J THE PROTOTYPE SYSTEMS APPROACH

No matter how perfect the internal system was, if the user interface did not work well, the system was judged to be unsatisfactory (Solomons, 2003). So, analysis have been conducted to determine what was needed to enable the use-cases for each prototype and its actor, then, created the physical user-interface designs that illustrated how users could use the systems to perform the use-cases, and finally, developed the proof-of-concept exploratory prototype systems.

This led to the software programming good practices that included the use of packages to group related classes or other modeling elements to reduce model's complexity; the analysis of dependencies between the packages to evaluate the system's architecture; the declaration of all attributes as privet and allow access to them only via a get and a set accessor-type operations; and the application of object-oriented concepts such as inheritance, polymorphism, and late binding.

This also led to the software patterns that describe a well-proved generic solution of a particular recurring design problem, but that could not specify a fully detailed solution (Schmidt et al. 2000, Buschmann et al. 2002). Therefore, for SACAPP's specific needs, a pattern template was developed (Table J.1), and then used the Model, View, Controller (MVC) software architecture pattern to give a clean way of organizing the code, plan the application, and make changes to it later (Figure J.1).

Table J.1 Pattern description template data

Pattern description template	
Name:	The name and a short summary of the pattern.
Context:	The situation in which the pattern may apply.
Problem:	The problem the pattern addresses, including a discussion of

	its associated forces.
Solution:	A fundamental solution principle underlying the pattern.
Dynamics:	Typical scenarios describing the run-time behaviour of the pattern.
Consequences:	The benefits the pattern provides and any potential liabilities.

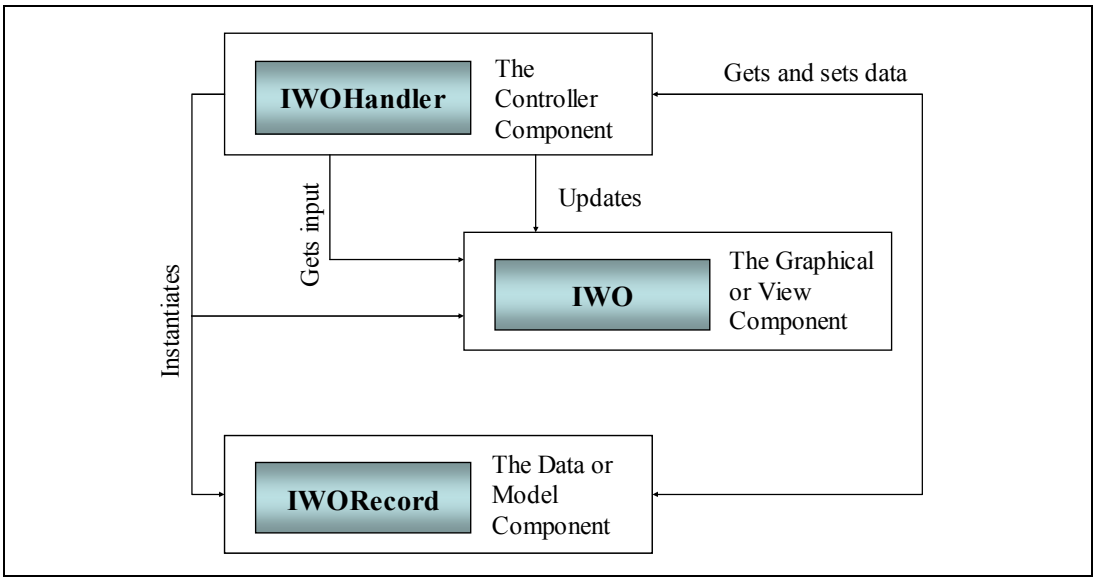

Figure J.1 The MVC divides the interactive application into three specified components. For example, the MVC for IWO pane contents: IWORecord.java (Model) stores the data that defines the component, has a constructor to build a IWORecord object that holds all the user input, and has the public get and set methods necessary to access individual data; IWO.java (View) is the visual representation of the data, creates and displays the GUI components for the user to enter information, and implements the update procedure; IWOHandler.java (Controller) takes user input on the view and changes the model, instantiates the IWO and IWORecord classes, and in addition, provides the functionality of the buttons for the IWO class by calling the public set and get methods of the other classes.