Module 6
Contact Session 07
Patterns – Part 2

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Model View Controller

Model-View-Controller Pattern

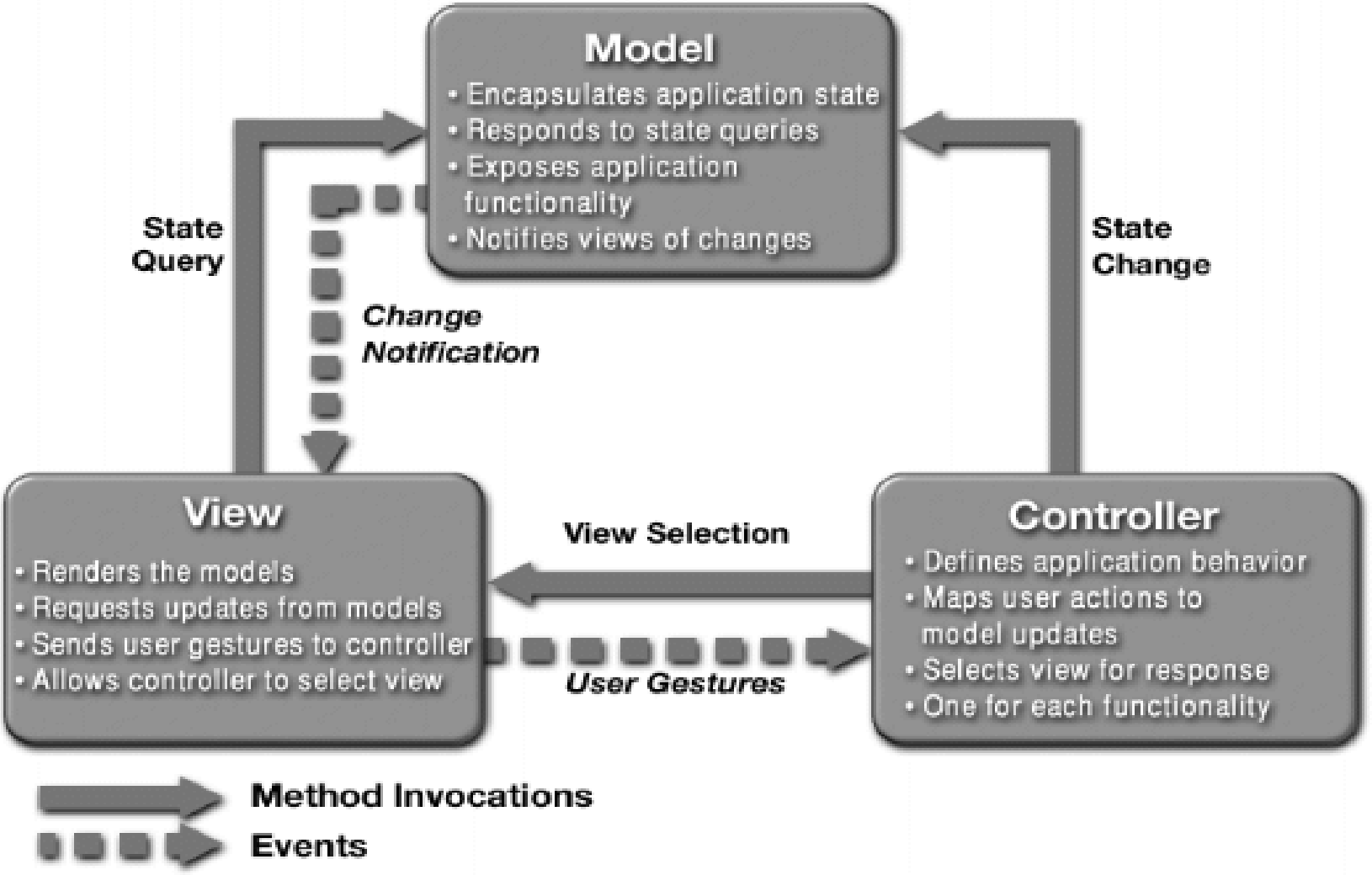
Context: User interface software is typically the most frequently modified portion of an interactive application. Users often wish to look at data from different perspectives, such as a bar graph or a pie chart. These representations should both reflect the current state of the data.

Problem: How can user interface functionality be kept separate from application functionality and yet still be responsive to user input, or to changes in the underlying application's data? And how can multiple views of the user interface be created, maintained, and coordinated when the underlying application data changes?

Solution: The model-view-controller (MVC) pattern separates application functionality into three kinds of components:

- A model, which contains the application's data
- A view, which displays some portion of the underlying data and interacts with the user
- A controller, which mediates between the model and the view and manages the notifications of state changes

MVC Example


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

MVC Solution - 1

Overview: The MVC pattern breaks system functionality into three components: a model, a view, and a controller that mediates between the model and the view.

Elements:

- The *model* is a representation of the application data or state, and it contains (or provides an interface to) application logic.
- The *view* is a user interface component that either produces a representation of the model for the user or allows for some form of user input, or both.
- The *controller* manages the interaction between the model and the view, translating user actions into changes to the model or changes to the view.

MVC Solution - 2

Relations: The *notifies* relation connects instances of model, view, and controller, notifying elements of relevant state changes.

Constraints:

- There must be at least one instance each of model, view, and controller.
- The model component should not interact directly with the controller.

Weaknesses:

- The complexity may not be worth it for simple user interfaces.
- The model, view, and controller abstractions may not be good fits for some user interface toolkits.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Service Oriented Architecture Pattern

Service Oriented Architecture Pattern

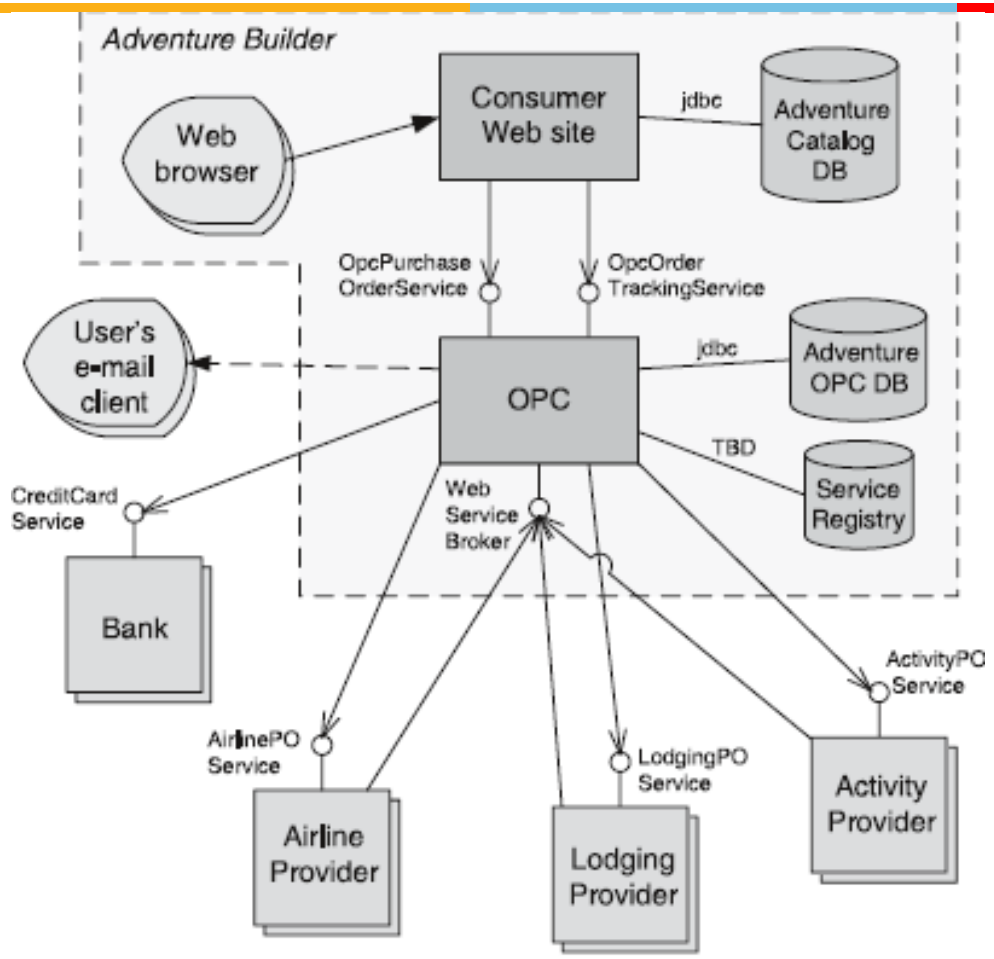
innovate achieve lead

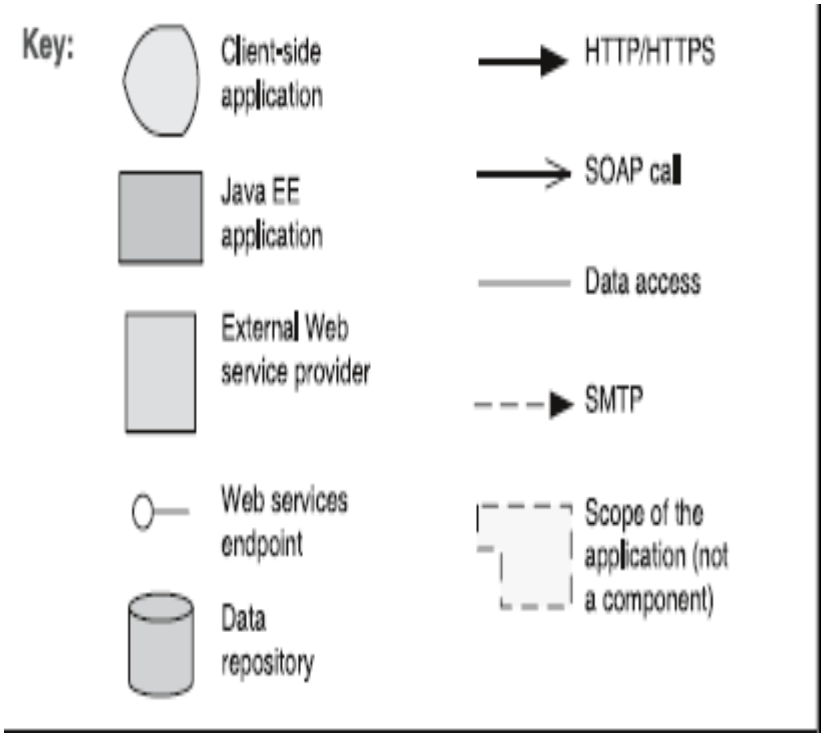
- **Context**: A number of services are offered (and described) by service providers and consumed by service consumers. Service consumers need to be able to understand and use these services without any detailed knowledge of their implementation.

Problem: How can we support interoperability of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across the Internet?

Solution: The service-oriented architecture (SOA) pattern describes a collection of distributed components that provide and/or consume services.

Service Oriented Architecture Example





Service Oriented Architecture Solution - 1

Overview: Computation is achieved by a set of cooperating components that provide and/or consume services over a network.

Elements:

- Components:
  - *Service providers,* which provide one or more services through published interfaces.
  - *Service consumers,* which invoke services directly or through an intermediary.
  - *Service providers* may also be service consumers.
- *ESB,* which is an intermediary element that can route and transform messages between service providers and consumers.
- *Registry of services,* which may be used by providers to register their services and by consumers to discover services at runtime.
- *Orchestration server,* which coordinates the interactions between service consumers and providers based on languages for business processes and workflows.

Service Oriented Architecture Solution - 2

innovate achieve lead

– Connectors:

- *SOAP connector,* which uses the SOAP protocol for synchronous communication between web services, typically over HTTP.
- *REST connector,* which relies on the basic request/reply operations of the HTTP protocol.
- *Asynchronous messaging connector,* which uses a messaging system to offer point-to-point or publish-subscribe asynchronous message exchanges.

Service Oriented Architecture Solution - 3

Relations: Attachment of the different kinds of components available to the respective connectors

Constraints: Service consumers are connected to service providers, but intermediary components (e.g., ESB, registry, orchestration server) may be used.

Weaknesses:

- SOA-based systems are typically complex to build.
- You don’t control the evolution of independent services.
- There is a performance overhead associated with the middleware, and services may be performance bottlenecks, and typically do not provide performance guarantees.