2.4 Interactive Systems

Today's systems allow a high degree of user interaction, mainly achieved with help of graphical user interfaces. The objective is to enhance the usability of an application. Usable software systems provide convenient access to their services, and therefore allow users to learn the application and produce results quickly.

When specifying the architecture of such systems, the challenge is to keep the functional core independent of the user interface. The core of interactive systems is based on the functional requirements for the system, and usually remains stable. User interfaces, however, are often subject to change and adaptation. For example, systems may have to support different user interface standards, customer-specific 'look and feel' metaphors, or interfaces that must be adjusted to fit into a customer's business processes. This requires architectures that support the adaptation of user interface parts without causing major effects to application-specific functionality or the data model underlying the software.

We describe two patterns that provide a fundamental structural organization for interactive software systems:

- The *Model-View-Controller* pattern (MVC) (125) divides an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface. A change-propagation mechanism ensures consistency between the user interface and the model.
- The *Presentation-Abstraction-Control* pattern (PAC) (145) defines a structure for interactive software systems in the form of a hierarchy of cooperating agents. Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspects of the agent from its functional core and its communication with other agents.

MVC provides probably the best-known architectural organization for interactive software systems. It was pioneered by Trygve Reenskaug [RWL96] and first implemented within the Smalltalk-80 environment [KP88]. It underlies many interactive systems and application frameworks for software systems with graphical user interfaces, such as MacApp [App89], ET++ [Gam91], and of course the Smalltalk libraries. Even Microsoft's Foundation Class Library [Kru96] follows the principles of MVC.

However, it is not our intention to explain the Smalltalk MVC implementation—many details of Smalltalk's MVC implementation are left out to give a clearer understanding of the underlying principles. Few readers will create a new framework for MVC, but are more likely to use an existing framework, or to partition their application following the key principles of MVC.

PAC is not used as widely as MVC, but this does not mean that it is not worth describing. As an alternative approach for structuring interactive applications, PAC is especially applicable to systems that consist of several self-reliant subsystems. PAC also addresses issues that MVC leaves unresolved, such as how to effectively organize the communication between different parts of the functional core and the user interface. PAC was first described by Joelle Coutaz [Cou87]. The first application of PAC was in the area of Artificial Intelligence [Cro85].