Considerations for writing applications

Boi Sletterink

March 25, 1999

Abstract

This document describes guidelines for designing and writing multimodal applications using the Modal Input Management classes developed for the STIMULATE project. The MIM classes provide an interface for applications to use various modalities. The goal was to make it easier for designers and programmers to develop multimodal applications, and to share those devices that provide the input and output capabilities among applications.

The basic idea behind writing multimodal applications is designing it with an open mind. Don't just think of the modalities that are available to you at designtime. Instead, try to design your application such that the data and state it contains can be queried and manipulated in an abstract way that is not tailored for the usual mouse-keyboard-graphics interface.

1 The idea behind Java UI libraries

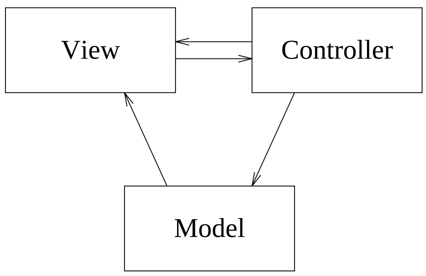

Figure 1: Original Model-View-Controller paradigm

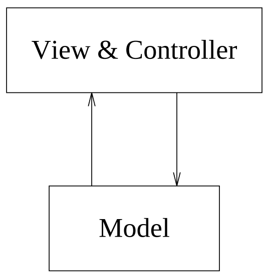

Figure 2: Java's implementation of MVC

Java's user interface libraries, AWT and Swing, are based on the Model-View-Controller model (Figure 1). The model is an abstract representation of the underlying problem and contains the data and methods for querying and manipulating it. The view filters and represents data from the model. The view is the media to present data to the user (feedback) and the media for the user to tell the computer something (input). The controller manipulates the model and the view. The controller reacts to the inputs provided by the view.

Since the view and controller are tightly coupled, both AWT and Swing have them represented by the same object (Figure 2). This greatly simplifies the design and implementation of the interfaces between them.

This triad, the model, view and controller, is usually seen as a component. This means that these three are a self-contained unit, which provides an abstraction, a way to manipulate it and to display it. This is what makes up JavaBeans; a single piece that containa data, optionally exhibits dynamic behaviour, can optionally be shown and manipulated.

Larger components can be hierarchically divided into smaller components. This means that the model consists of different submodels, and the view and controller consist of different subviews and subcontrollers. The embedding component can then define interactions between the different subcomponents. This makes it relatively easy to design larger and more complex components or applications.

A natural way to extend the user interfaces of Java applications, given this MVC-paradigm, would be to write new viewers and controllers that handle the other modalities. This concept is both elegant and adheres to the MVC-paradigm, but it does have a few problems with applications that are not designed for multimodal interfaces.

2 Common problems with real-world implementations

Usually, when people design the model, they define the data and operations with the standard mouse-keyboard-display in mind. While this approach definately works out for the systems most people have, it will not work for multimodal applications. This will result in insufficient information and operations in the model when it comes to other modalities. For example, the design will usually not include textual identifiers for every single object that may be manipulated.

While this is no problem with a mouse interface, it may be at least inconvenient when the user uses a speech recognizer. Also, the model may incorporate names for every object, but not include methods for efficiently searching through all objects (of a certain type) for a given name. This makes the speech view and controller more complicated and slower than necessary. It may even get too slow to be usable.

Another issue that may show up is that the user will probably want to manipulate views across modalities. For example, a user may want to say `"set scaling to 150%"` to (surprisingly) set the scaling of the current graphical view to 150 percent. Or the user may use a listbox to change the voice used for the text-to-speech feedback.

Also, views written for conventional programs will usually not be designed with the idea of multiple controllers in mind. Since there was no concept of multiple modalities at design time, the view's interfaces will only contain methods tailored for or even accessible to its accompanying controller, and vice versa. As mentioned, the view and controller are usually one, although some application designers keep them separated for larger applications. Keeping them separate may yield a cleaner and better maintainable design.

Yet another issue is that some components cannot be properly introspected. Their models and views are locked up inside and they are only manipulatable from the GUI. Composite components may not allow introspection of inner components, making it impossible to manipulate or query them. Neither models nor views or controllers are available.

3 Design recommendations

Following these observations, we'd like to make a few recommendations for designing multimodal applications. These recommendations are certainly not mandatory, or the only way, or even the best way to do it; the area of application-independent multimodal design methods is largely unexplored. Most research groups on multimodal human-computer interactions focus on their own specific application.

It is clear that the model needs to provide information across all possible dimensions. Include attributes like names for both objects and relationships between objects. Try to think of the model in an abstract way, think of basic operations on the model. Don't use the GUI, mouse and keyboard as the only reference. Expose data and methods using 3D coordinates if the model uses them internally; some input devices can provide these. Only providing mapped 2D coordinate-based data and methods will cripple these devices.

Make it possible for objects other than the provided view-controller pair to inspect and manipulate the model. Make sure submodels are accessible. Provide efficient search functions when dealing with collections of objects.

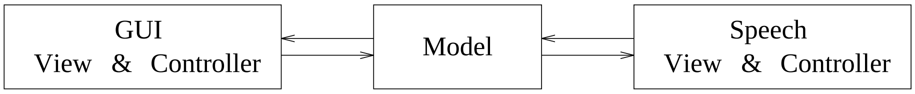

Figure 3: Multiple view-controller pairs for multiple modalities

You'll need a different view-controller pair for every modality (Figure 3). A view is not necessarily a visible entity; it is the generalized concept of getting information to and from the user. When merging several modalities into one view/controller implementation, don't assume that all of them will be available.

To facilitate integration of modal inputs, it may be a good idea to build a `presentation model` to accompany the usual `data model` (the data model being the model that was mentioned above). This presentation model represents the state of the views and controllers, such that one controller can introspect and manipulate other views and controllers.