of general programming. For instance, the basic interactors available exhibit only part of the desired functionality needed in the example presented in the Introduction. Obtaining a behavior close to the one we want is highly non trivial.

Interactors that change objects by moving them, highlighting them, etc. have limitations about their reusability, since they need special kinds of objects to act upon. For example, an object to be moved by a *move-grow-interactor* needs a special slot called *box* that has as its value a list of four numbers that usually determine its position and dimensions. Moreover, the object must have constraints that fix its actual position and dimension according to the values in *box*. A system that allowed the accumulation of all the information that refers to an interactor's behavior in the own interactor would be more satisfactory, since it would be more reusable. We shall see

how Virtual Slots present this feature.

Description of the System

We start this section by defining a basic concept: a *nested slot* of an object is a slot of another object that in turn is the value of a (possibly nested) slot of the original object. For example, in the folder manager, the filling-style of the icon that represents a folder is a nested slot of the window that displays that icon.

The Virtual Slots system allows the user to specify the value that nested slots of objects will have, even before the necessary intermediate objects are determined. This information is kept in a declarative way in the object, and it is interpreted and used when the intermediate slots are set. The semantics of a specification saying that the Virtual Slot *[slot1] ... [slot(N+1)]* of a given <object> has a given <value> is that

```
IF <object> has a slot, [slot1],
       whose value is an object,
       that we denote by <slot1>
   AND <slot1>  has a slot, [slot2],
       whose value is an object,
       that we denote by <slot2>
   ...
   AND <slot(N-1)>  has a slot, [slotN],
       whose value is an object,
       that we denote by <slotN>
THEN the slot [slot(N+1)] of  <slotN>
     will have value <value>
```

Virtual Slots are in the spirit of traditional frame-based knowledge representation systems, which permit an efficient implementation through simple mechanisms of special kinds of production rules. Our point of view is that a production system must be implemented on top of a user interface tool only in case it is absolutely necessary due to the complexity of the tasks, as in [3].

For instance, the developer of the folder manager described in the Introduction can use a *moving-feedback-object* to represent the file being dragged (for example, a white bevelled rectangle), and a *static-feedback-object* to highlight the object the dragged file is on (for example, a red rectangle). Using Virtual Slots, the definition of the dragging interactor can specify that its nested slot *[static-feedback-object] [visible-p]* will have as value a formula that evaluates whenever it is needed to the current value of the predicate *accepts-p* evaluated with arguments *[first-obj-over] [object-type]* and *[current-obj-over] [object-type]*. As a consequence of this, the *static-feedback-object* will become visible whenever the object under the mouse accepts the file being moved. Similarly, the definition can specify that the *static-feedback-object* adjusts its size to that of the object under the mouse, and that the *moving-feedback-object* will follow the mouse. Finally, sound generation can be specified as a side effect of the object highlighting.

The above specification can be part of the definition of a more general interactor that will allow arbitrary moving and static feedback objects, an arbitrary accepting function and an arbitrary action to be performed in case of a drop. This interactor will be easily reusable in other applications. This degree of reusability is not present in current systems. The key fact that allows this level of reusability using the Virtual Slots system is that all the constraints are specified in the abstract object that is to be stored in a library.

Finally, we shall give an additional example that will illustrate further our techniques. It corresponds to a simple tool for the edition of graphical objects. It is a ruler that can be attached to a component of an interface, and then it appears over one of its sides and adapts to its size. After this, any of the ends of the ruler can be dragged along the ruler direction, and the object the ruler is attached to changes its size accordingly. Again, specifying the behavior of this object by means of Virtual Slots is very simple, and, most important, all the behavior of the object is encapsulated within it, so it can be stored in a library.

Conclusions and Future Work

We have shown how Virtual Slots allow a high level of reusability in UI development by specifying constraints in abstract objects to be stored in a library. This technique permits the partial specification of qualities of objects without making complete commitments about the kind of objects to be used.

A simple prototype of Virtual Slots that includes the functionality described in this paper has been implemented in KR, on top of Garnet. We have plans to develop a more complete prototype with a bigger declarative expression (including declarations for components of multivalued slots) and to extend the system to handle UIs developed in Humanoid.

Virtual Slots are a part of KIISS, a project funded by the National Research Plan of Spain, under grant No. TIC93-0268.

REFERENCES

1. Hill, R. D., et. al.: *The Rendezvous Architecture and Language for Constructing Multi-User Applications.* ACM Transactions on Computer-Human interaction, 1(2), 1994.

2. Maloney, J.: *Using Constraints for User Interface Construction.* PhD thesis, Department of Computer Science and Engineering, University of Washington, August 1991.

3. Moriyon, R., Szekely, P., and Neches, R.: *Automatic Generation of Help from Interfaces Design Models.* In Proceedings CHI'94, pp. 225-231, April 1994.

4. Myers, B.A., et. al.: *Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces*. IEEE Computer 23(11), pp. 71-85, November 1990.

5. Sannella, M.: *SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction.* In Proceedings UIST'94, pp. 137-146, November 1994.

6. Szekely, P., Luo, P., and Neches, R.: *Beyond Interface Builders: Model-Based Interface Tools.* In Proceedings INTERCHI'93, pp. 383-390, April 1993.