Presentation-Abstraction-Control (PAC) Pattern

PAC Introduction

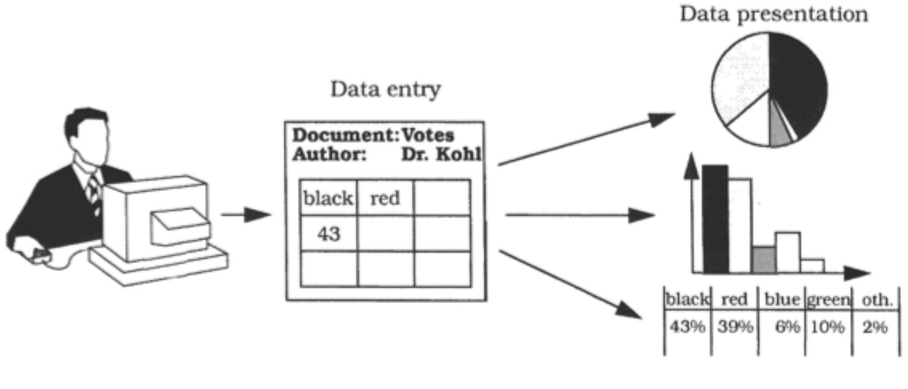
- The Presentation-Abstraction-Control architectural pattern (PAC) defines a structure for interactive software systems in the form of a hierarchy of cooperating agents.
- Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction, and control.
- The PAC *abstraction* component corresponds roughly to the *model* component of MVC.
- The *presentation* component in PAC is a combination of the *view* and *control* components in MVC.
- The *control* component of PAC mediates between agents and has no equivalent in MVC.

PAC Agents

- In the context of the Presentation-Abstraction-Control pattern, an agent is an information processing component that includes:
  - Event receivers and transmitters.
  - Data structures to maintain state.
  - A processor that handles incoming events, updates its own state, and may produce new events.
- Agents can be a s small as a single object, but also as complex as a complete software system.

Example

- Consider a simple information system for political elections with proportional representation. This offers a spreadsheet for entering data and several kinds of tables and charts for presenting current standings. Users interact with the software through a graphical interface.
- Different versions, however, adapt the user interface to specific needs. For example, one version supports additional views of the data, such as the assignment of parliament seats to political parties.



Context

- 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.
- 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 captures both a horizontal and vertical decomposition.

Forces

- Agents often maintain their own state and data.
  - For example, in a production planning 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.

Forces (2)

- 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.

Solution (2)

- The agent's presentation component provides the visible behavior 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.

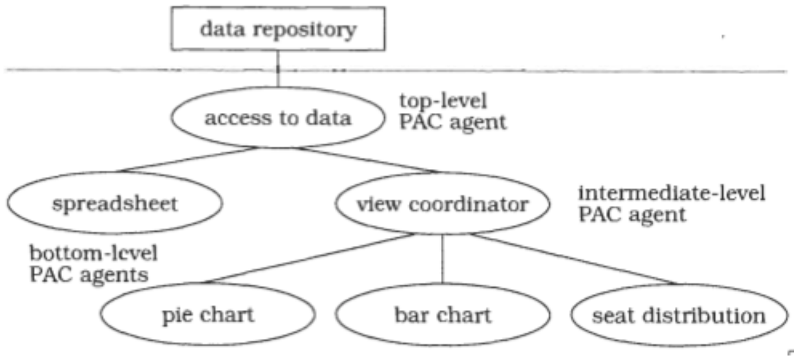
Solution (3)

- The *top-level* PAC agent provides the functional core of the system.
  - 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.
  - Most other PAC agents depend or operate on this core.
- *Bottom-level* PAC agents represent self-contained 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 represent either combinations of, or relationships between, lower-level agents, e.g. a floor plan and an external view of a house in a CAD system for architecture.

Example (2)

- Our information system for political elections defines a top-level PAC agent that provides access to the data repository underlying the system. (The data repository itself is not part of the application.)
- At the bottom level we specify four PAC agents: one spreadsheet agent for entering data, and three view agents for each type of diagram for representing the data.
- The application has one intermediate-level PAC agent. This coordinates the three bottom-level view agents and keeps them consistent. The spreadsheet agent is directly connected to the top-level PAC agent.
- Users of the system only interact with bottom-level agents.

Example (3)



Top-Level Agent

- The abstraction component of the top-level agent provides the global data model of the software.
  - This is the main responsibility 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 centimeters 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.

Top-Level Agent (2)

- The presentation component of the top-level agent often has few responsibilities. In some systems, there is no top-level presentation component at all.
- The control component of the top-level PAC agent has three responsibilities:
  - 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. Requests from lower-levels are forwarded to the abstraction or presentation component.
  - It coordinates the hierarchy of PAC agents. It maintains information about connections between the top-level agent and lower-level agents.
  - 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.

Example: Top-Level Agent

- In our example information system for political elections, the abstraction component of the top-level PAC agent provides an application-specific interface to the underlying data repository.
  - It implements functions for reading and writing election data.
  - It also implements all functions that operate on the election data, such as algorithms for calculating projections and seat distributions.
  - It further includes functions for maintaining data, such as those for updating and consistency checking.
- The control component organizes communication and cooperation with lower-level agents, namely the view coordinator and spreadsheet agents.
- This top-level PAC agent does not include a presentation component.

Bottom-Level Agents

- 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 Agents (2)

- The *presentation* component of a bottom-level PAC agent presents a specific view of the corresponding 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.
- The *abstraction* component 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 Agents (3)

- The *control* component of a bottom-level PAC agent maintains consistency between the abstraction and presentation components, thereby decoupling them.
- 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.
  - Incoming data is forwarded to its abstraction component.
  - Outgoing events and data, e.g. error messages, are sent to the associated higher-level agent.

Example: Bottom-Level Agent

- Consider a bar-chart agent in our information system for political elections.
- Its abstraction component saves the election data presented in the chart, and maintains chart-specific information such as the order of presentation for the data.
- The presentation component is responsible for displaying the bar chart in a window, and for providing all the functions that can be applied to it, such as zooming, moving, and printing.
- The control component serves as a level of indirection between the presentation and abstraction components. The control component is also responsible for the bar-chart agent's communication with the view coordinator agent.

Intermediate-Level Agents

- Intermediate-Level PAC agents can fulfill 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 behavior encompasses both the behavior of its components and the new characteristics that are added to the composite object.
- 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.

Intermediate-Level Agents (2)

- 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.

Example: Intermediate-Level Agent

- The *view coordinator* in our example system for political elections is a typical intermediate-level agent.
- Its *presentation* component provides a palette that allows users to create views of the election data, such as bar or pie charts.
- The *abstraction* component maintains data about all currently-active views, each of which is realized by its own bottom-level agent.
- The main responsibility of the *control* component is to coordinate all subordinate agents.
  - It forwards incoming notifications about data model changes taking place in the top-level agent to the bottom-level agents, and organizes their update.
  - It also includes functionality to create and delete bottom-level agents on user request.

Behavior: Example Scenario 1

1. A user asks the presentation component of the view coordinator agent to open a new bar chart.
2. The control of the view coordinator agent instantiates the desired bar-chart agent.
3. The view coordinator agent sends an 'open' event to the control component of the new bar-chart agent.
4. The control component of the bar-chart agent first retrieves data from the top-level PAC agent. The view coordinator agent mediates between bottom and top-level agents. The data returned to the bar-chart agent is saved in its abstraction component. Its control component then calls the presentation component to display the chart.
5. The presentation component creates a new window on the screen, retrieves data from the abstraction component by requesting it from the control component, and finally displays it within the new window.

Behavior: Example Scenario 2

1. The user enters new data into a spreadsheet. The control component of the spreadsheet agent forwards this data to the top-level PAC agent.
2. The control component of the top-level PAC agent receives the data and tells the top-level abstraction to change the data repository accordingly. The abstraction component of the top-level agent asks its control component to update all agents that depend on the new data. The control component of the top-level PAC agent therefore notifies the view coordinator agent.
3. The control component of the view coordinator agent forwards the change notification to all view PAC agents it is responsible for coordinating.
4. As in the previous scenario, all view PAC agents then update their data and refresh the image they display.

PAC Benefits

- Separation of concerns.
  - Different semantic concepts in the application domain are represented by separate agents.
  - Each agent maintains its own state and data, coordinated with, but independent of other PAC agents.
  - Individual PAC agents also provide their own user interface.
  - This allows the development of a dedicated data model and user interface for each semantic concept or task within the application, independently of other semantic concepts or tasks.
- Support for change and extension.
  - Changes within the presentation or abstraction components of a PAC agent do not affect other agents in the system.
  - New agents are easily integrated into an existing PAC architecture without major changes to existing PAC agents.

PAC Benefits (2)

- Support for multitasking.
  - PAC agents can be distributed easily to different threads, processes, or machines. Extending a PAC agent with appropriate inter-process communication functionality only affects its control component.
  - Multitasking also facilitates multi-user applications. For example, in our information system a newscaster can present the latest projection while data entry personnel update the data base with new election data.

PAC Liabilities

- Increased system complexity.
  - The implementation of every semantic concept within an application as its own PAC agent may result in a complex system structure.
  - If the level of granularity is too fine, the system could drown in a sea of agents. Agents must also be coordinated and controlled, which requires additional coordination agents.
- Complex control component.
  - The quality of the control component implementations is therefore crucial to an effective collaboration between agents, and therefore for the overall quality of the system architecture.
- Efficiency.
  - The overhead in the communication between PAC agents may impact system efficiency especially if agents are distributed.
- Applicability.
  - The smaller the atomic semantic concepts of an application are, and the more similar their user interfaces, the less applicable this pattern is.