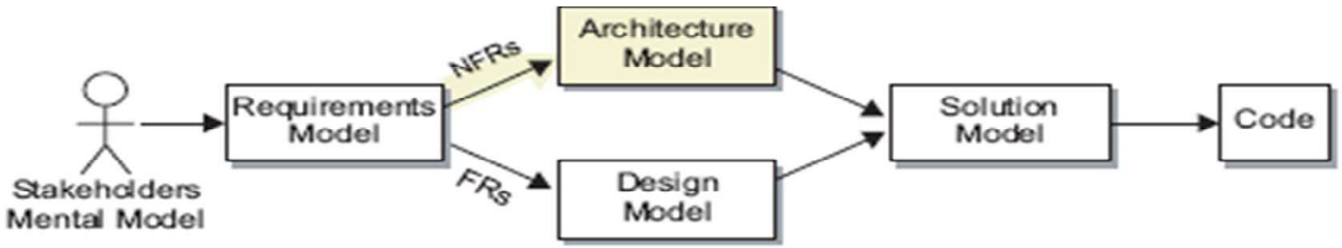
9 Introducing Architectural Concepts and Diagrams

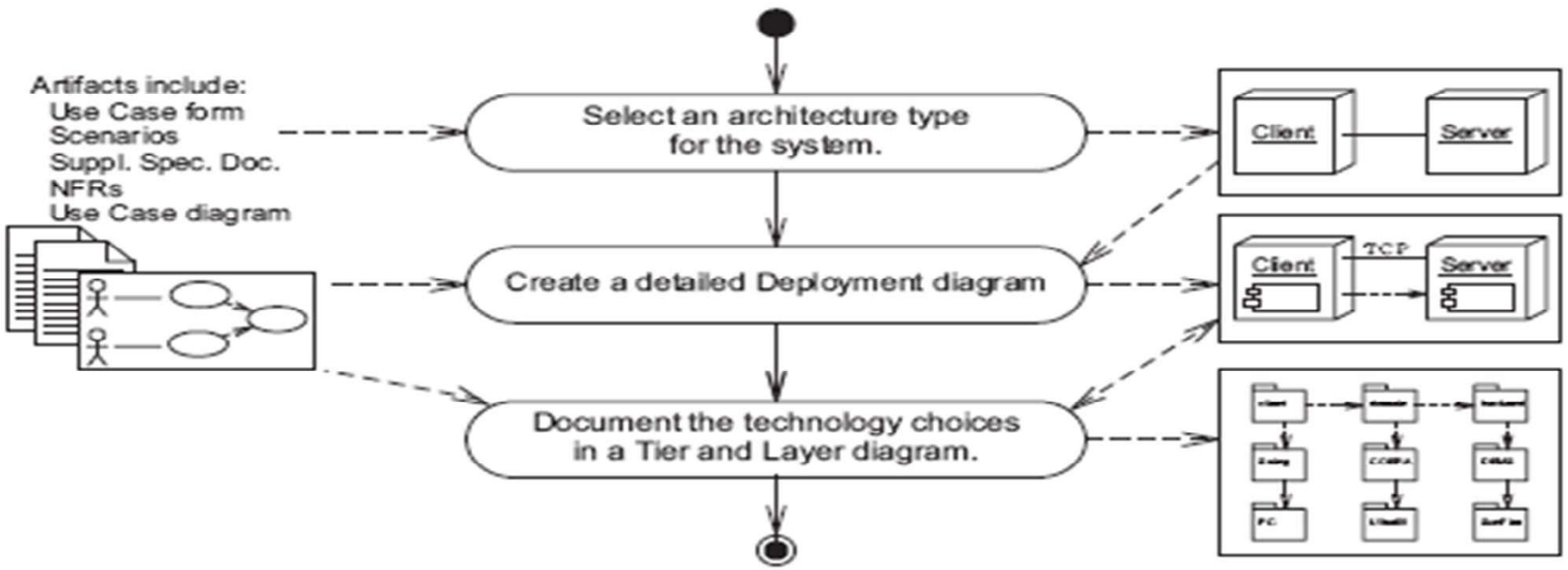
Objectives

After completing this lesson, you should be able to do the following:

- Difference between architecture and design:
  - Design produces components to implement a use case.
  - Architecture provides a template into which the designed components are realized.
- You can model the architecture using:
  - Deployment diagrams
  - Component diagrams
  - Packages
  - Tiers and layers

Process Map


Stakeholders
Mental Model
Requirements
Model
NFRs
FRs
Architecture
Model
Design
Model
Solution
Model
Code

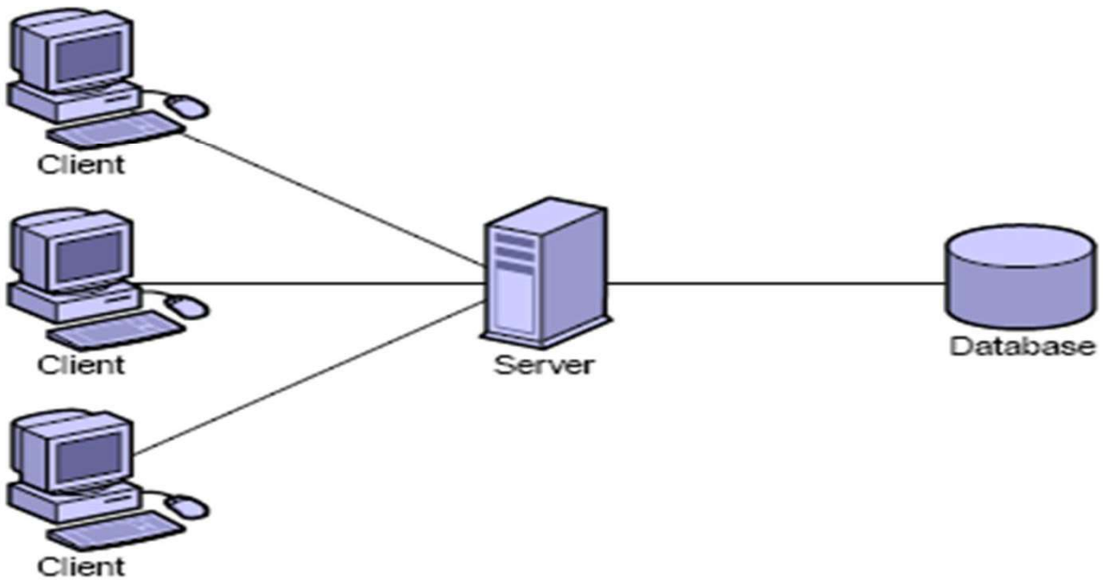

Artifacts include:
Use Case form
Scenarios
Suppl. Spec. Doc.
NFRs
Use Case diagram
Select an architecture type
for the system.
Create a detailed Deployment diagram
Document the technology choices
in a Tier and Layer diagram.
Client
Server
Client
TCP
Server

Justifying the Need for the Architect Role

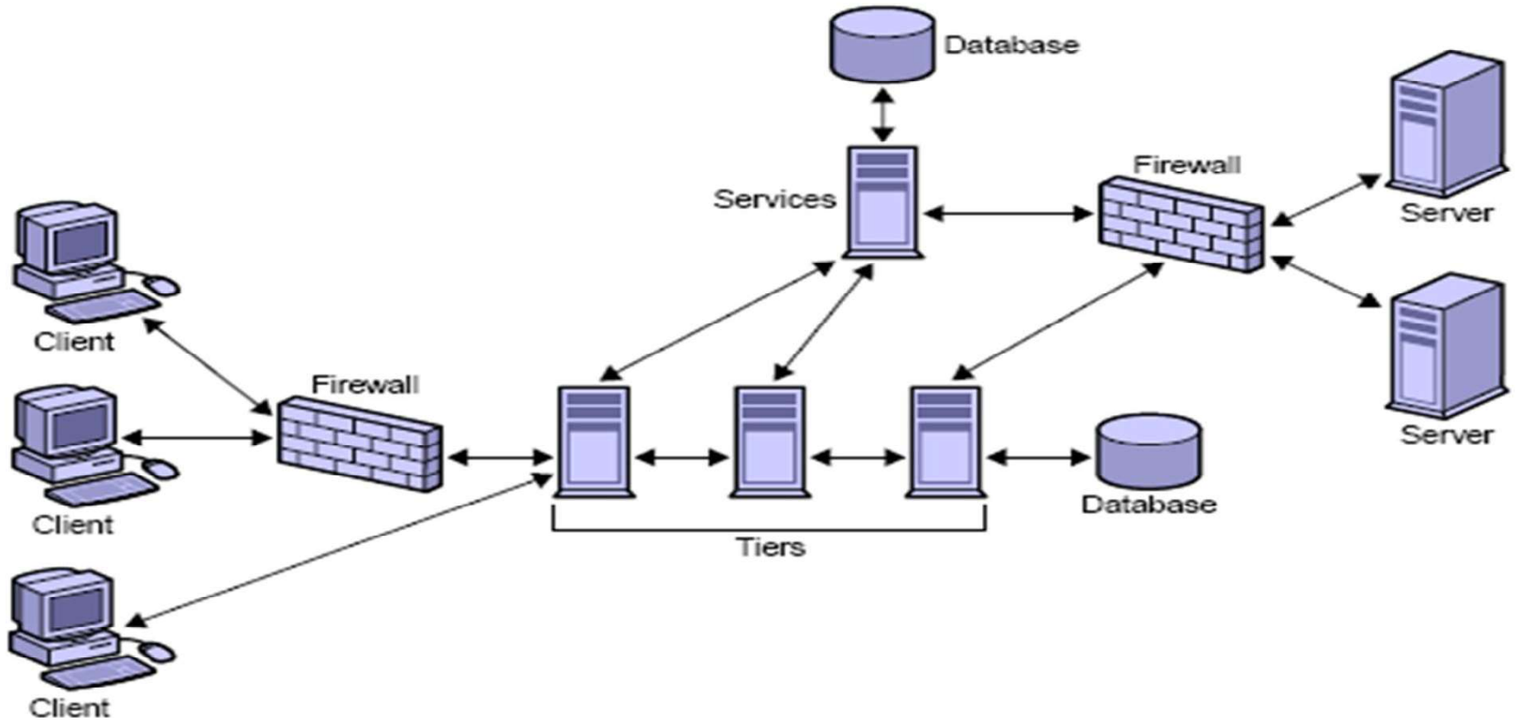
Why is it that software engineering is now employing people in this role? Because of two crucial changes:

- Scale
- Distribution

- Minimally distributed systems (such as client-server)



Highly distributed systems



Distinguishing Between Architecture and

- The table shows how architects differ from designers.

	Architect	Designer
Abstraction level	High/broad Focus on few details	Low/specific Focus on many details
Deliverables	System and subsystem plans, architectural prototype	Component designs, code specifications
Area of focus	Nonfunctional requirements, risk management	Functional requirements

Architectural Principles

- Separation of Concerns
- Dependency Inversion Principle
- Separate volatile from stable components
- Use component and container frameworks
- Keep component interfaces simple and clear
- Keep remote component interfaces coarse-grained

Architectural Patterns and Design Patterns

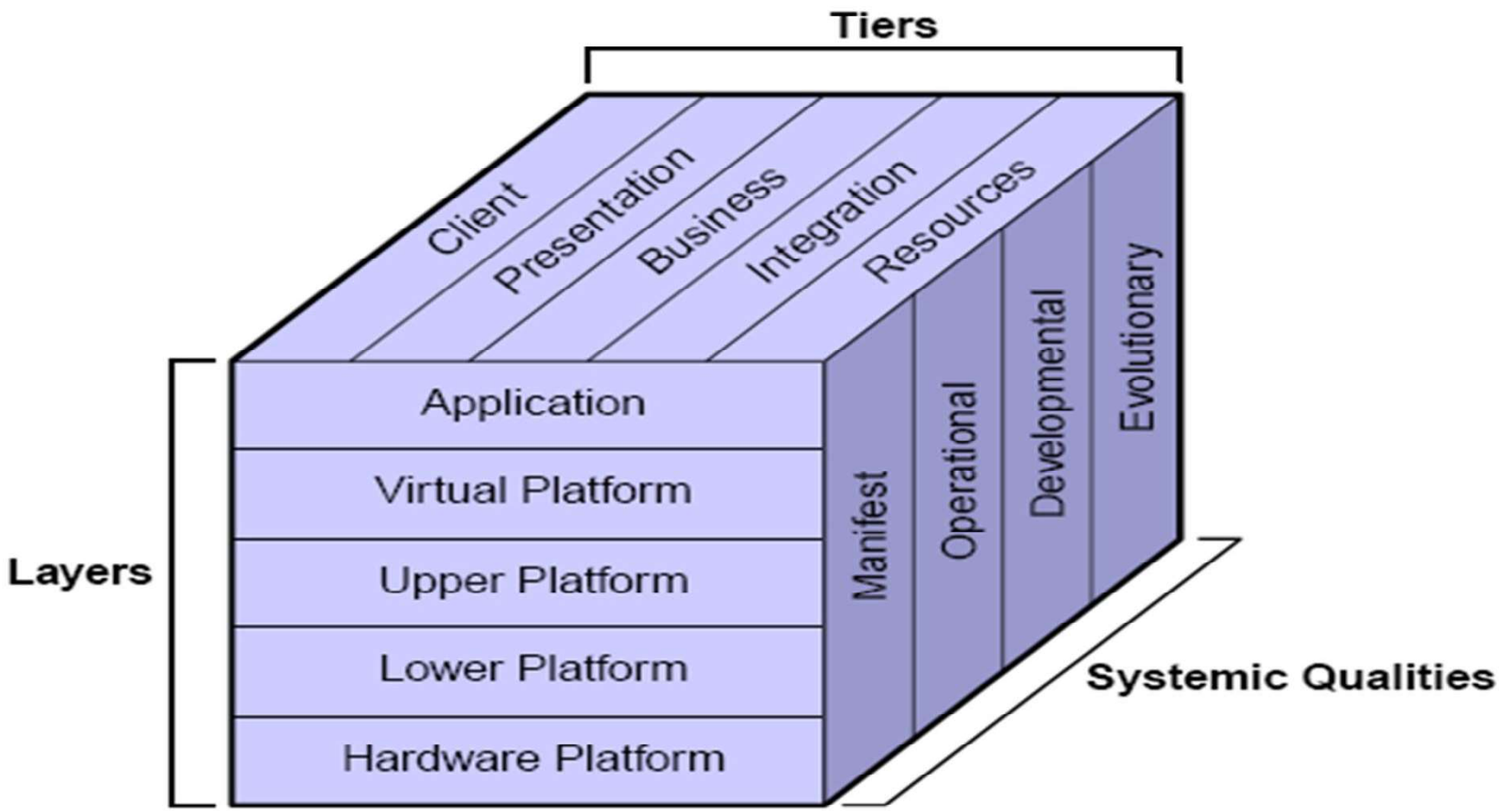
- An architect plans systems using a pattern-based reasoning process.
- An architect must be familiar with a variety of pattern catalogs to be effective.
- Types of patterns:
  - Design patterns define structure and behavior to construct effective and reusable OO software components to support functional requirements.
  - Architectural patterns define structure and behavior for systems and subsystems to support nonfunctional requirements.

Tiers, Layers, and Systemic Qualities

The Sun Tone Architecture Methodology recommends the following architectural dimensions:

- The tiers to separate the logical concerns of the application
- The layers to organize the component and container relationships
- The systemic qualities identify strategies and patterns across the tiers and layers

Applying the SunTone Architecture



Tiers

- **tiers – *"A logical or physical organization of components into an ordered chain of service providers and consumers."***

  - Client – Consists of a thin client, such as a web browser.
  - Presentation – Provides the HTML pages and forms that are sent to the Web browser and processes the user's requests.
  - Business – Provides the business services and entities.
  - Integration – Provides components to integrate the Business tier with the Resource tier.
  - Resource – Contains all backend resources, such as a DataBase Management System (DBMS) or Enterprise Information System (EIS).

Layers

layers – ***"The hardware and software stack that hosts services within a given tier. (layers represent component/container relationships)"***

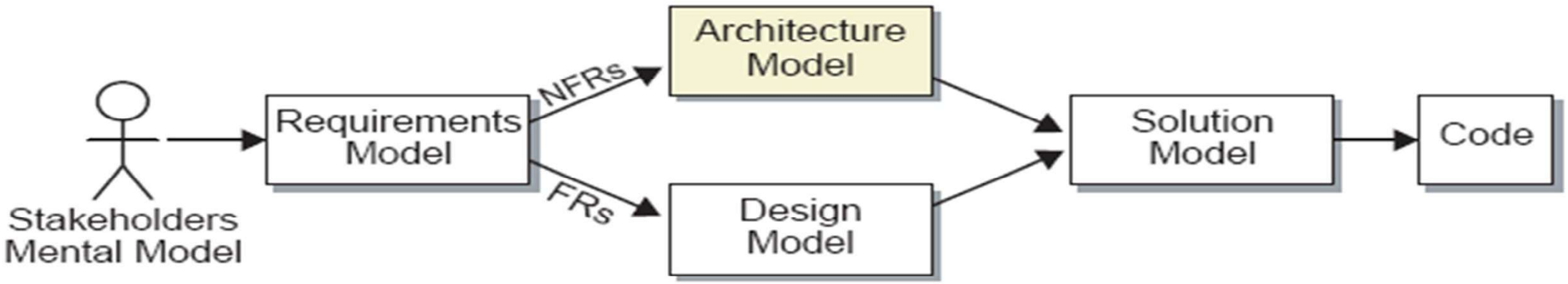
- Application – Provides a concrete implementation of components to satisfy the functional requirements.
- Virtual Platform – Provides the APIs that application components implement.
- Upper Platform – Consists of products such as web and EJB technology containers and middleware.
- Lower Platform – Consists of the operating system.
- Hardware Platform – Includes computing hardware such as servers, storage, and networking devices.

Systemic Qualities

“The strategies, tools, and practices that will deliver the requisite quality of service across the tiers and layers.”

- Manifest – Addresses the qualities reflected in the enduser experience.
- Operational – Addresses the qualities reflected in the execution of the system.
- Developmental – Addresses the qualities reflected in the planning, cost, and physical implementation of the system.
- Evolutionary – Addresses the qualities reflected in the long-term ownership of the system.

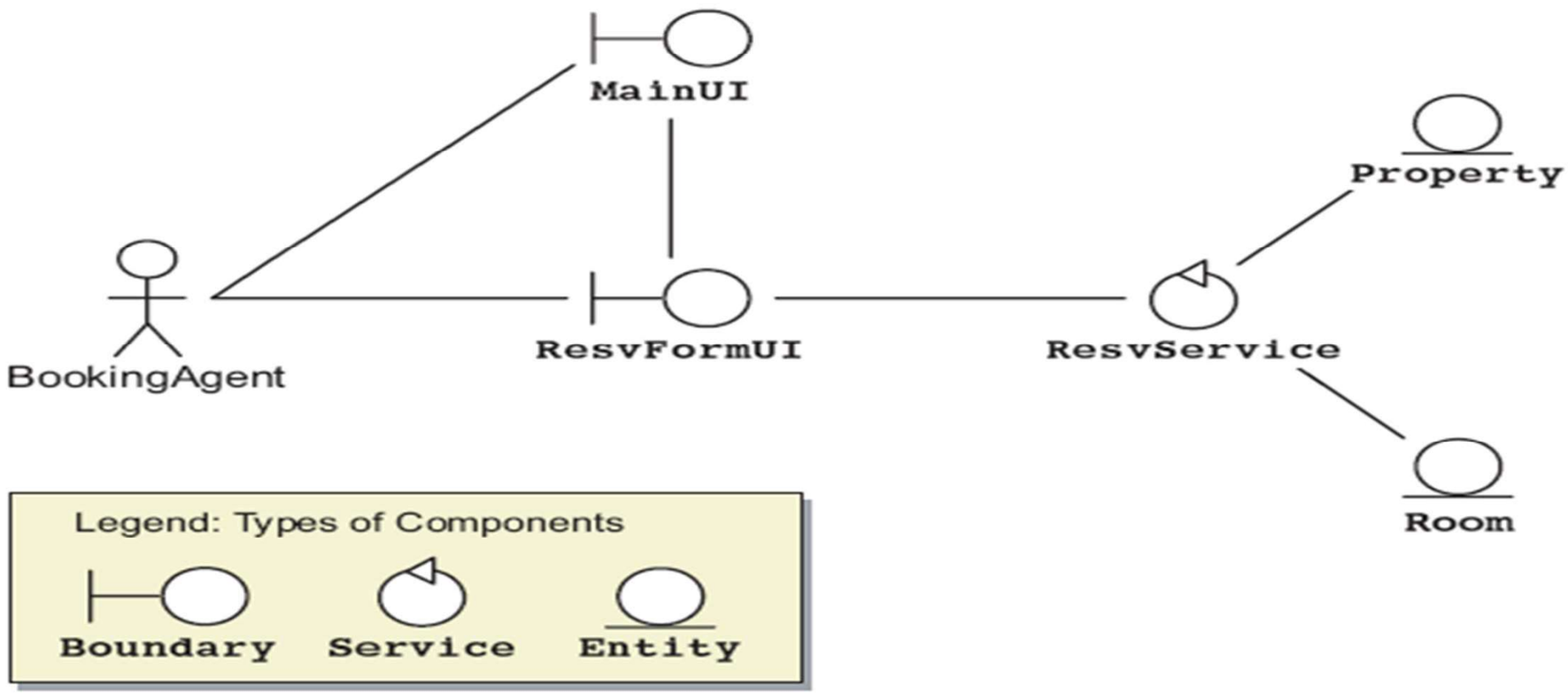
- The Architecture model is essential to the creation of the Solution model:



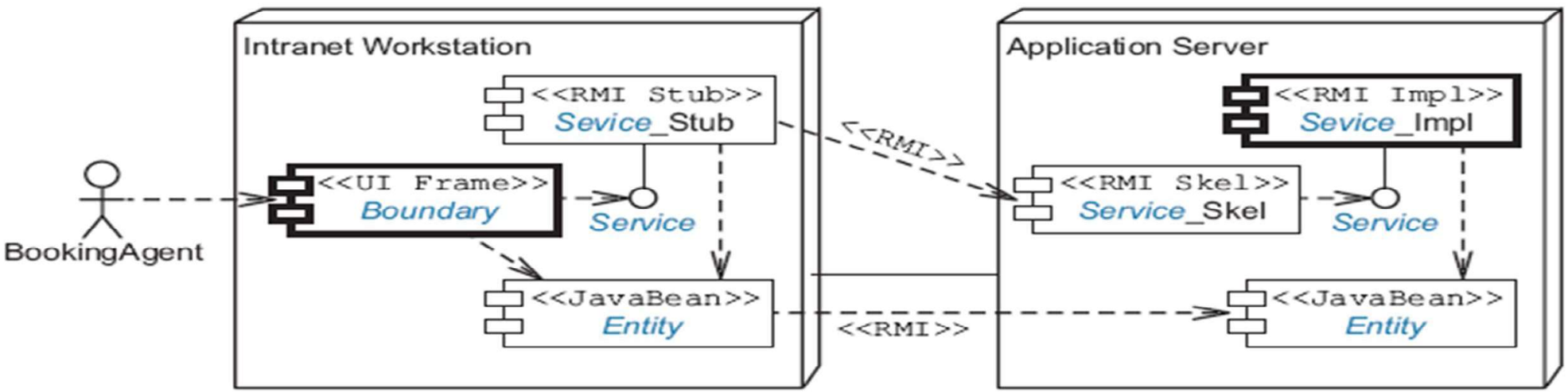
Introducing the Architecture Workflow

1. Select an architecture type for the system.
2. Create a detailed Deployment diagram for the architecturally significant use cases.
3. Refine the Architecture model to satisfy the NFRs.
4. Create and test the Architecture baseline.
5. Document the technology choices in a tiers and layers Package diagram.
6. Create an Architecture template from the final, detailed Deployment diagram.

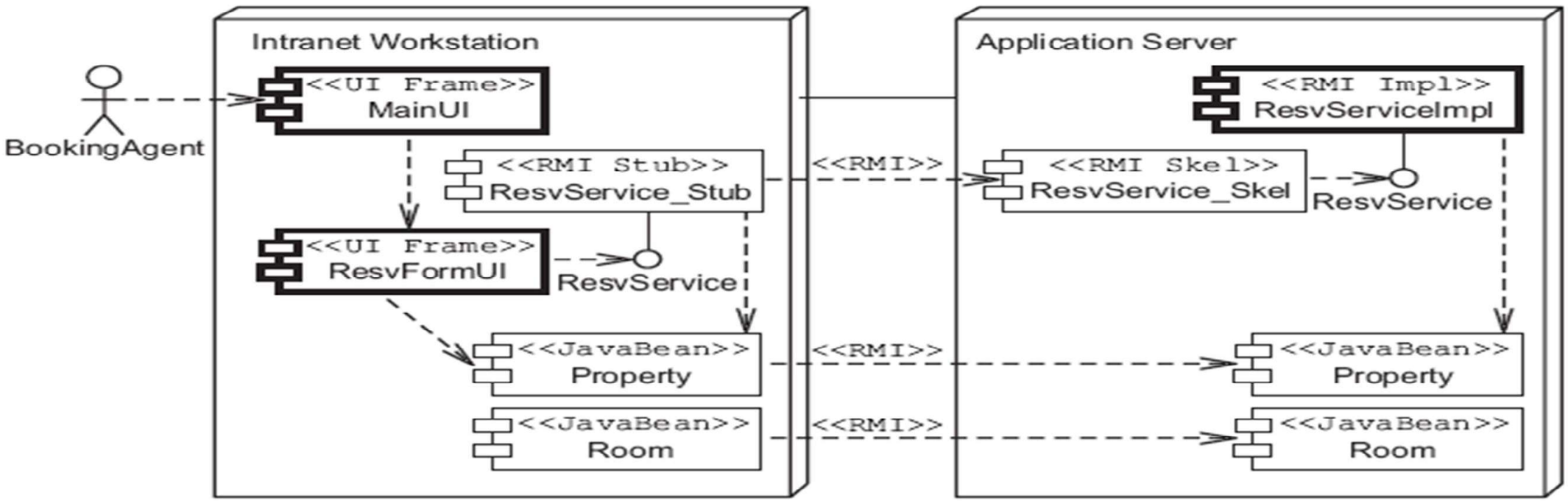
Example Design Model



Example Architecture Template



Example Solution Model



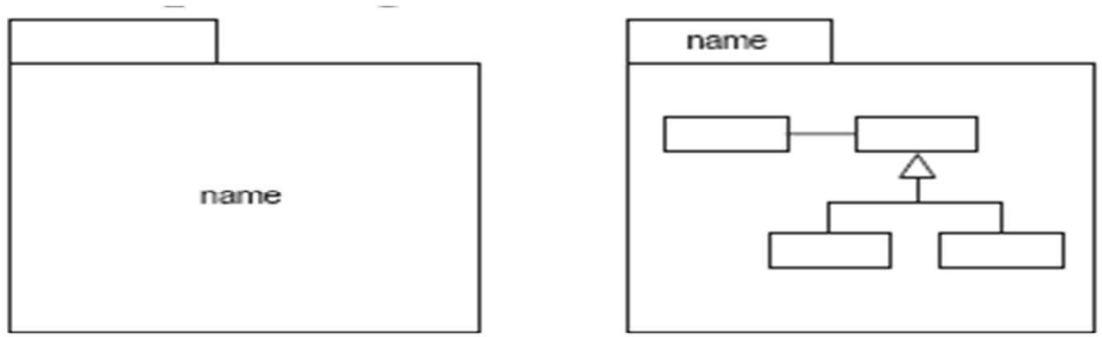
Architectural Views

The views of the Architecture model take many forms. Some elements (such as risk mitigation plans) are documented with text. Others can be recorded using UML diagrams:

- Package diagrams
- Component diagrams
- Deployment diagrams

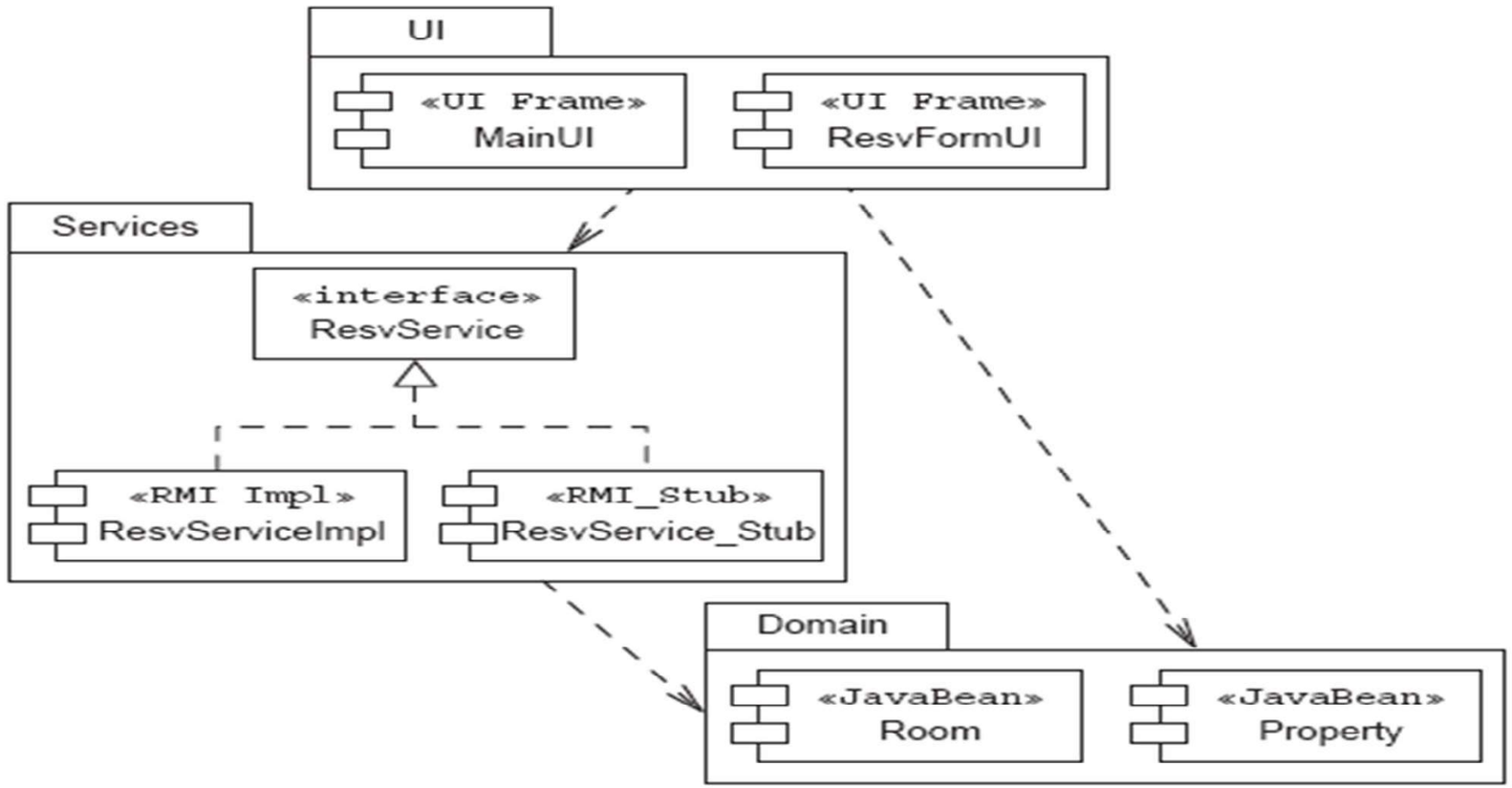
Identifying the Elements of a Package

- A UML package diagram shows dependencies between packages, which can hold any UML element.
- The UML package notation:

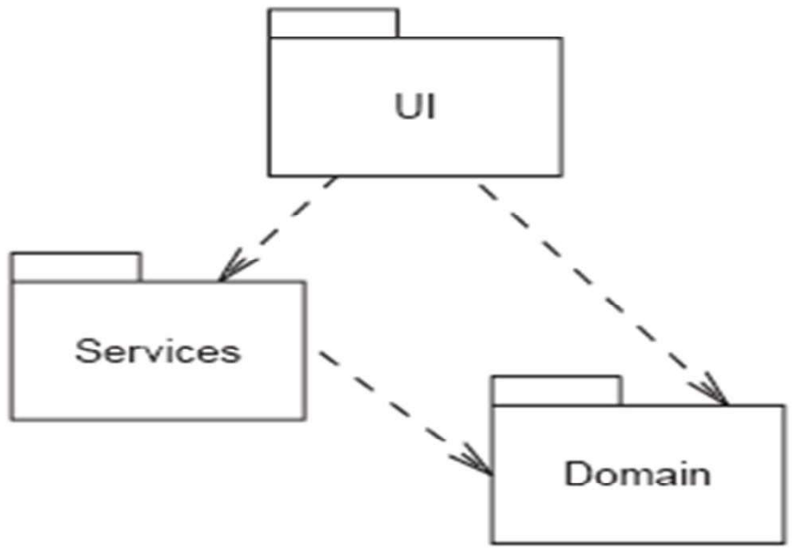


- The package name can be placed in the body box or in the name box.
- You can place any UML entity in a package, including other packages.

Example Package Diagram

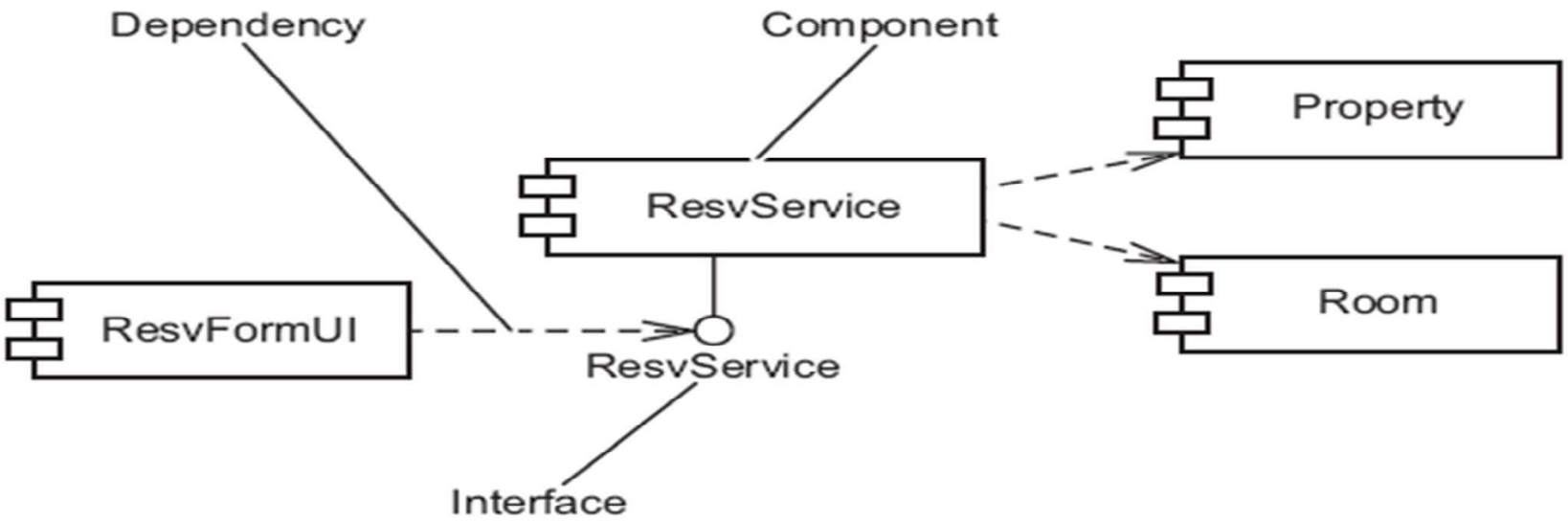


An Abstract Package Diagram



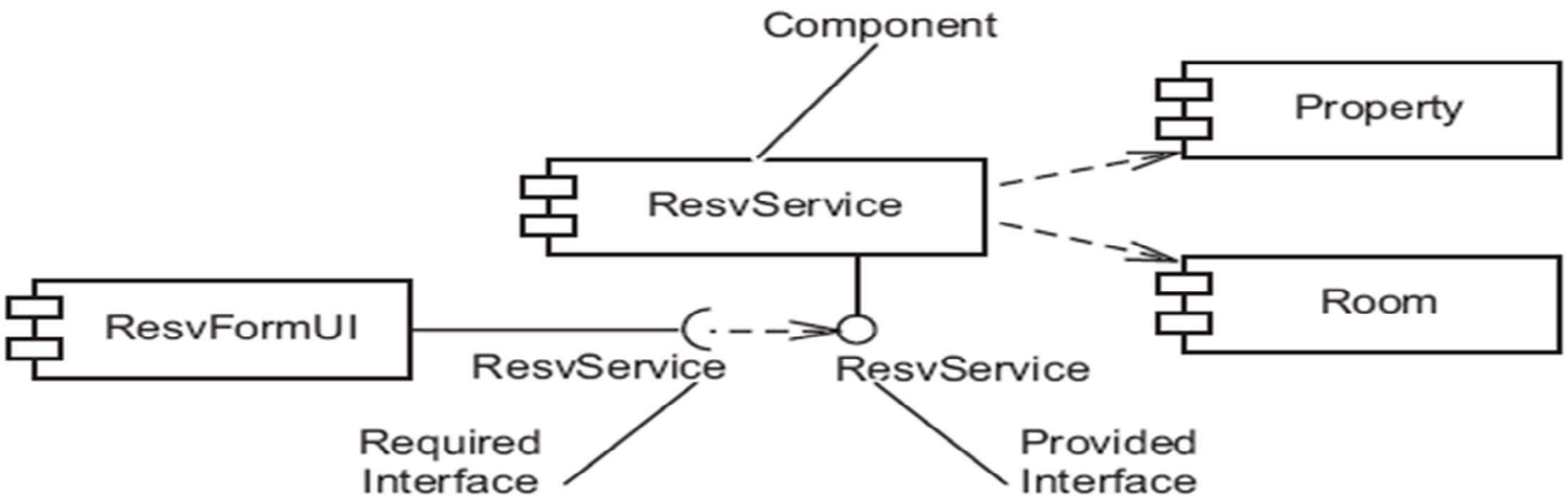
Identifying the Elements of a Component

A UML Component diagram is composed of the following elements:



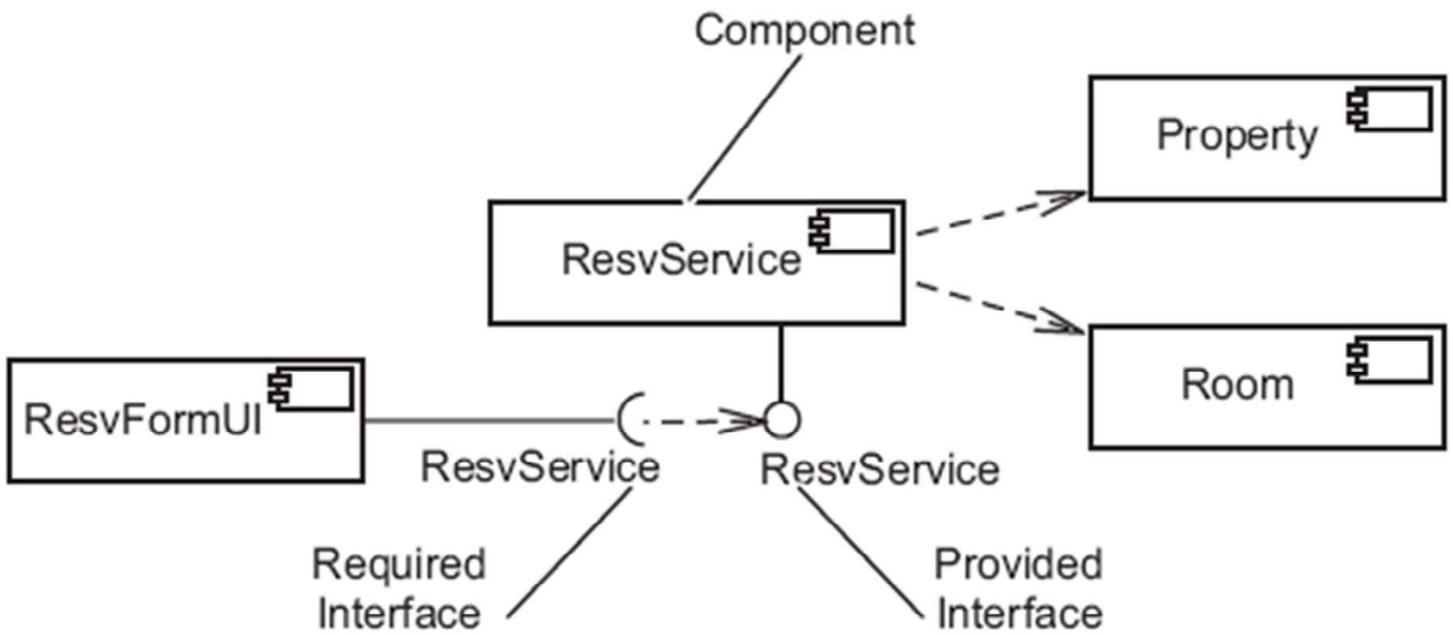
Identifying the Elements of a Component

- Notation for required interfaces in UML 2:



Identifying the Elements of a Component

- Component notation in UML 2:

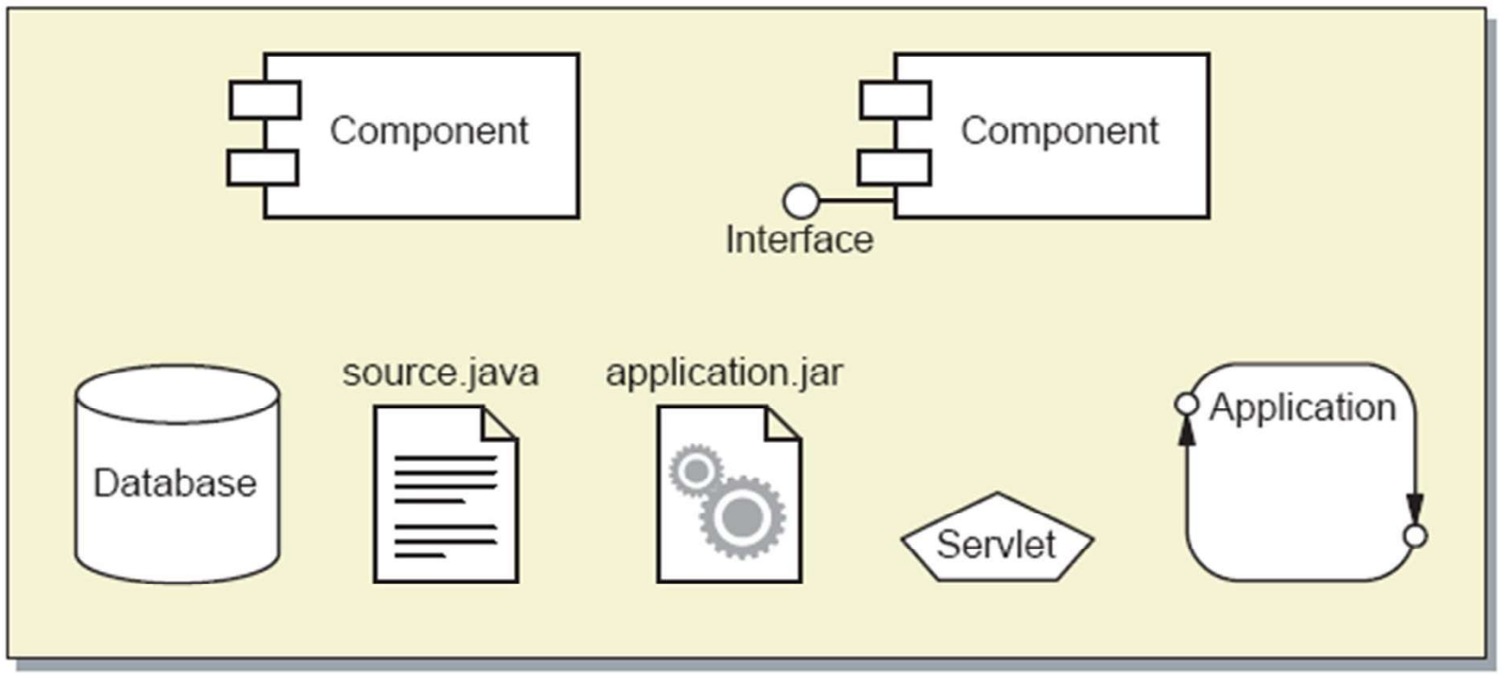


Characteristics of a Component

- A component represents any *software unit.*
- A component can be large and abstract.
- A component can be small.
- A component might have an interface that it exports as a service to other components.
- A component can be a file, such as a source code file, an *object file, a complete executable, a data file, HTML or* media files, and so on.

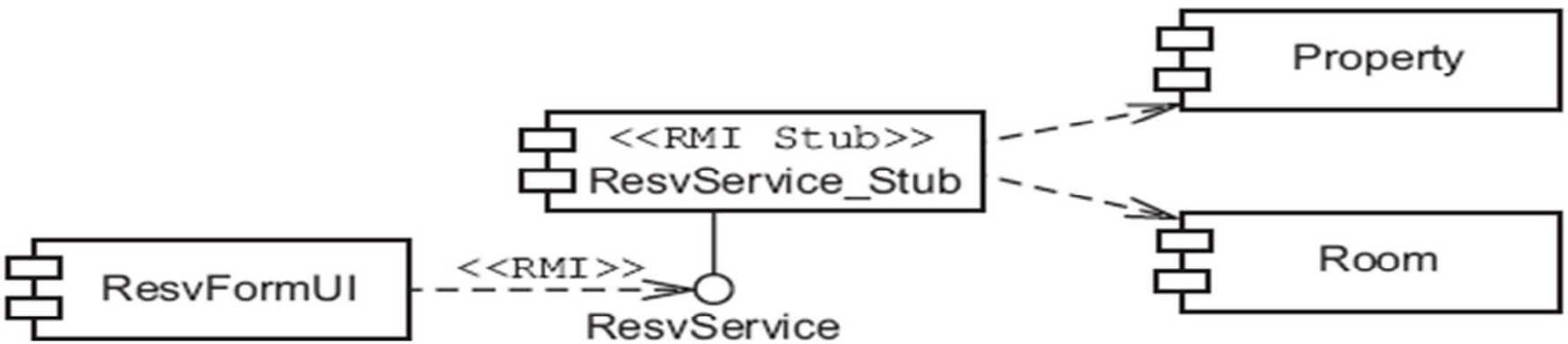
Types of Components

- A component is any physical software unit:



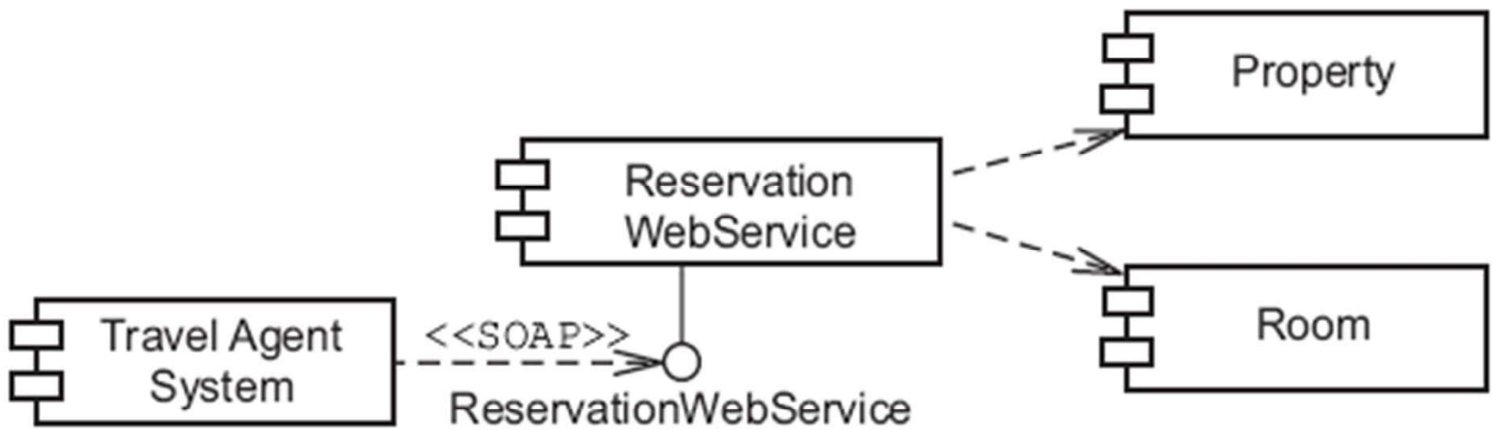
Example Component Diagrams

- Component diagrams can show software dependencies. This example shows an RMI dependency:



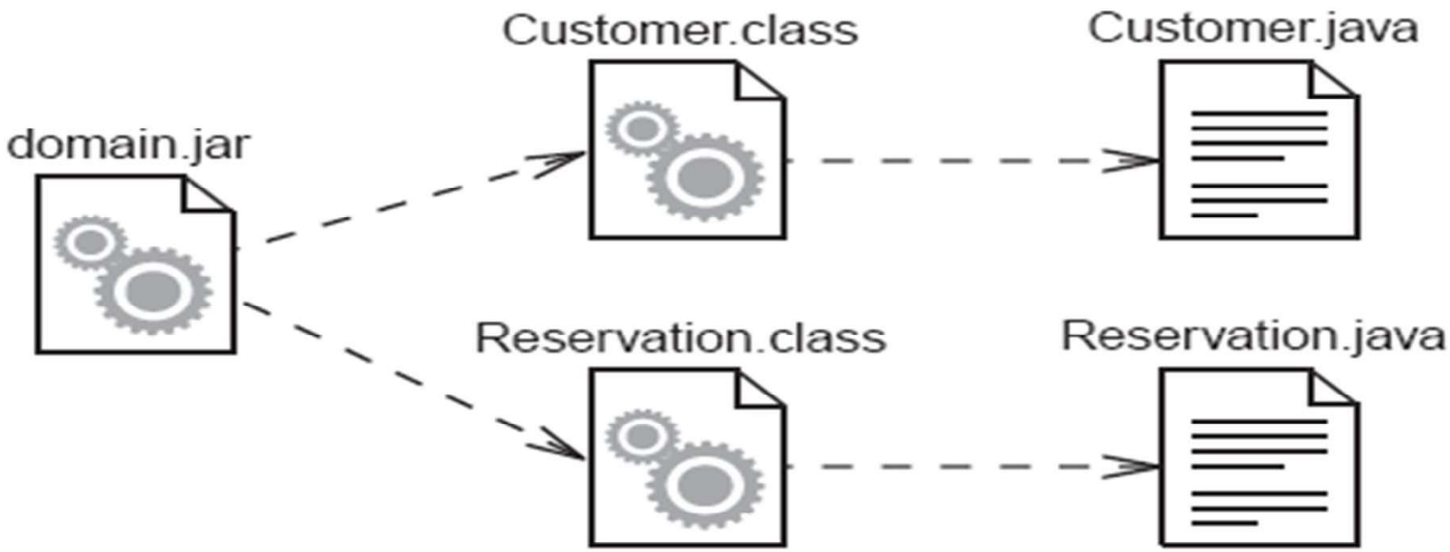
Example Component Diagrams

- Component diagrams can show software dependencies. This example shows a Web Service dependency:

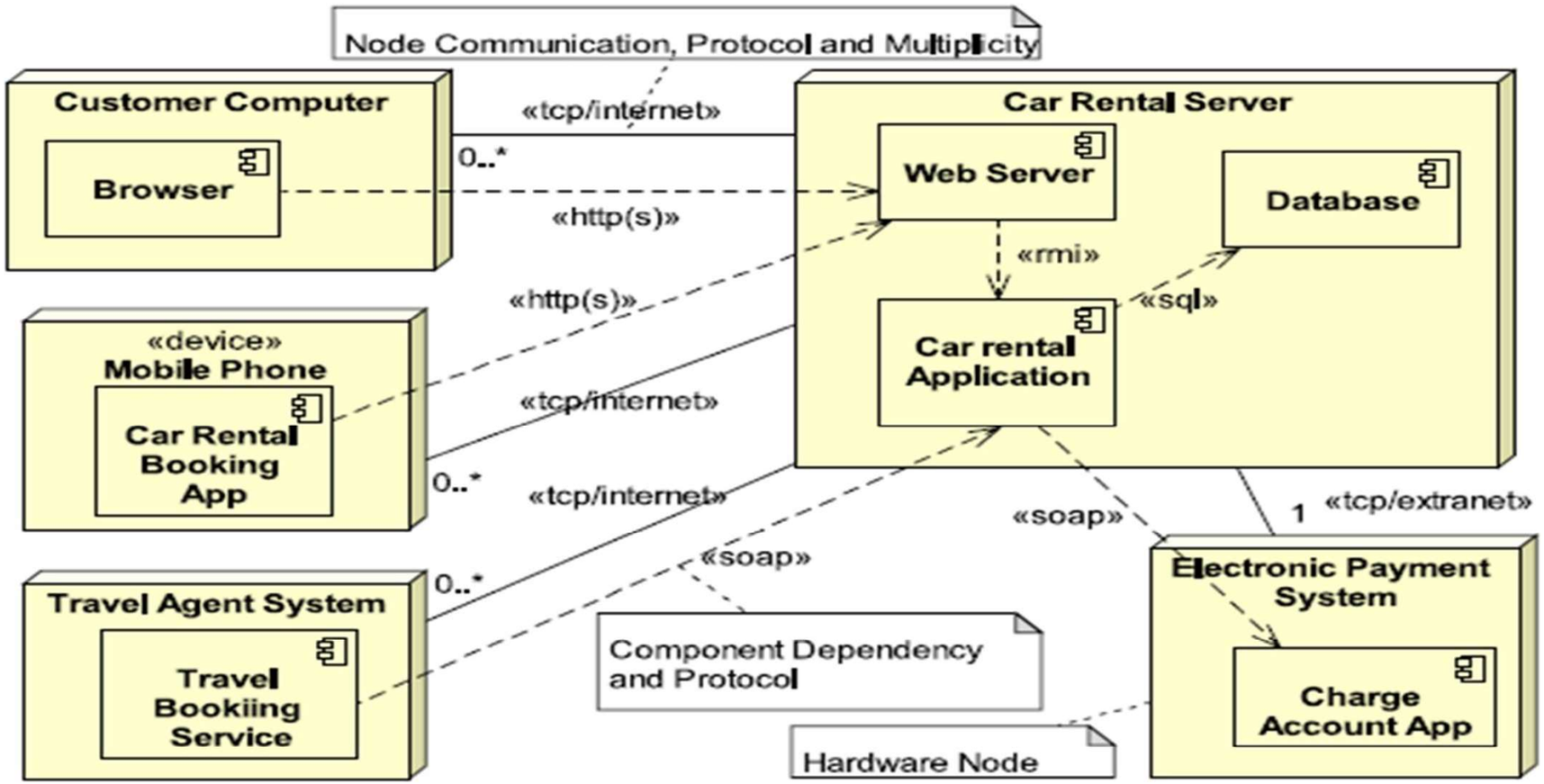


Example Component Diagrams

- Component diagrams can represent build structures:



Identifying the Elements of a Deployment



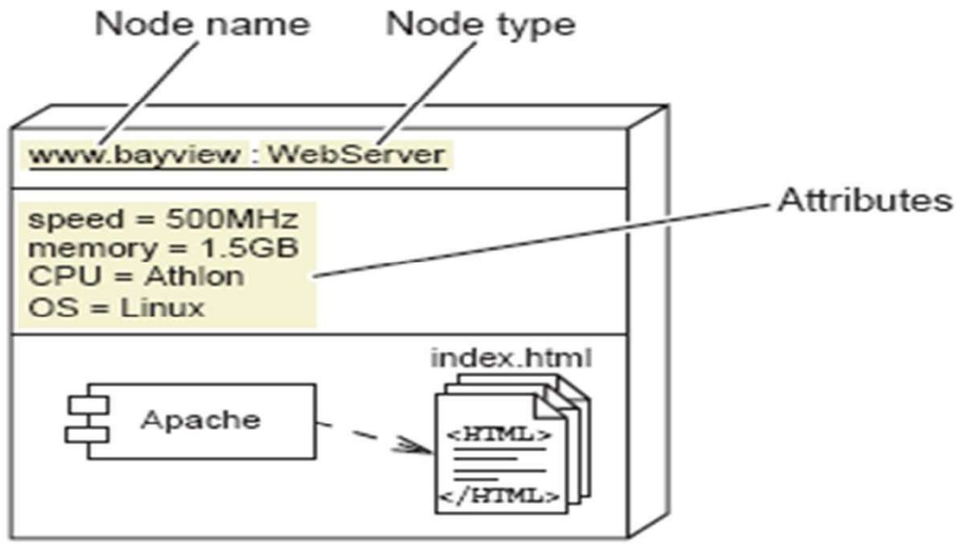
The Purpose of a Deployment Diagram

- Hardware nodes can represent any type of physical hardware.
- Links between hardware nodes indicate connectivity and can include the communication protocol used between nodes.
- Software components are placed within hardware nodes to show the distribution of the software across the network.

Types of Deployment Diagrams

There are two forms:

- A *descriptor Deployment diagram shows the* fundamental hardware configuration.
- An *instance Deployment diagram shows a specific* hardware configuration. For example:



Selecting the Architecture Type

The architecture you use depends on many factors, including:

- The platform constraints in the system requirements
- The modes of user interaction
- The persistence mechanism
- Data and transactional integrity

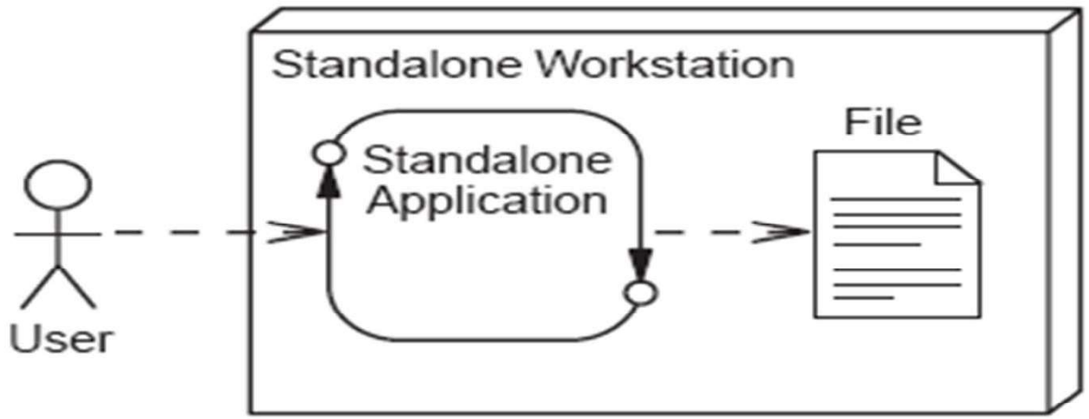
Selecting the Architecture Type

There are hundreds of successful software architectures. Here are a few common types:

- Standalone applications
- Client/Server (2-tier) applications
- N-tier applications
- Web-centric n-tier applications
- Enterprise n-tier applications

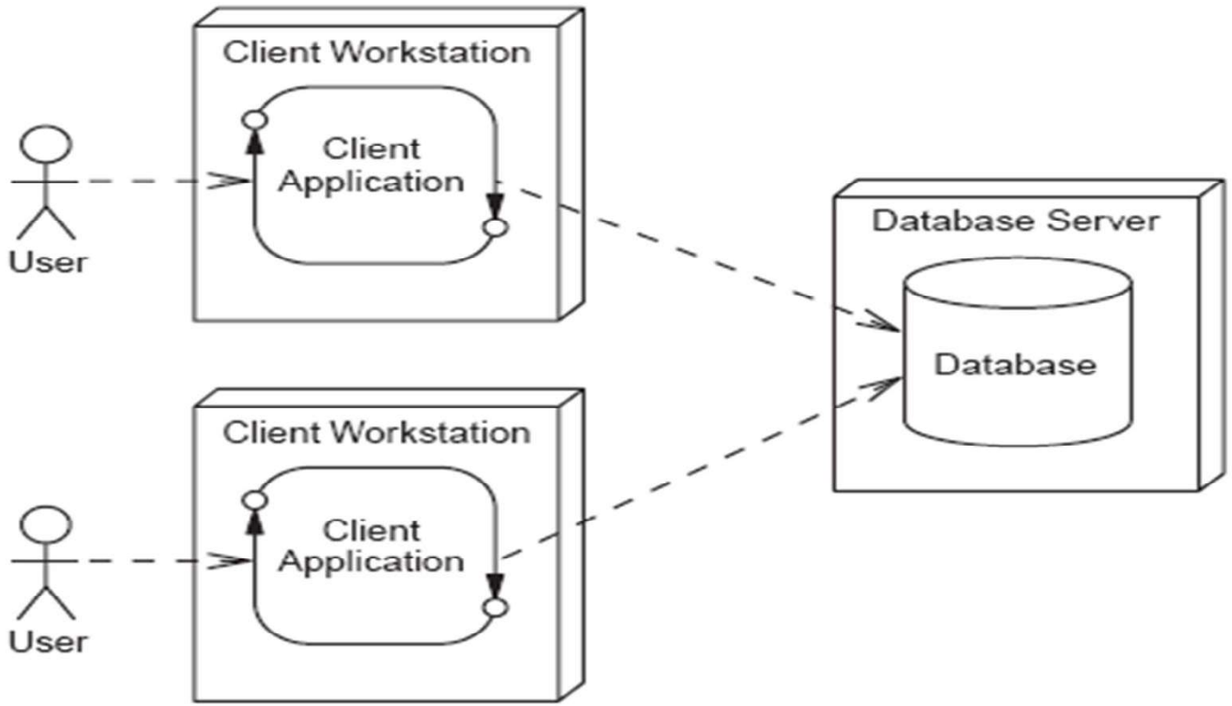
Standalone Applications

- No external data sources (all application data exists on a file server)
- No network communication (all application components exist on one machine)



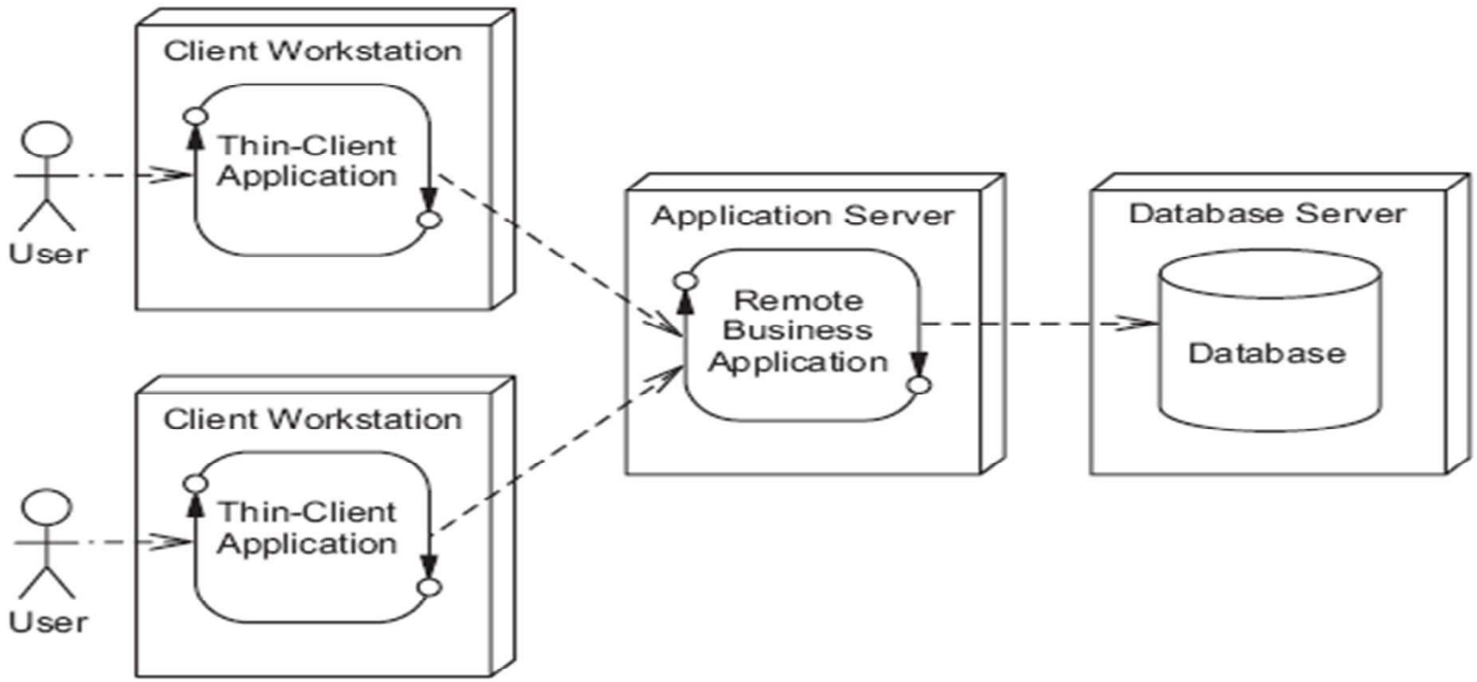
Client/Server (2-Tier) Applications

- Thick client (with business logic in the client tier)
- Data store manages data integrity



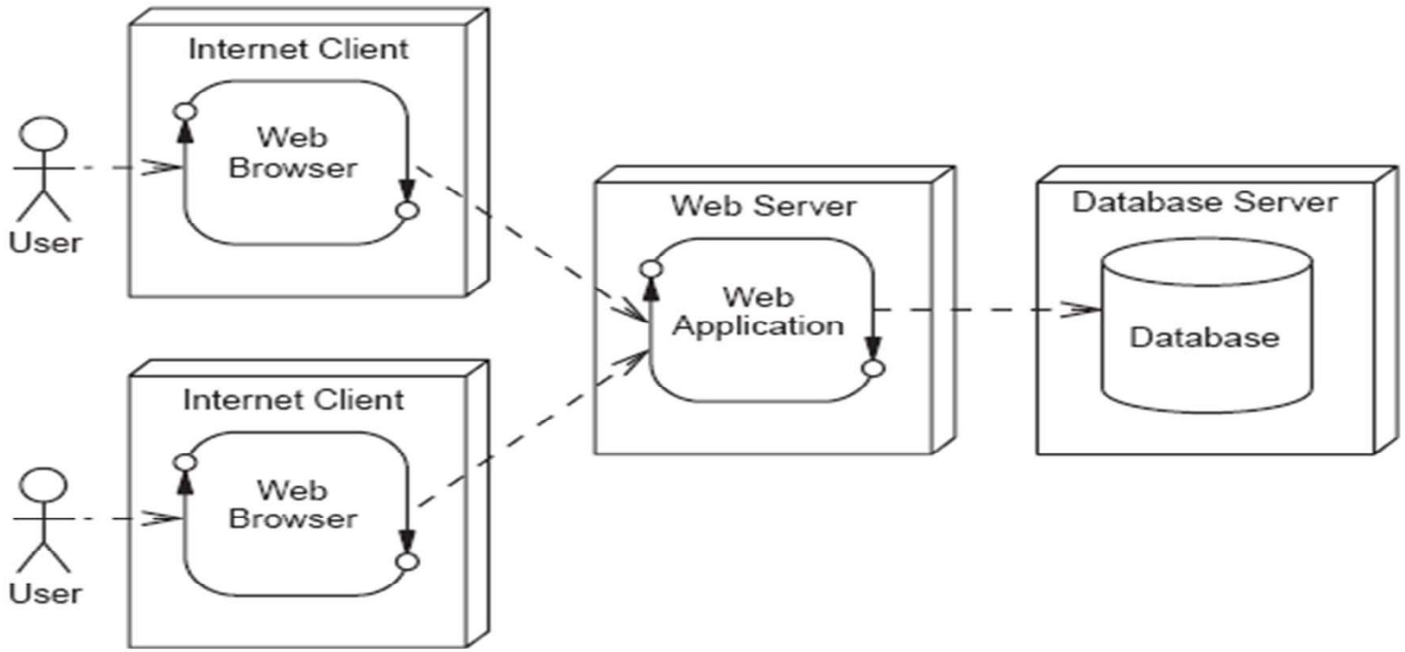
N-Tier Applications

- Thin client (business logic is in the application server)
- Application server manages data integrity

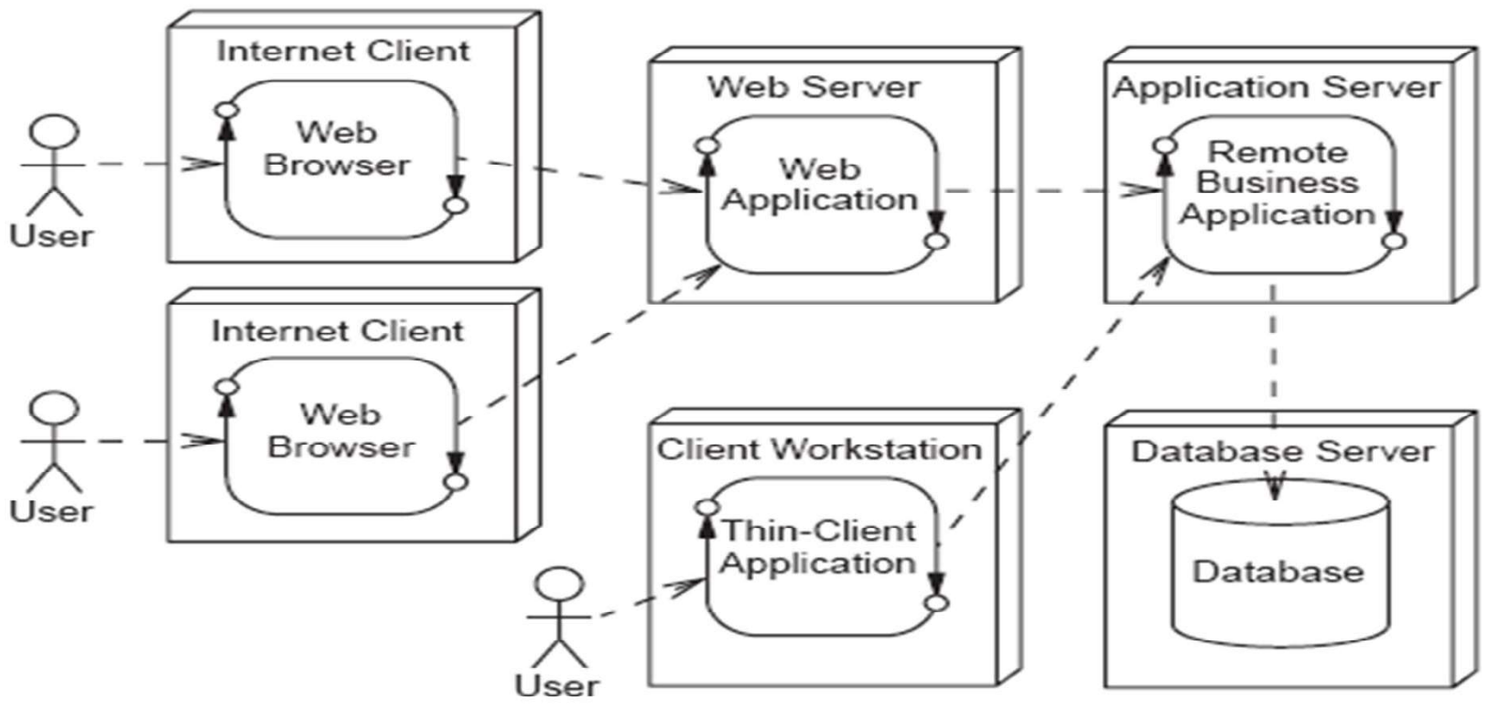


Web-Centric (N-Tier) Applications

- Web browser becomes the thin client
- Web server provides presentation and business logic



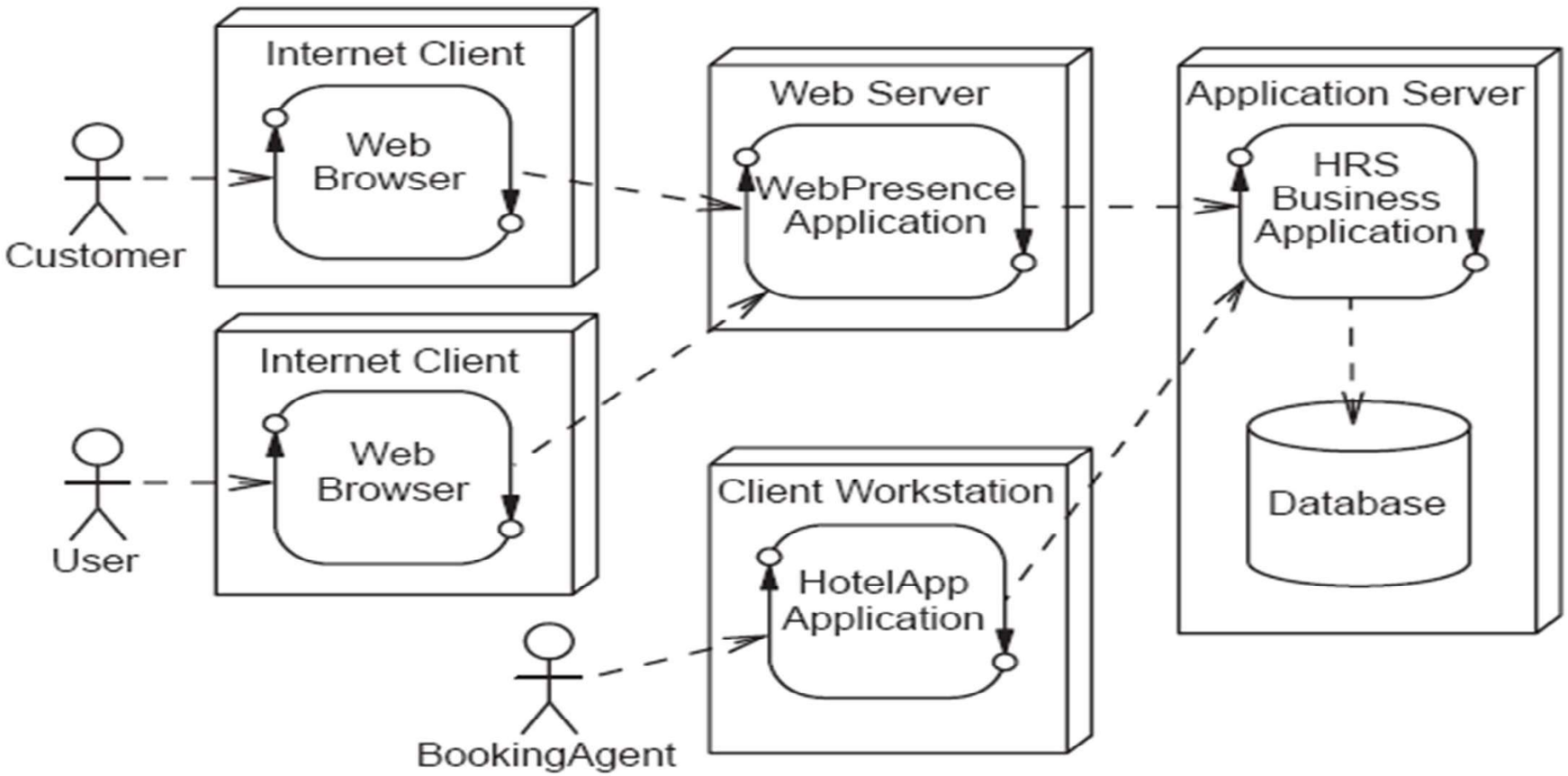
Enterprise (N-Tier) Architecture Type



Enterprise (N-Tier) Architecture Type

- Two thin clients:
  - Web browser for Internet users
  - GUI thin client for intranet users
- Web application server provides presentation logic
- Application server provides business logic

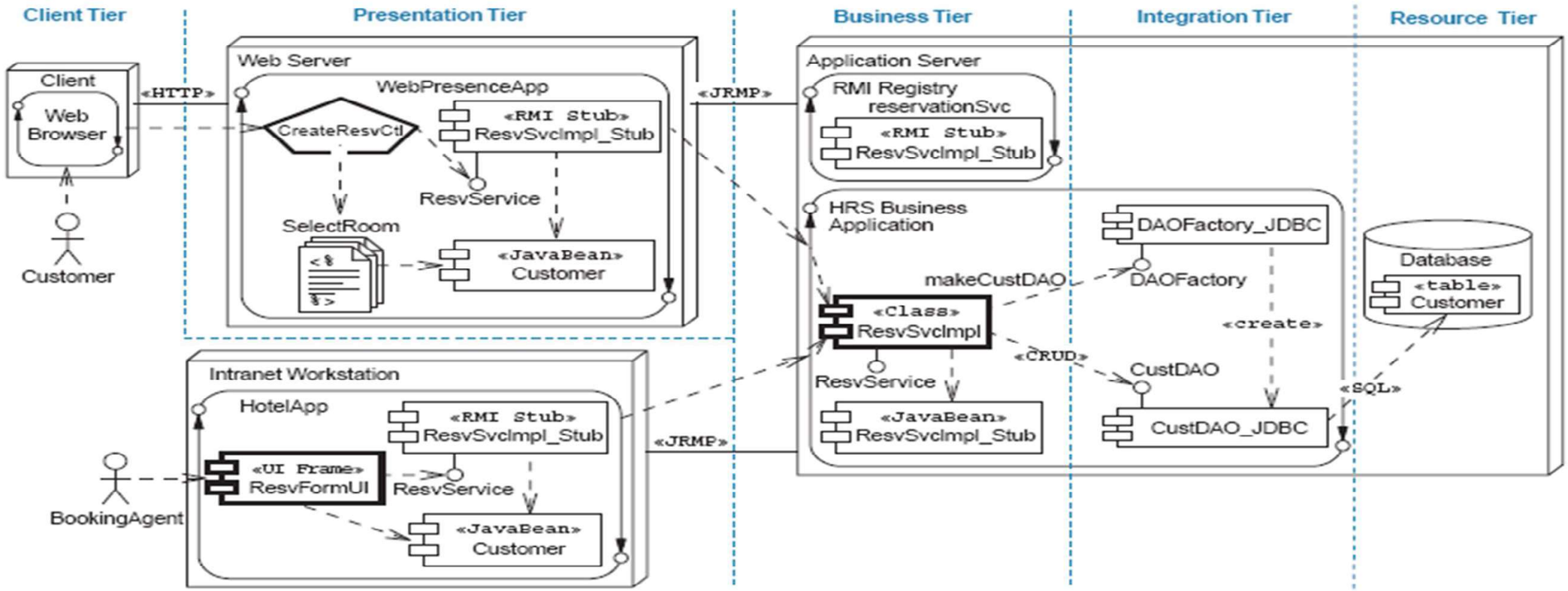
Hotel Reservation System Architecture



Creating The Detailed Deployment Diagram

1. Design the components for the architecturally significant use cases.
2. Place design components into the Architecture model.
3. Draw the detailed Deployment diagram from the merger of the design and infrastructure components.

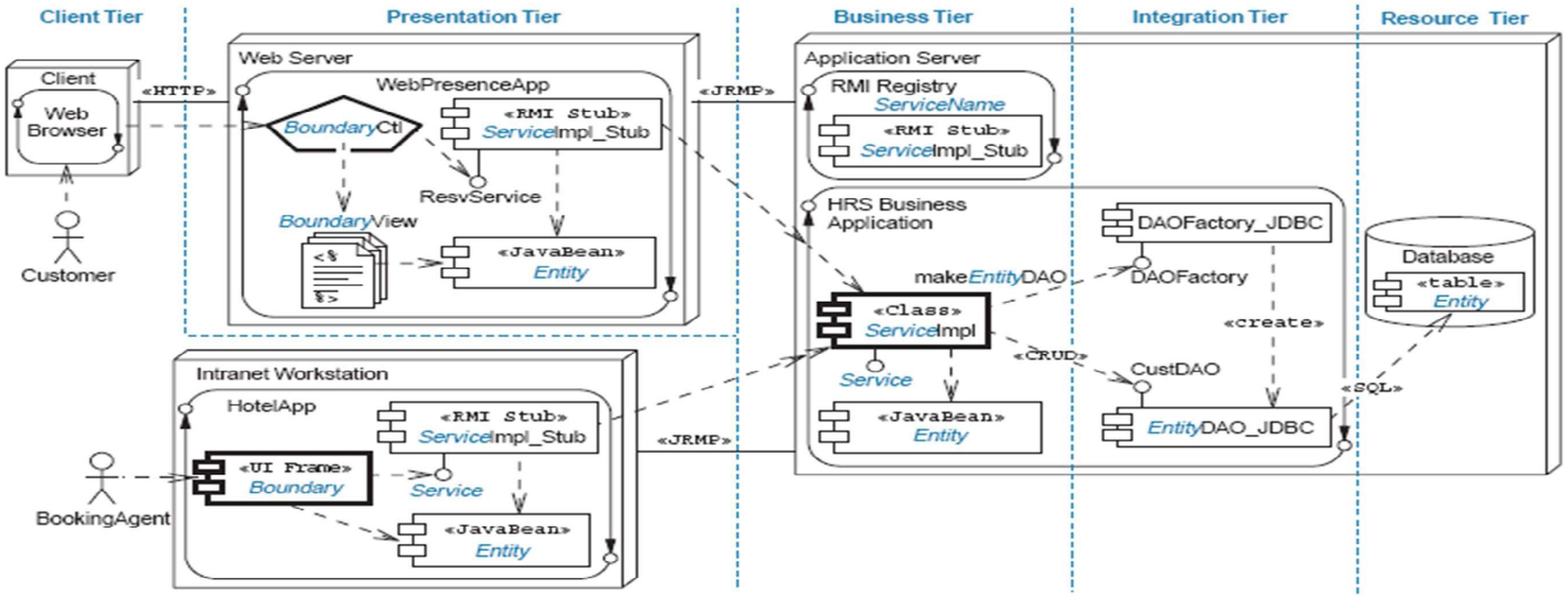
Example Detailed Deployment Diagram



Creating the Architecture Template

1. Strip the detailed Deployment diagram to just one set of Design components: boundary, service, and entity.
2. Replace the name of the Design component with the type (for example, ResvSvcImpl_Stub becomes *ServiceImpl_Stub).*

Example Architecture Template

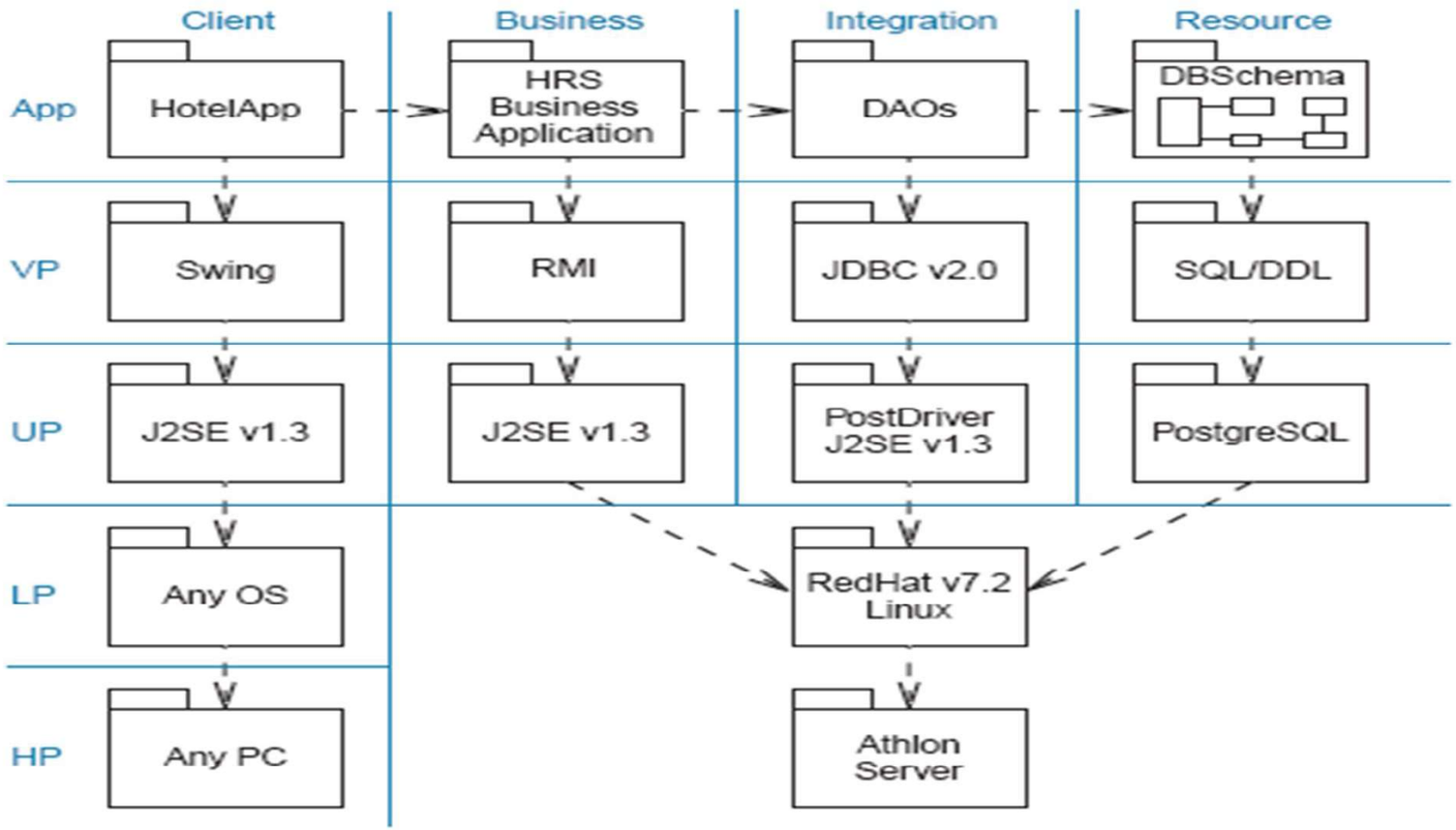


Creating the Tiers and Layers Package

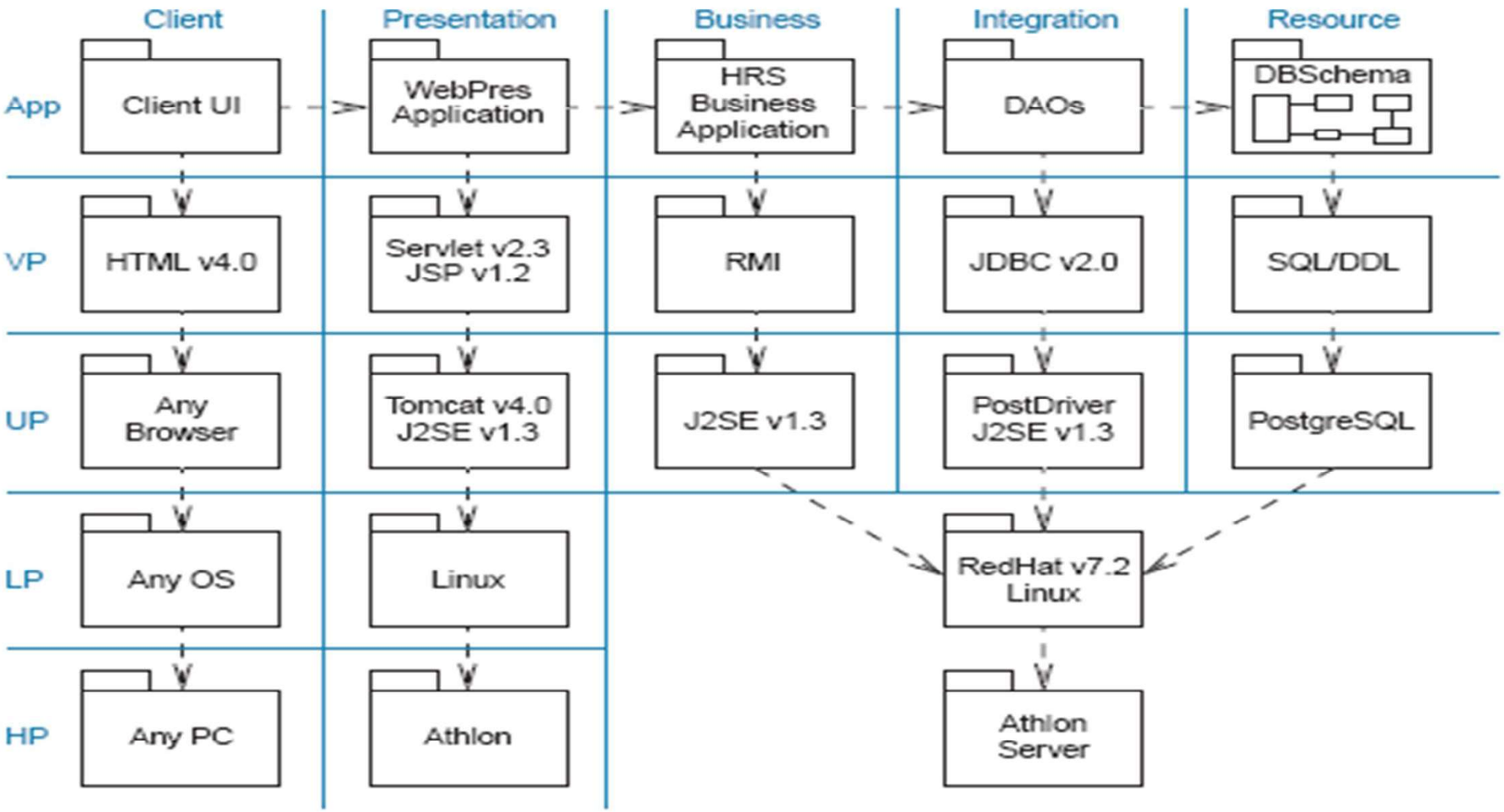
For each tier:

1. Determine what application components exist.
2. Determine what technology APIs, communication protocols, or specifications are required that the components require.
3. Determine which container products to use.
4. Determine which operating system to use.
5. Determine what hardware to use.

Tiers and Layers Diagram for the HotelApp



Tiers and Layers Diagram for the Hotel



Summary

In this lesson, you should have learned the following:

- Difference between architecture and design:
  - Design produces components to implement a use case.
  - Architecture provides a template into which the designed components are realized.
- You can model the architecture using:
  - Deployment diagrams
  - Component diagrams
  - Packages
  - Tiers and layers

Practice : Overview

This practice covers the following topics:

- Working with Deployment diagrams
- Component diagrams
- Packages