Module 6
Contact Session 02

Patterns – Part 2 Suppli

Interactive Systems

Harvinder S Jabbal
SSZG653 Software Architectures

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
  - the Layers pattern
  - the Pipes and Filters pattern
  - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
  - Broker
- and refers to two patterns in other categories,
  - Microkernel
  - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
  - the Model-View-Controller pattern (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

Interactive Systems

- This category comprises two patterns,
  - the Model-View-Controller pattern

    (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Interactive Systems

User Interaction

- 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.

user interface

- 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.

Model-View-Controller pattern (MVC)

innovate achieve lead

- MVC 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)

- PAC 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.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Model-View-Controller

Model-View-Controller

- The MVC 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.

Context

- Interactive applications with a flexible human-computer interface.

Problem

- User interfaces are especially prone to change requests. When you extend the functionality of an application, you must modify menus to access these new functions.
- A customer may call for a specific user interface adaptation, or a system may need to be ported to another platform with a different 'look and feel' standard.
- Even upgrading to a new release of your windowing system can imply code changes.
- The user interface platform of long-lived systems thus represents a moving target.

Problem

- Different users place conflicting requirements on the user interface.
- A typist enters information into forms via the keyboard.
- A manager wants to use the same system mainly by clicking icons and buttons.
- Consequently, support for several user interface paradigms should be easily incorporated.
- Building a system with the required flexibility is expensive and error-prone if the user interface is tightly interwoven with the functional core.
- This can result in the need to develop and maintain several substantially different software systems, one for each user interface implementation.
- Ensuing changes spread over many modules.

Influences

The following forces influence the solution:

- The same information is presented differently in different windows, for example, in a bar or pie chart.
- The display and behaviour of the application must reflect data manipulations immediately.
- Changes to the user interface should be easy, and even possible at run-time.
- Supporting different 'look and feel' standards or porting the user interface should not affect code in the core of the application.

Solution

- Model-View-Controller (MVC] was first introduced in the Smalltalk-80 programming environment.
- MVC divides an interactive application into the three areas:
    - processing,
    - output, and
    - input.

Model

- The model component encapsulates core data and functionality.
- The model is independent of specific output representations or input behaviour.

View

- View components display information to the user. A view obtains the data from the model.
- There can be multiple views of the model.

Controller

- Each view has an associated controller component.
- Controllers receive input, usually as events that encode mouse movement, activation of mouse buttons, or keyboard input.
- Events are translated to service requests for the model or the view.
- The user interacts with the system solely through controllers.

MVC

- The separation of the model from view and controller components allows multiple views of the same model.
- If the user changes the model via the controller of one view, all other views dependent on this data should reflect the changes.
- The model therefore notifies all views whenever its data changes.
- The views in turn retrieve new data from the model and update the displayed information.
- This change- propagation mechanism is described in the Publisher-Subscriber pattern

Structure: Model

- The model component contains the functional core of the application.
- It encapsulates the appropriate data, and exports procedures that perform application-specific processing.
- Controllers call these procedures on behalf of the user.
- The model also provides functions to access its data that are used by view components to acquire the data to be displayed.
- The change-propagation mechanism maintains a registry of the dependent components within the model.
- All views and also selected controllers register their need to be informed about changes.
- Changes to the state of the model trigger the change-propagation mechanism.
- The change-propagation mechanism is the only link between the model and the views and controllers.

CRC Diagram

Class Model	**Collaborators** • View • Controller
Responsibility • Provides functional core of the application. • Registers dependent views and controllers. • Notifies dependent components about data changes.	

Structure: View

- View components present information to the user.
- Different views present the information of the model in different ways.
- Each view defines an update procedure that is activated by the change- propagation mechanism.
- When the update procedure is called, a view retrieves the current data values to be displayed from the model, and puts them on the screen.
- During initialization all views are associated with the model, and register with the change-propagation mechanism.
- Each view creates a suitable controller.
- There is a one-to-one relationship between views and controllers.
- Views often offer functionality that allows controllers to manipulate the display.
- This is useful for user-triggered operations that do not affect the model, such as scrolling.

CRC Diagram

Class View	**Collaborators**
Responsibility • Creates and initializes its associated controller. • Displays information to the user. • Implements the update procedure. • Retrieves data from the model.	• Controller • Model

Structure: Controller

- The controller components accept user input as events.
- How these events are delivered to a controller depends on the user interface platform.
- For simplicity, let us assume that each controller implements an event-handling procedure that is called for each relevant event.
- Events are translated into requests for the model or the associated view.
- If the behaviour of a controller depends on the state of the model, the controller registers itself with the change-propagation mechanism and implements an update procedure.
- For example, this is necessary when a change to the model enables or disables a menu entry.

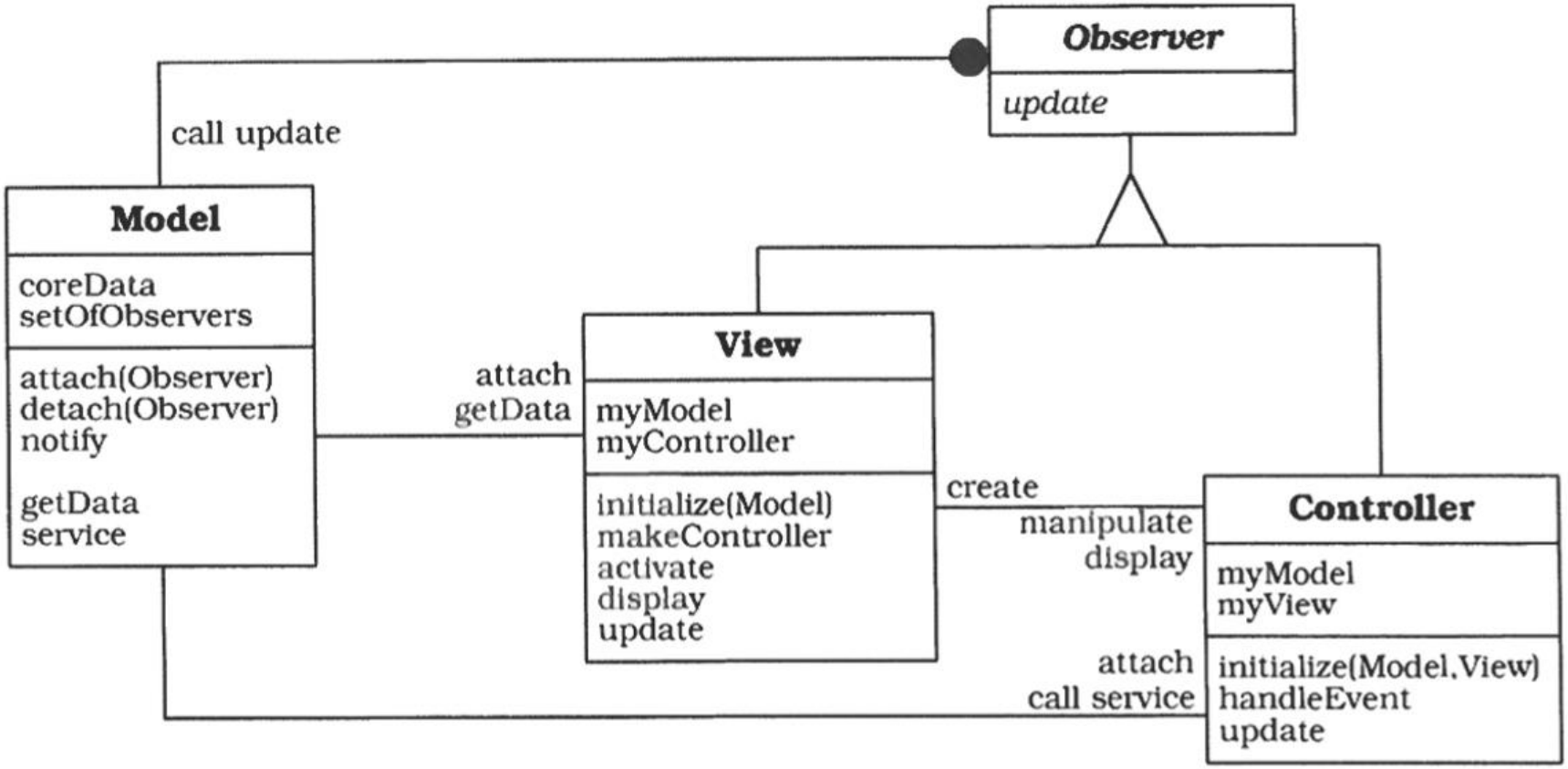
CRC Diagram

Class Controller	***Collaborators***
Responsibility • Accepts user input as events. • Translates events to service requests for the model or display requests for the view. • Implements the update procedure, if required.	• View • Model

Popular Structural Variations

- Object Oriented
    - implementation of MVC would define a separate class for each component
- C++
    - view and controller classes share a common parent that defines the update interface.
- Smalltalk.
    - the class Object dehes methods for both sides of the change- propagation mechanism.
    - A separate class Observer is not needed.

Structure: C++



Implementation

innovate achieve lead

1. Separate human-computer interactionfrom corefunctionality
2. Implement the change-propagationm echanism
3. Design and implement the views
4. Design and implement the controllers
5. Design and implement the view-controller relationship
6. Implement the set-up of MVC
7. Dynamic view creation
8. 'Pluggable controllers
9. Infrastructure for hierarchical views and controllers
10. Further decoupling from system dependencies

Variants

- Document-View
  - This variant relaxes the separation of view and controller.
  - In several GUI platforms, window display and event handling are closely interwoven.
  - For example, the X Window System reports events relative to a window.
  - You can combine the responsibilities of the view and the controller from MVC in a single component by sacrificing exchangeability of controllers.

Known Usages

- Smalltalk
  - The VisualWorks Smalltalk environment supports different 'look and feel' standards by decoupling view and controllers via display and sensor classes
- MFC (Microsoft Foundation Class Library)
  - The Document-View variant of the Model-View-Controller pattern is integrated in the Visual C++ environment
- ET++
  - Establishes 'look and feel' independence by defining a class Window port that encapsulates the user interface platform dependencies, in the same way as do our display and sensor classes. Uses document-view variant.

Consequences: Benefits

- Multiple views of the same model
- Synchronized views.
- 'Pluggable' views and controllers.
- Exchangeability of 'look and feel’.
- Framework Potential.

Consequences: Liabilities

- Increased complexity.
- Potential for excessive number of updates.
- Intimate connection between view and controller.
- Close coupling of views and controllers to a model.
- Inefficiency of data access in view.
- Inevitability of change to view and controller when porting.
- Difficulty of using MVC with modern user-interface took.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

PAC: Presentation-Abstraction-Control

Presentation-Abstraction-Control

- The Presentation-Abstraction-Control architectural pattern (PAC) 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

Context

- Development of an interactive application with the help of agents.
- In the context of this pattern an agent denotes an information-processing component that includes event receivers and transmitters, data structures to maintain stale, and a processor that handles incoming events, updates its own state, and that may produce new events.
- Agents can be as small as a single object, but also as complex as a complete software system.
- We use the terms agent and PAC agent as synonyms in this pattern description

Problem

- Interactive systems can often be viewed as a set of cooperating agents.
- Agents specialized in human-computer interaction accept user input and display data.
- Other agents maintain the data model of the system and offer functionality that operates on this data.
- Additional agents are responsible for diverse tasks such as error handling or communication with other software systems.
- Besides this horizontal decomposition of system functionality, we often encounter a vertical decomposition.

Example

- Production planning systems (PPS) , for example, distinguish between production planning and the execution of a previously specified production plan.
- For each of these tasks separate agents can be defined.
- In such an architecture of cooperating agents, each agent is specialized for a specific task, and all agents together provide the system functionality.
- This architecture also captures both a horizontal and vertical decomposition.

Forces

- Agents often maintain their own state and data.
  - For example, in a PPS system, the production planning and the actual production control may work on different data models, one tuned for planning and simulation and one performance-optimized for efficient production.
  - However, individual agents must effectively cooperate to provide the overall task of the application.
  - To achieve this, they need a mechanism for exchanging, data, messages, and events.
- Interactive agents provide their own user interface, since their respective human-computer interactions often differ widely.
  - For example, entering data into spreadsheets is done using keyboard input, while the manipulation of graphical objects uses a pointing device.
- Systems evolve over time.
  - Their presentation aspect is particularly prone to change.
  - The use of graphics, and more recently, multi-media features, are examples of pervasive changes to user interfaces.
  - Changes to individual agents, or the extension of the system with new agents, should not affect the whole system.

Solution

- Structure the interactive application as a tree-like hierarchy of PAC agents.
- There should be one top-level agent, several intermediate-level agents, and even more bottom-level agents.
- Every agent is responsible for a specific aspect of the application's functionality, and consists of three components: presentation, abstraction, and control.
- The whole hierarchy reflects transitive dependencies between agents.
- Each agent depends on all higher-level agents up the hierarchy to the top-level agent.

Agents

- The agent's presentation component provides the visible behaviour of the PAC agent.
- Its abstraction component maintains the data model that underlies the agent, and provides functionality that operates on this data.
- Its control component connects the presentation and abstraction components, and provides functionality that allows the agent to communicate with other PAC agents.

The top-level PAC agent

- The top-level PAC agent provides the functional core of the system.
- Most other PAC agents depend or operate on this core. Furthermore, the top-level PAC agent includes those parts of the user interface that cannot be assigned to particular subtasks, such as menu bars or a dialog box displaying information about the application.

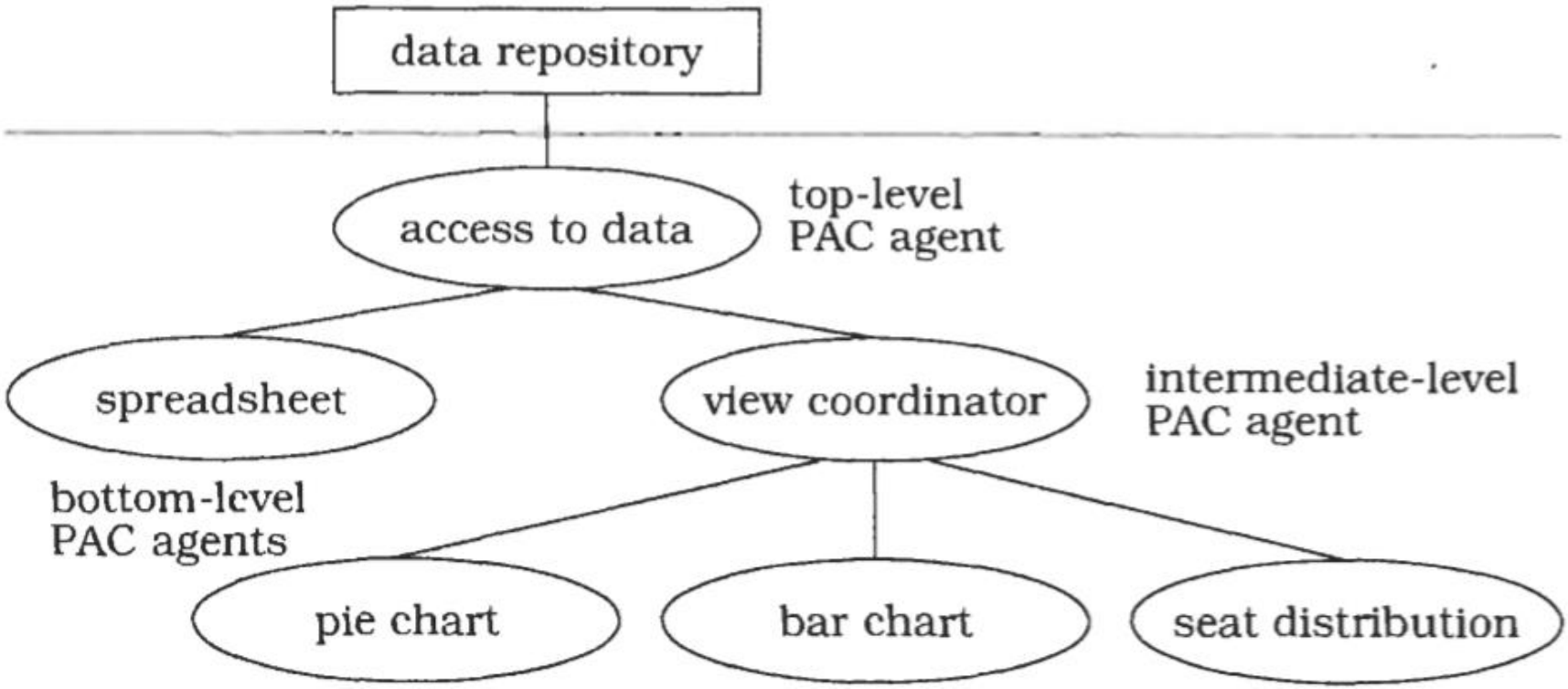
Bottom-level PAC agents

- Bottom-level PAC agents represent self-contained semantic concepts on which users of the system can act, such as spreadsheets and charts.
- The bottom-level agents present these concepts to the user and support all operations that users can perform on these agents, such as zooming or moving a chart.

Intermediate-level PAC agents

- Intermediate-level PAC agents represent either combinations of, or relationships between, lower-level agents.
    - For example, an intermediate-level agent may maintain several views of the same data, such as a floor plan and an external view of a house in a CAD system for architecture.

Typical Hierarchy of Agents



Structure: Top Level Agent (component)

- The main responsibility of the top-level PAC agent is to provide the global data model of the software.
- This is maintained in the abstraction component of the top-level agent.
- The interface of the abstraction component offers functions to manipulate the data model and to retrieve information about it.
- The representation of data within the abstraction component is media-independent.
- For example, in a CAD system for architecture, walls, doors, and windows are represented in centimetres or inches that reflect their real size, not in pixels for display purposes.
- This media-independency supports adaptation of the PAC agent to different environments without major changes in its abstraction component.

Structure: Top Level Agent (presentation)

innovate achieve lead

- The presentation component of the top-level agent often has few responsibilities.
- It may include user-interface elements common to the whole application.
- In some systems, such as the network traffic manager, there is no top-level presentation component at all.

Structure: Top Level Agent (control)

- The control component of the top-level PAC agent has three responsibilities:

1
   - It allows lower-level agents to make use of the services of the top-level agents, mostly to access and manipulate the global data model.
   - Incoming service requests from lower-level agents are forwarded either to the abstraction component or the presentation component.

2
   - It coordinates the hierarchy of PAC agents.
   - It maintains information about connections between the top-level agent and lower-level agents.
   - The control component uses this information to ensure correct collaboration and data exchange between the top-level agent and lower-level agents.

3
   - It maintains information about the interaction of the user with the system.
   - For example, it may check whether a particular operation can be performed on the data model when triggered by the user.
   - It may also keep track of the functions called to provide history or undo/redo services for operations on the functional core.

Bottom Level Agent

- Bottom-level PAC agents represent a specific semantic concept of the application domain, such as a mailbox in a network traffic management system or a wall in a mobile robot system.
- This semantic concept may be as low-level as a simple graphical object such as a circle, or as complex as a bar chart that summarizes all the data in the system.

Bottom Level Agent (presentation)

- The presentation component of a bottom-level PAC agent presents a specific view of the corresponding semantic concept, and provides access to all the functions users can apply to it.
- Internally, the presentation component also maintains information about the view, such as its position on the screen.

Bottom Level Agent (abstraction)

- The abstraction component of a bottom-level PAC agent has a similar responsibility as the abstraction component of the top-level PAC agent, maintaining agent-specific data.
- In contrast to the abstraction component of the top-level agent, however, no other PAC agents depend on this data.

Bottom Level Agent (control)

- The control component of a bottom-level PAC agent maintains consistency between the abstraction and presentation components, thereby avoiding direct dependencies between them.
- It serves as an adapter and performs both interface and data adaptation.
- The control component of bottom-level PAC agents communicates with higher-level agents to exchange events and data.
- Incoming events-such as a 'close window' request-are forwarded to the presentation component of the bottom-level agent, while incoming data is forwarded to its abstraction component.
- Outgoing events and data, for example error messages, are sent to the associated higher-level agent.

Bottom Level Agent

- Concepts represented by bottom-level PAC agents, such as the bar and pie charts in the example, are atomic in the sense that they are the smallest units a user can manipulate. The users can only operate on the bar chart as a whole, for instance by changing the scaling factor of the y-axis. They cannot, for example, resize an individual bar of a bar chart.
- Bottom-level PAC agents are not restricted to providing semantic concepts of the application domain. You can also specify bottom-level agents that implement system services.
- For example, there may be a communication agent that allows the system to cooperate with other applications and to monitor this cooperation

Intermediate-Level PAC agents (composition)

innovate achieve lead

- Intermediate-Level PAC agents can fulfil two different roles:
- composition and coordination
- When, for example, each object in a complex graphic is represented by a separate PAC agent, an intermediate-level agent groups these objects to form a composite graphical object.
- The intermediate-level agent defines a new abstraction, whose behaviour encompasses both the behaviour of its components and the new characteristics that are added to the composite object.

Intermediate-Level PAC agents (coordination)

- The second role of an intermediate-level agent is to maintain consistency between lower- level agents, for example when coordinating multiple views of the same data.
- The abstraction component maintains the specific data of the intermediate-level PAC agent. The presentation component implements its user interface. The control component has the same responsibilities of the control components of bottom-level PAC agents and of the top-level PAC agent.

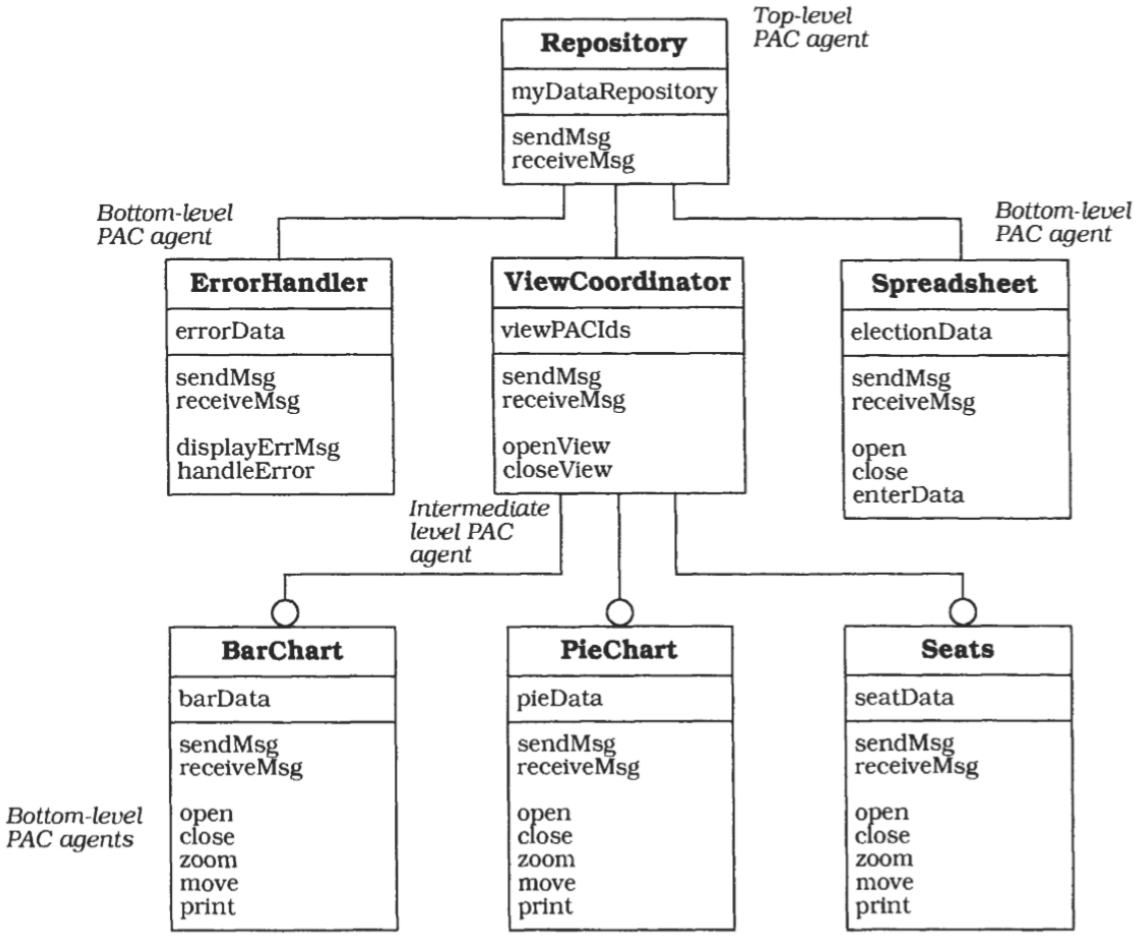
CRC Diagrams

Class	Collaborators
Top-level Agent	• Intermediate-level Agent • Bottom-level Agent
Responsibility • Provides the functional core of the system. • Controls the PAC hierarchy.	

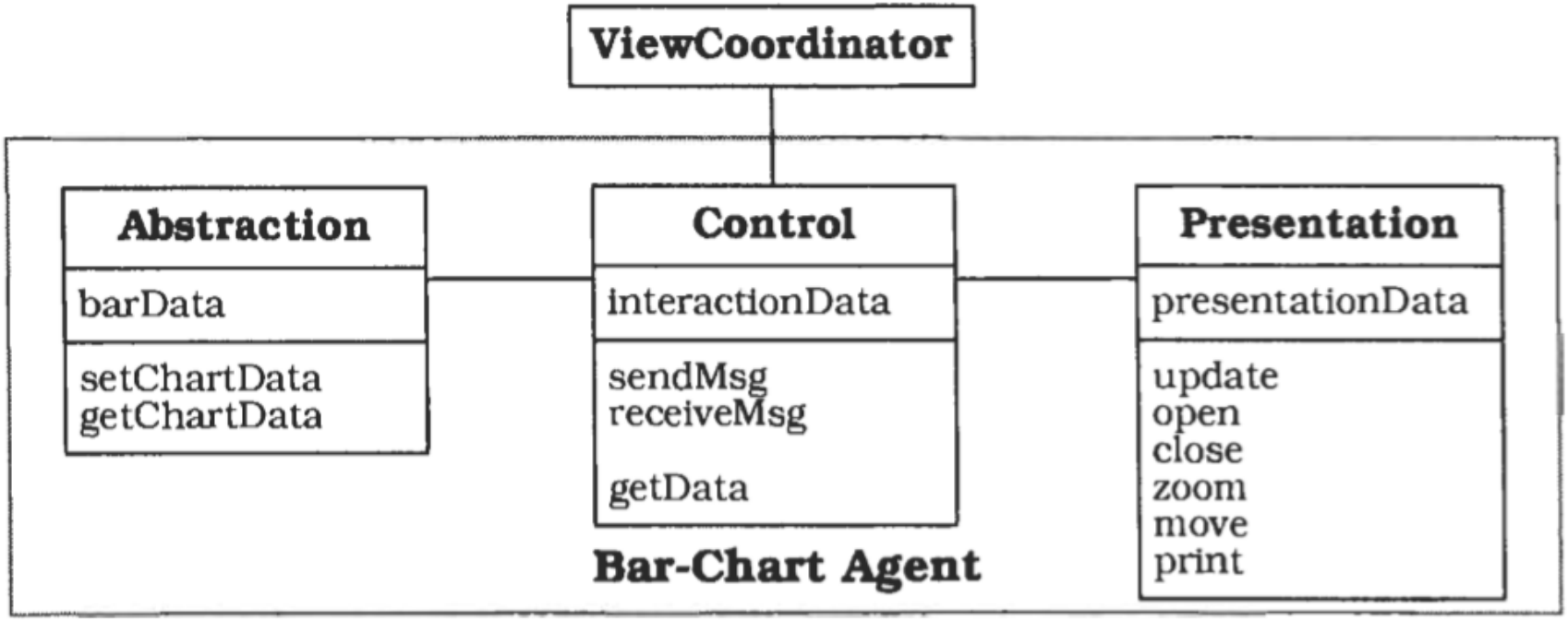
Class	Collaborators
Interm. -level Agent	• Top-level Agent • Intermediate-level Agent • Bottom-level Agent
Responsibility • Coordinates lower-level PAC agents. • Composes lower-level PAC agents to a single unit of higher abstraction.	

Class	Collaborators
Bottom-level Agent	• Top-level Agent • Intermediate-level Agent
Responsibility • Provides a specific view of the software or a system service, including its associated human-computer interaction.	

Typical PAC Object Model



Internal Structure of a PAC Agent



Implementation

1. Define a model of the application.
2. Define a general strategy for organizing the PAC hierarchy
3. Specify the top-level PAC agent.
4. Specify the bottom-level PAC agents
5. Specih bottom-level PAC agents for system services.
6. Specify intermediate-level PAC agents to compose lower-level PAC agents
7. Specify intermediate-level PAC agents to coordinate lower-level PAC agents.
8. Separate core functionality from human-computer interaction
9. Provide the external interface.

Variants

- Many large applications-especially interactive ones-are multi-user systems. Multi-tasking is thus a major concern when designing such software systems.

Variants

- PAC agents as active objects

- PAC agents as processes.

Known Usages

- Network Traffic Management.
- Mobile Robot.

Consequences: benefits

- Separation of concerns
- Support for change and extension
- Support for multi-tasking

Consequences: Liabilities

- Increased system complexity.
- Complex control component
- Applicability.

Thank you

innovate achieve lead

The END