



Introduction to Software Architecture

**Software Architecture
3rd Year – Semester 1
Lecture 1
By Udara Samaratunge**

What is Software Architecture?

Let's Explore with an Analogy...

Software Architecture:

What, Where, How?

What is Software Architecture?

- *Software architecture* encompasses the structures of large software systems:
 - abstract view
 - eliminates details of implementation, algorithm, & data representation
 - concentrates on the behavior & interaction of “black box” elements

Definition

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Food for Thought...

- Quick Exercise:
 - What is the relationship of a system's software architecture to the environment in which the system will be constructed and exist?

- **Answer:**
 - Software architecture is a result of *technical*, *business*, and *social* influences.
 - In turn, it affects each of these environments.

Architecture Business Cycle (ABC)



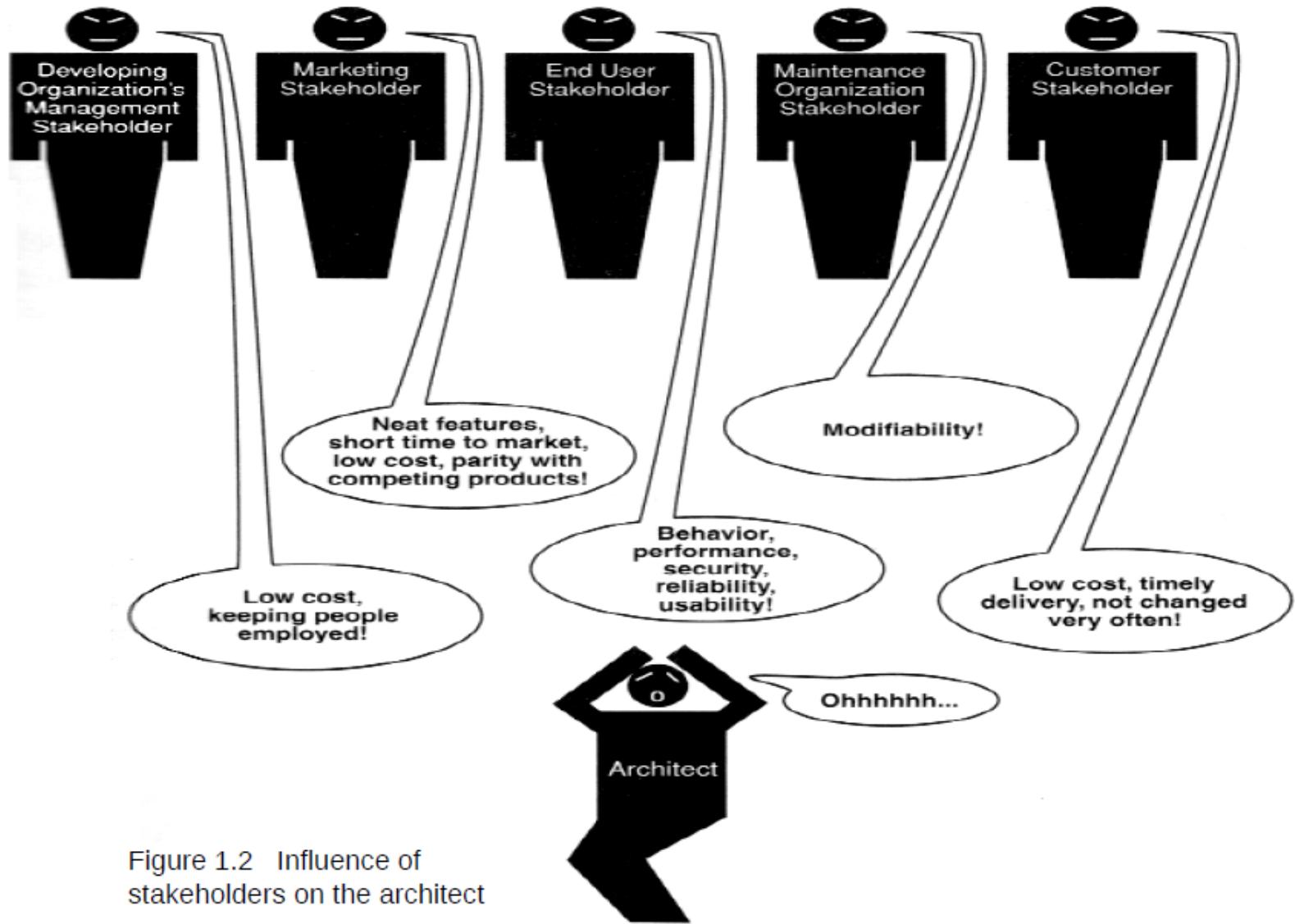


Figure 1.2 Influence of stakeholders on the architect

Architectural Influences

- Stakeholders
 - each stakeholder has different concerns & goals, some contradictory
- Development Organization
 - immediate business, long-term business, and organizational (staff skills, schedule, & budget)
- Background & Experience of the Architects
 - repeat good results, avoid duplicating disasters
- The Technical Environment
 - standard industry practices or common SE techniques

Software Architecture Patterns

Software Architecture Patterns

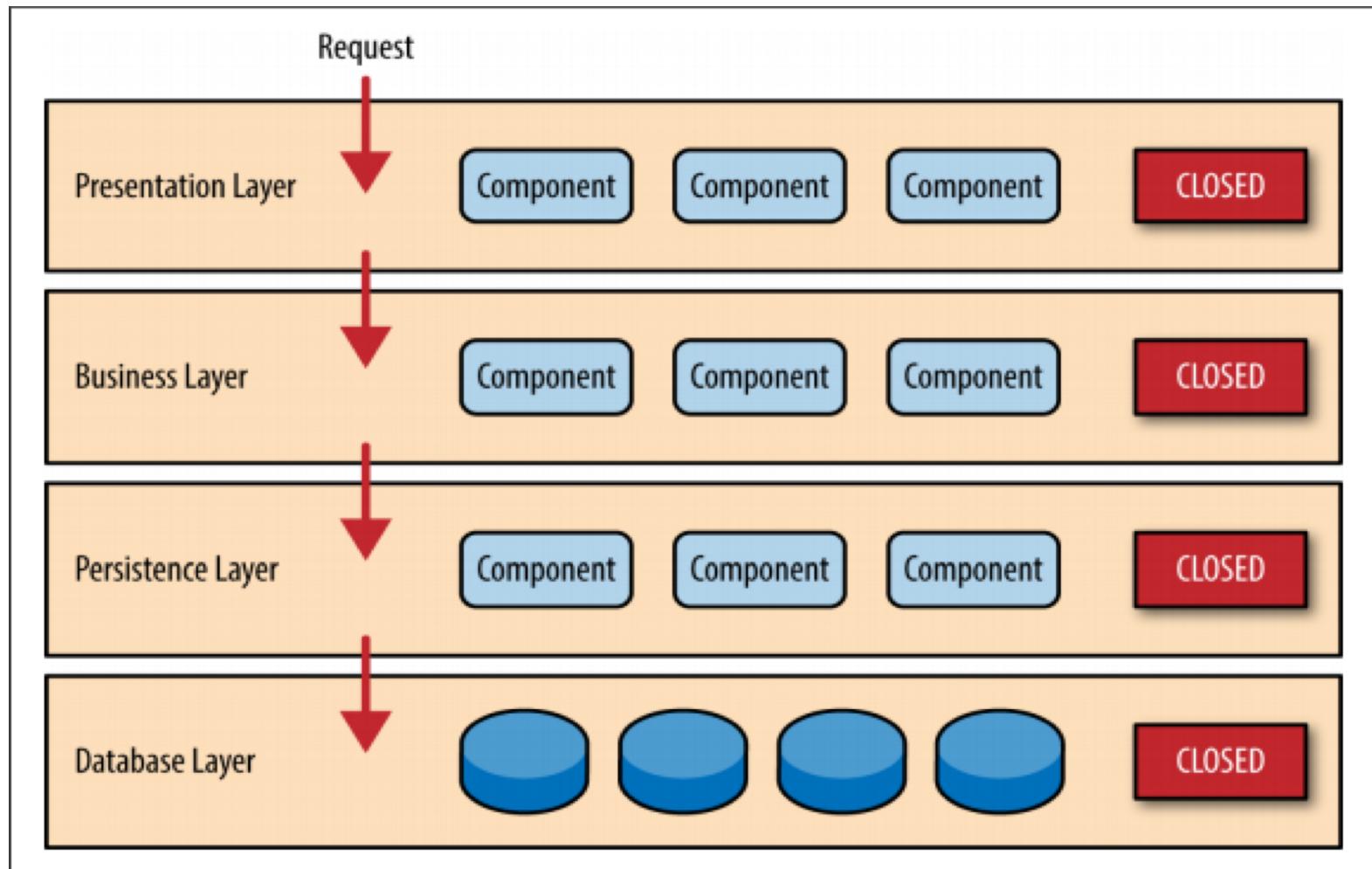
Why Use a Pattern?

- Proven construct
- Easy to communicate
- Keep things in order

Software Architecture Patterns

- Layered Architecture
- Event-Driven Architecture
- Microkernel Architecture
- Micro services Architecture

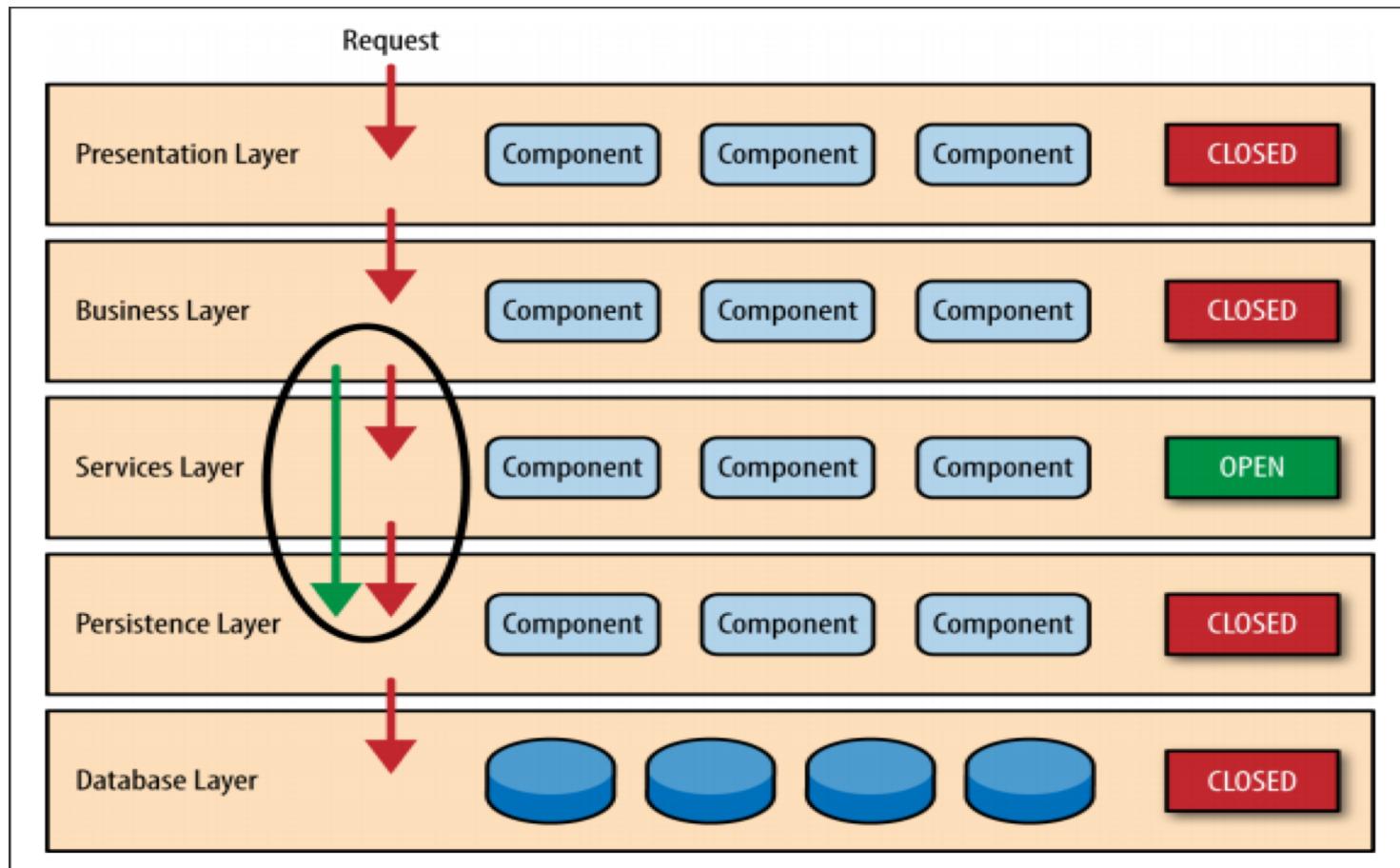
Layered Architecture



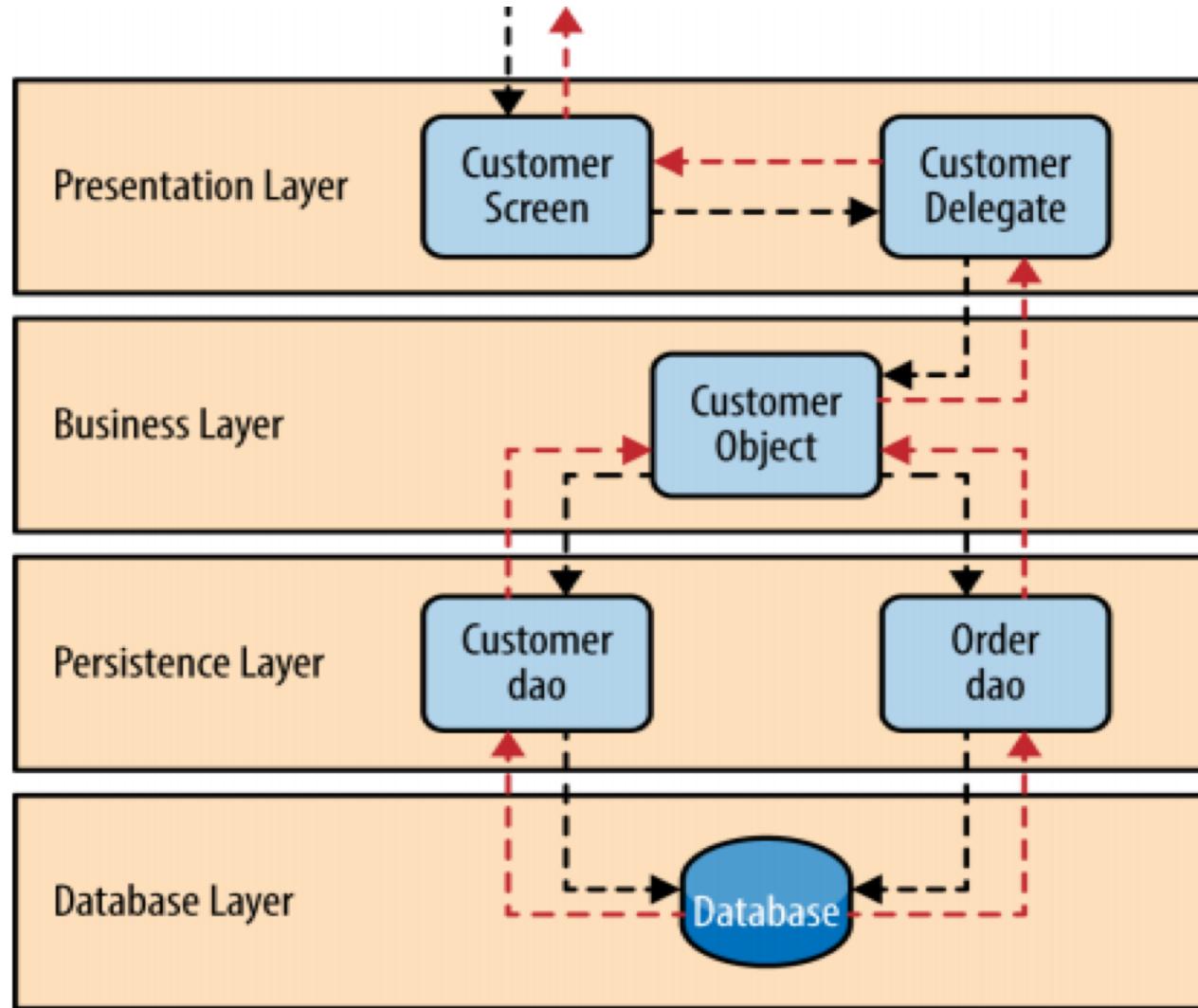
Key Concepts

- Separation of concerns
- Layers' Isolation
- Open/Closed Layers

Open/Closed Layers



Pattern Example



The Architecture Sinkhole Anti-Pattern

Requests flow through layers without processing

Layered Architecture Pattern Analysis

- Overall Agility - **Low**
- Ease of Deployment - **Low**
- Testability - **High**
- Performance - **Low**
- Scalability - **Low**
- Ease of Development - **High**

Event Driven Architecture

- Distributed
- Asynchronous
- Highly scalable
- Highly adaptable

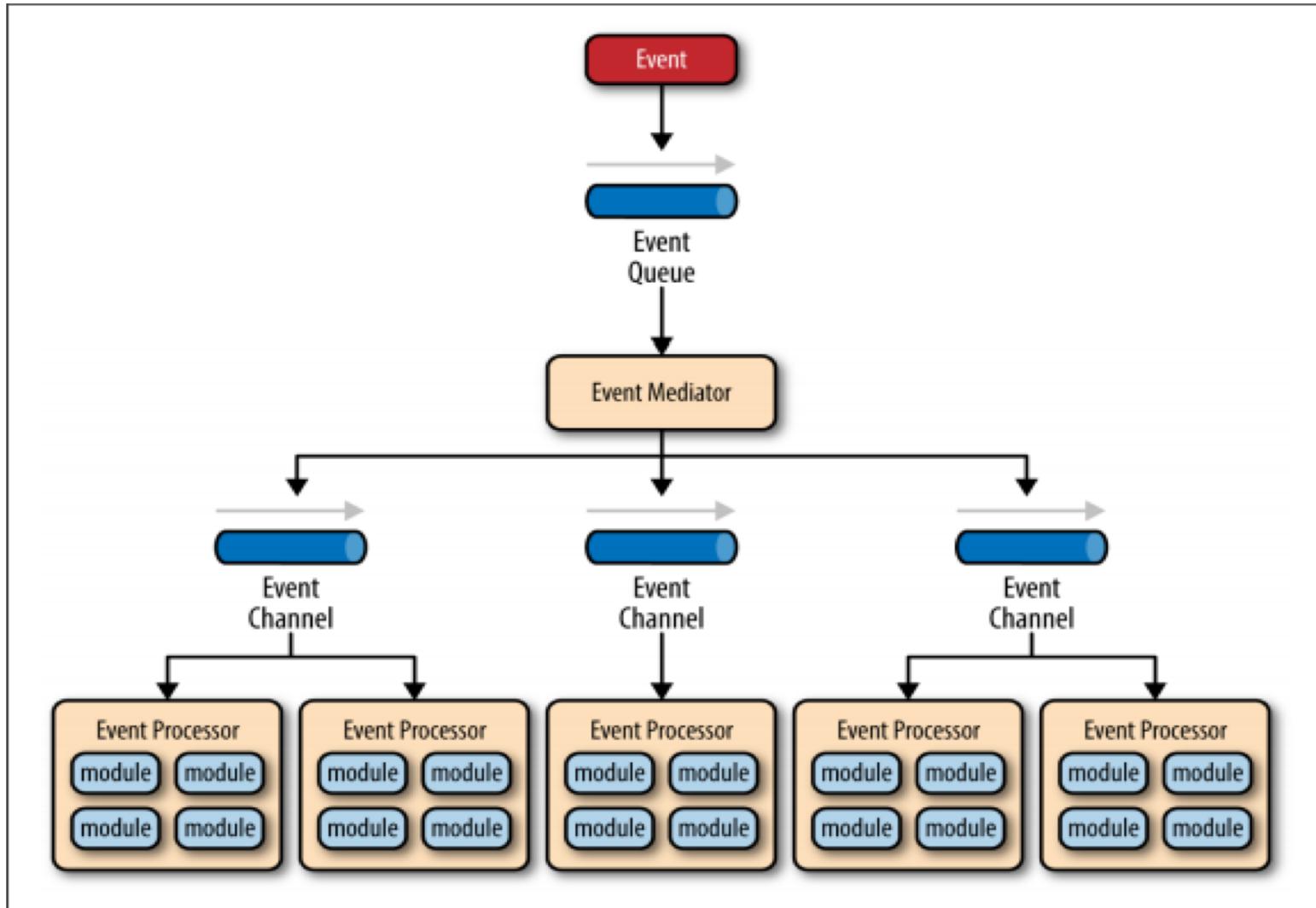
Two main topologies:

- Mediator
- Broker

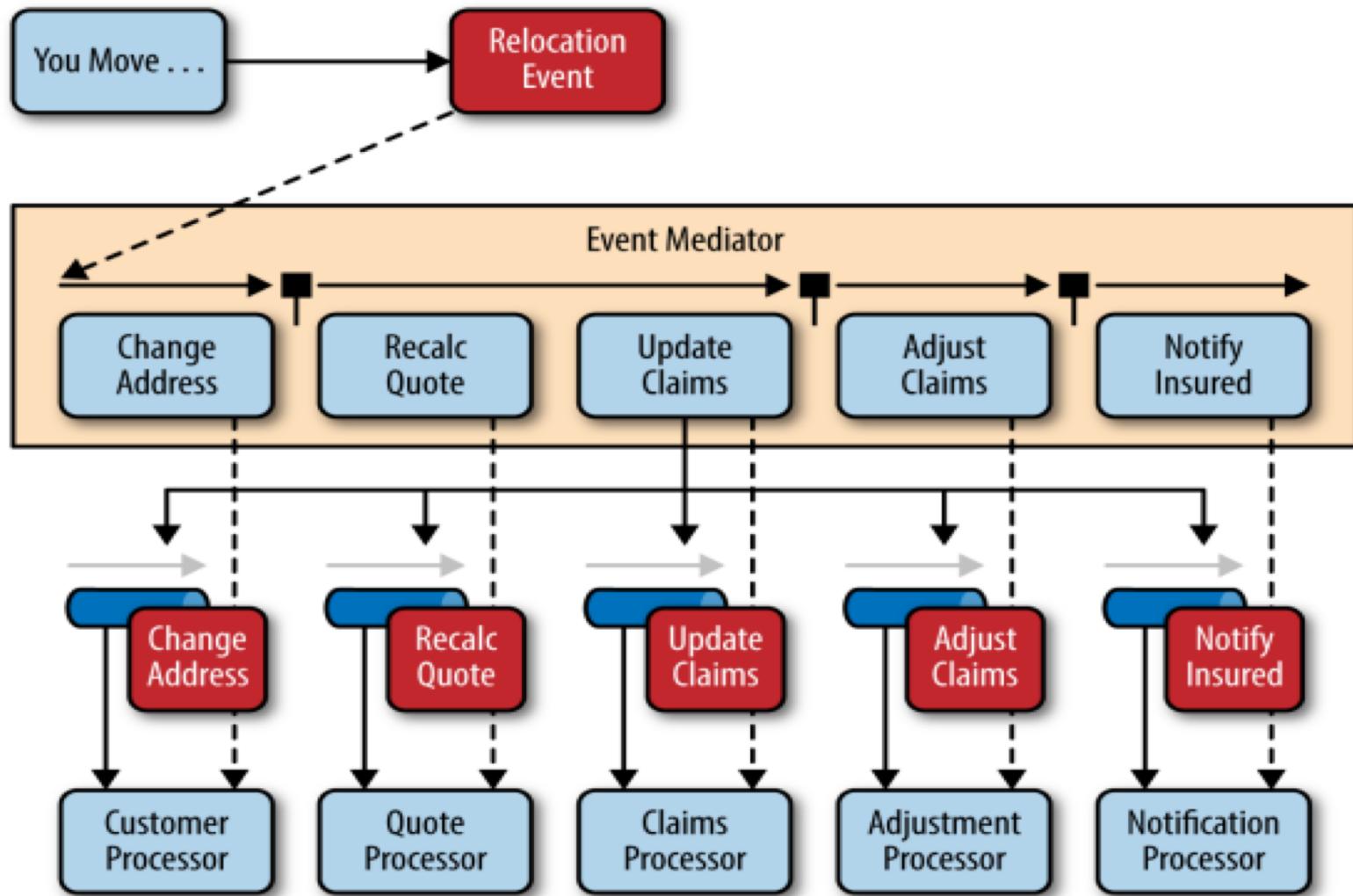
Mediator Topology

- Events processing have multiple steps that require orchestration
- Four main components:
 - Event Queues
 - Event Mediator
 - Event Channels
 - Event Processors

Mediator Topology



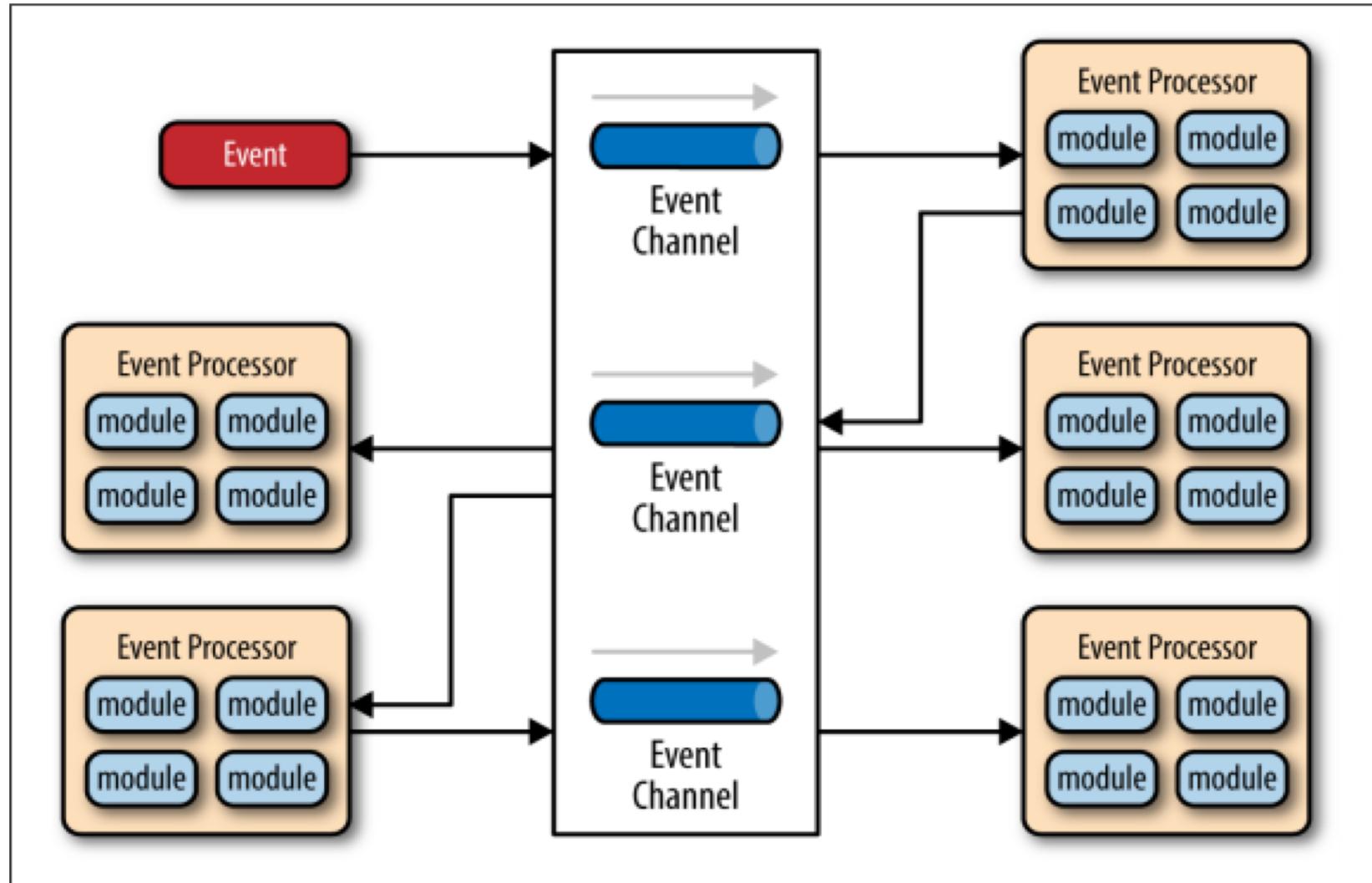
Mediator Topology Example



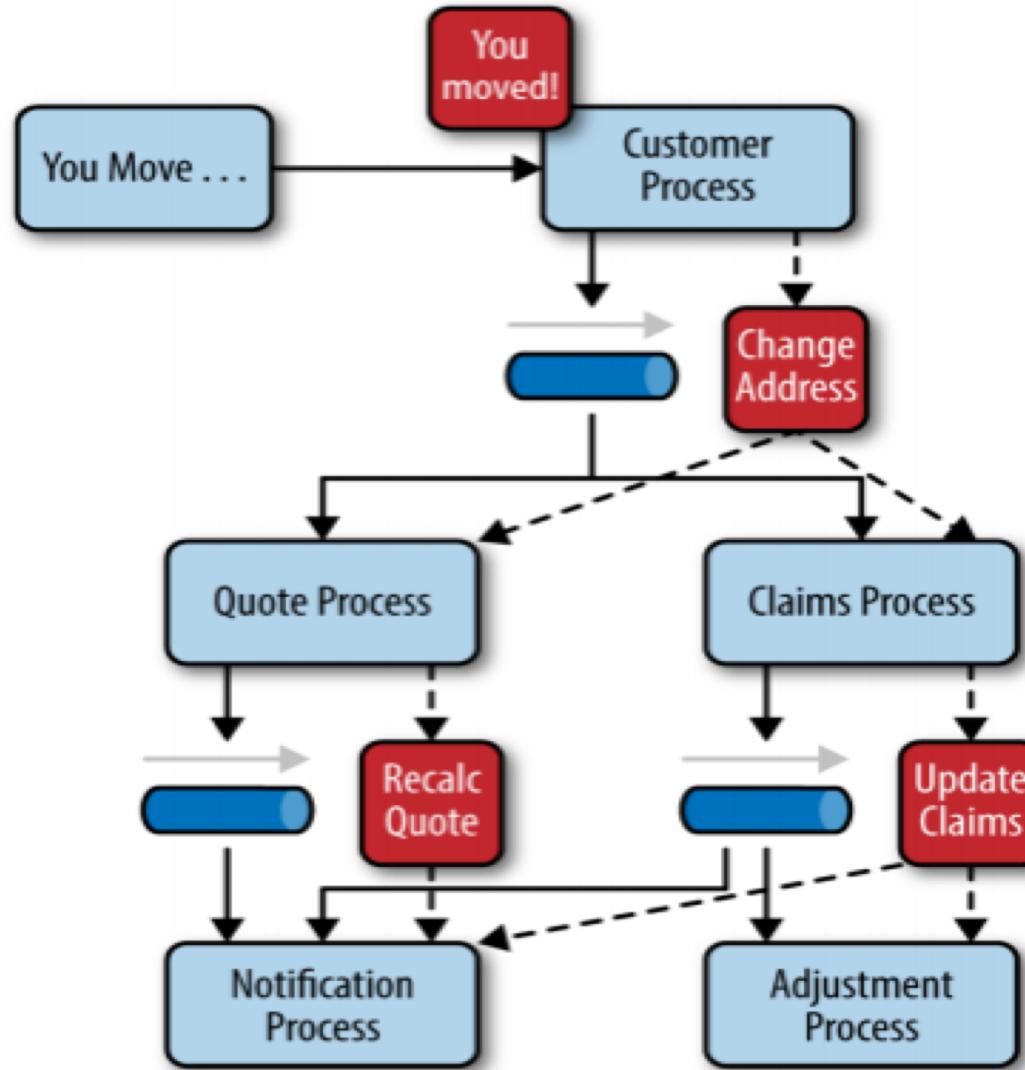
Broker Topology

- No central event mediator
- Message flows across processors in a chain like fashion
- Main components:
 - Message Broker
 - Event Processor

Broker Topology



Broker Topology Example

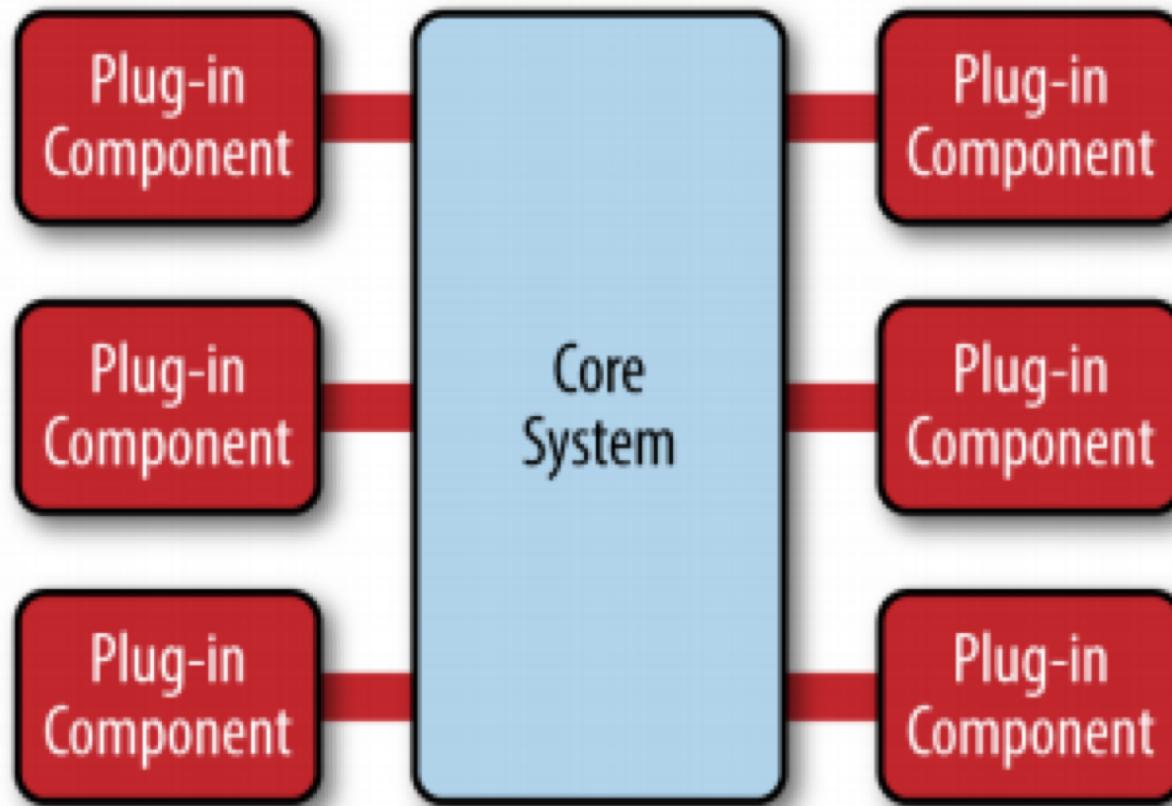


Microkernel Architecture

Microkernel Architecture

- Plugin Architecture Pattern
- Natural for Product Based Apps
- Consists of:
 - Core System
 - Plugins
- Can be embedded in other patterns

Microkernel Architecture

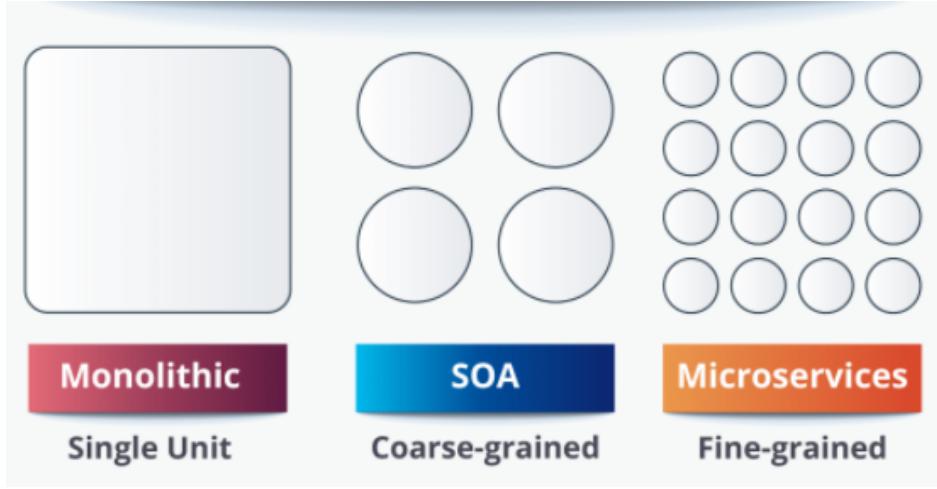


Microkernel Pattern Analysis

- Overall Agility - **High**
- Ease of Deployment - **High**
- Testability - **High**
- Performance - **High**
- Scalability - **Low**
- Ease of Development - **Low**

Micro services Architecture

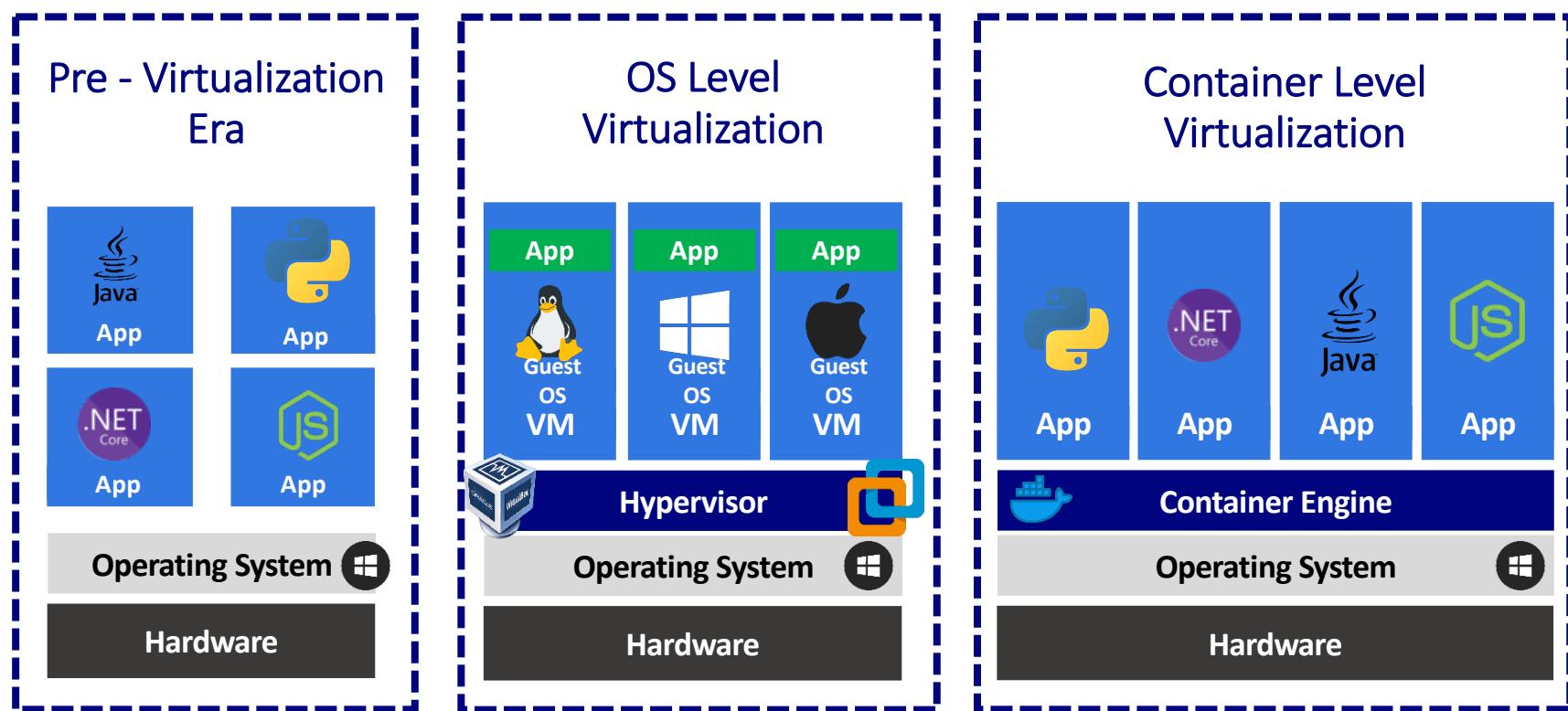
Background



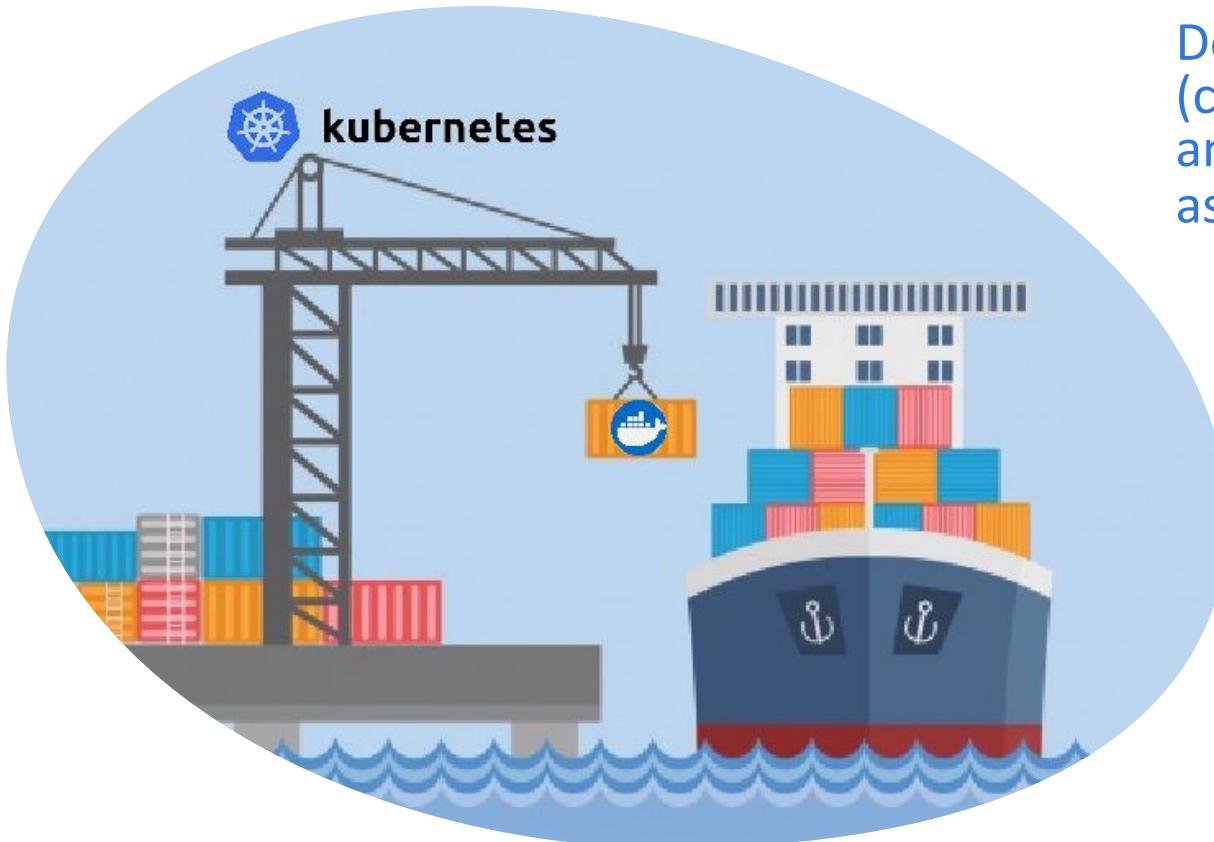
Many leading companies adapted to the micro-services



Roadmap of Microservices



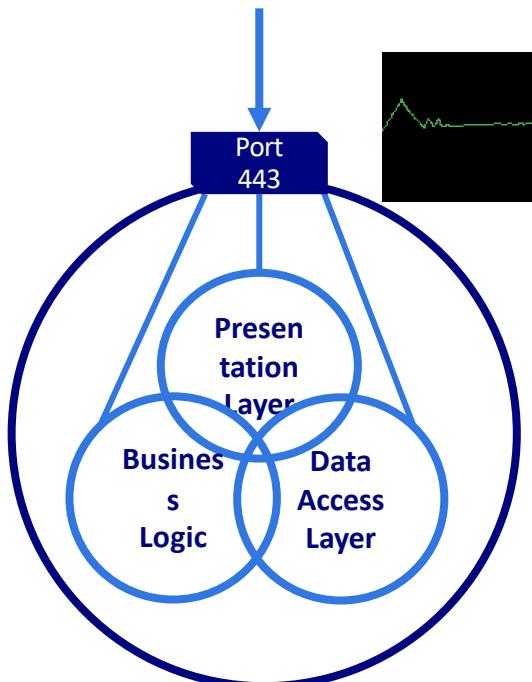
Docker vs Kubernetes



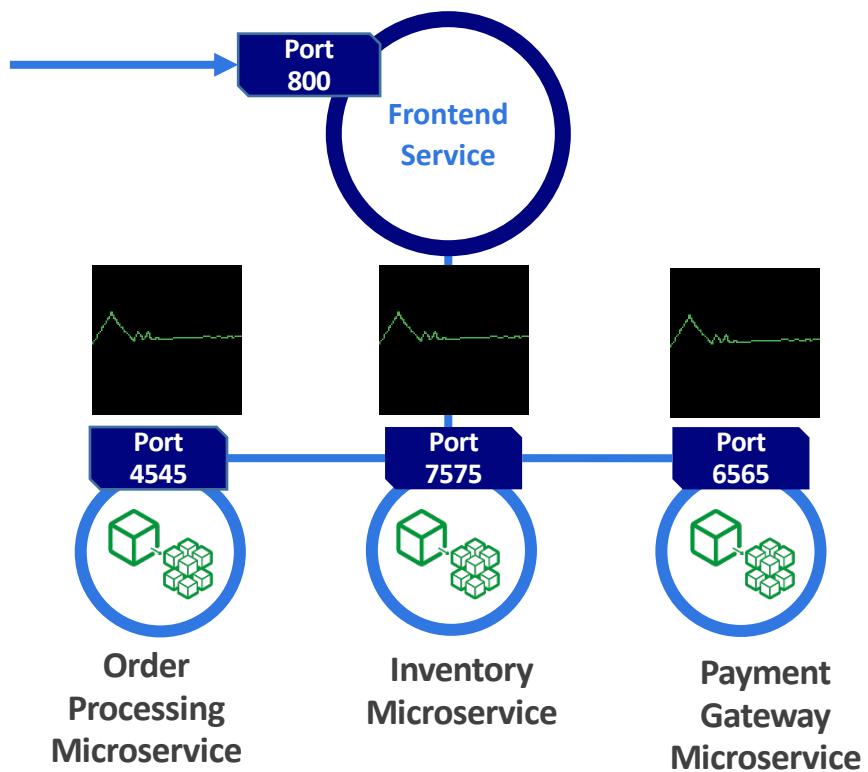
Docker enables (containerization) packaging and running microservices as a lightweight.

Kubernetes provides the container orchestration capabilities to run docker containers.

Monolithic Architecture



Microservices Architecture



Micro Services Architecture

Pattern Analysis

- Overall Agility - High
- Ease of Deployment - High
- Testability - High
- Performance - Low
- Scalability - High
- Ease of Development - High

Pattern Comparison

Layered Pattern	A solid general purpose pattern - best when you are not sure. Avoid the “Sinkhole Anti-Pattern” Tends to encourage Monolithic Applications
Event-Driven	Relatively complex. Distributed architectures issues must be addressed, such as remote processor availability, lack of responsiveness, reconnection logic, and failure recovery. No transactions across processors. Difficult to create and maintain processor contracts
Microkernel	Can be embedded and used within other patterns. Great support for evolutionary design and incremental development. Should always be the first choice for product-based applications
Microservices	Easy to perform real-time production deployments. Very agile-oriented architecture Shares complexity issues with data-driven pattern

Pattern Comparison

	Layered	Event-driven	Microkernel	Microservices
Overall Agility	⬇️	⬆️	⬆️	⬆️
Deployment	⬇️	⬆️	⬆️	⬆️
Testability	⬆️	⬇️	⬆️	⬆️
Performance	⬇️	⬆️	⬆️	⬇️
Scalability	⬇️	⬆️	⬇️	⬆️
Development	⬆️	⬇️	⬇️	⬆️

The End

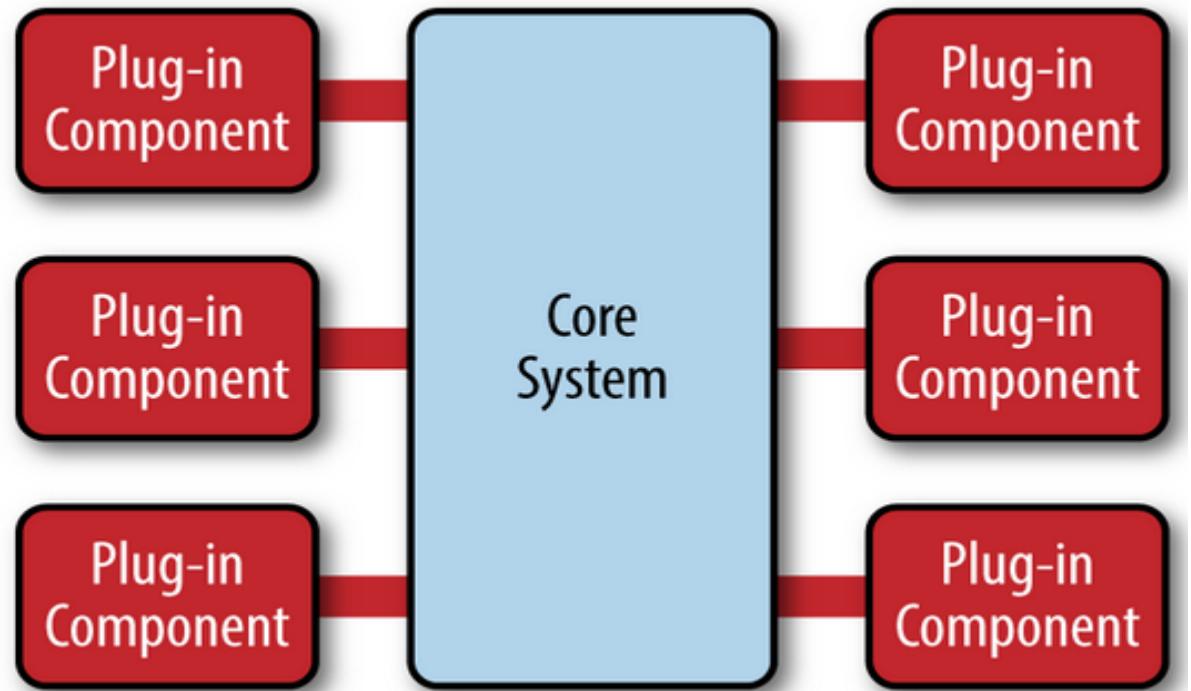


Microkernel Architecture (Plugin-architecture Pattern)

**Software Architecture
3rd Year – Semester 1
By Udara Samaratunge**

Plugin-architecture Pattern

- Two Components
 - Core System
 - Plugin module
- The plug-in modules are stand-alone, independent components.
- **Core system** needs to know about which plug-in modules are available (uses Plugin registry)
- OSGi represents Microkernel Architecture for its plugin development.



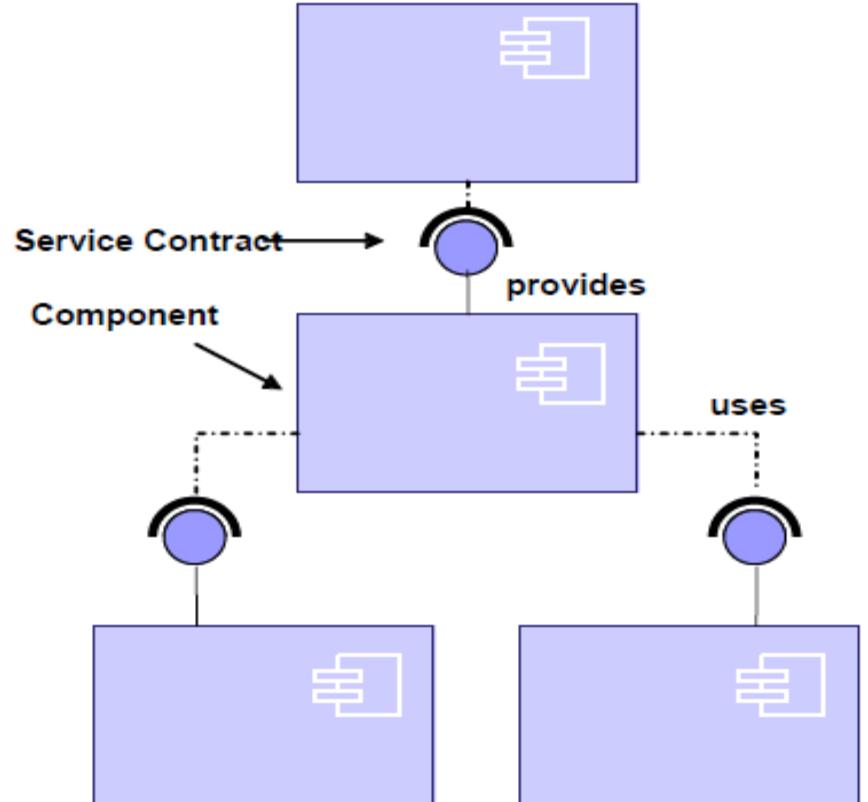
What is OSGi?

- OSGi (Open Services Gateway initiative)
- OSGi is a **framework** which allows **modular development of applications using java**.
- A Java framework for developing (remotely) deployed service applications, that require:
 - Reliability
 - Large scale distribution
 - Wide range of devices
 - Collaborative
- Created through a collaboration of industry leaders
 - IBM, Ericsson, Nokia, Sony, Telcordia, Samsung, ProSyst,
 - Gatespace, BenQ, Nortel, Oracle, Sybase, Espial, and many more

- OSGi containers allow you to **break your application into individual modules**. (are jar files with additional meta information and called **bundles** in OSGi terminology)
- Manage the **cross-dependencies** between modules.
- An OSGi framework then offers you dynamic loading/unloading, configuration and control of these bundles - without requiring restarts.
- Major Framework vendors are
 - ProSyst,
 - Gate space Telematics, and
 - IBM
 - Siemens
 - Espial
- Open source implementations
 - Apache Felix
 - Eclipse Equinox
 - Gate space Knopflerfish

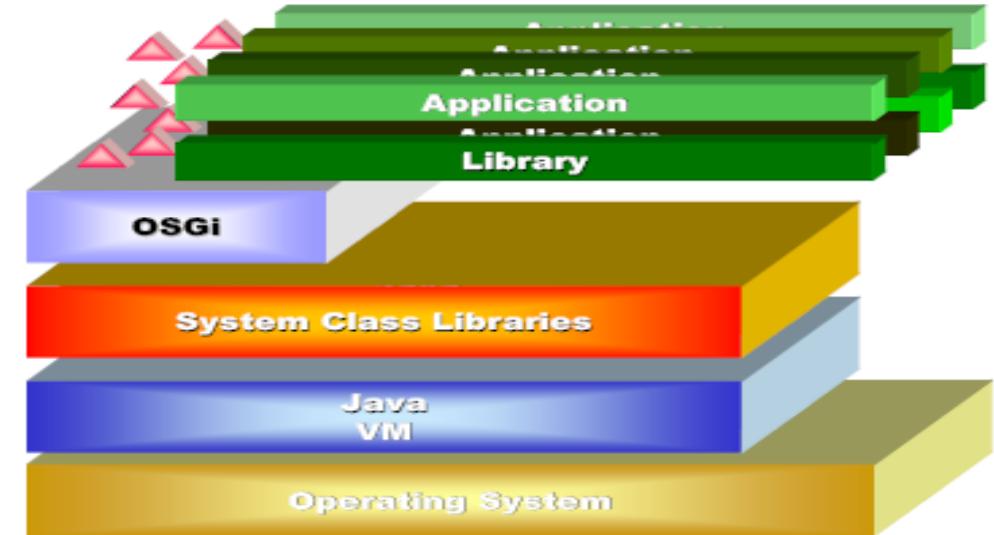
The OSGi Platform vs Service Oriented Architectures (SOA)

- Separate the **contract** from the **implementation**
- Allows alternate implementations
- Dynamically discover and **bind** available implementations
- Binding based on **contract** (interface definitions)
- Components are **reusable**
- Components are not coupled to implementation details of other components, only their independent interfaces have to be known

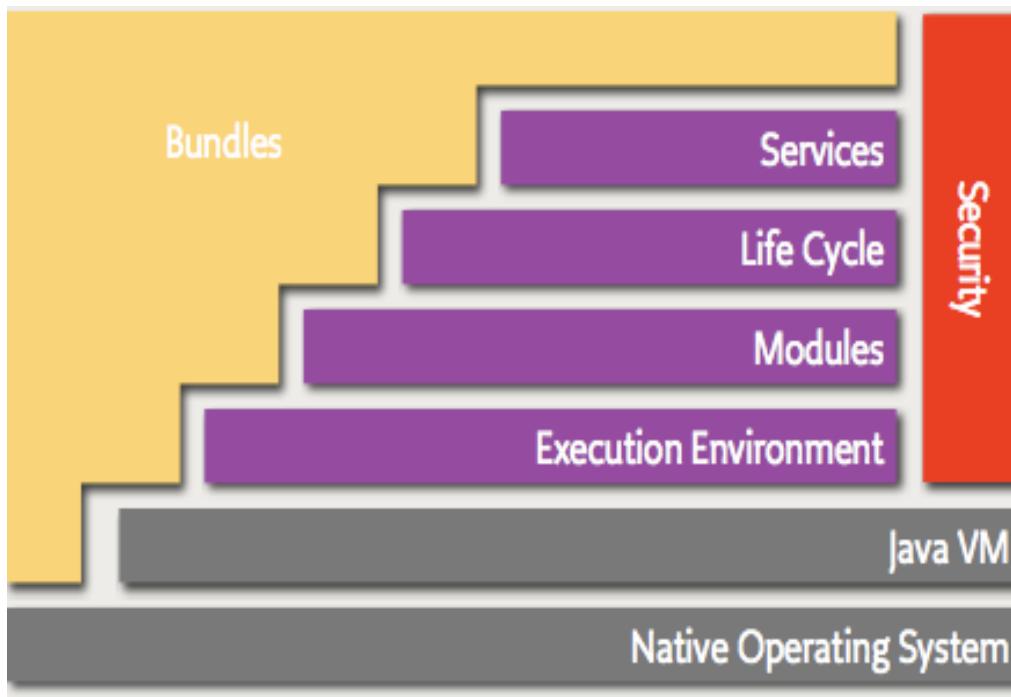


The OSGi Framework Architecture

- Allows applications to share a single Java VM.
- Handles all class loading in a much better defined way than standard Java.
 - Versioning!
- Gives **isolation/security** between applications.
- Mediates between communication & collaborations between applications.
- Provides life cycle management (install, start, stop, update, etc).
- Policy free
 - Policies are provided by bundles



OSGi Bundle Architecture



- The following list contains a short definition of the terms:
 - **Bundles** - Bundles are the OSGi components made by the developers.
 - **Services** - The services layer connects bundles in a dynamic way by offering a *publish-find-bind model for plain old Java objects*.
 - **Life-Cycle** - The API to install, start, stop, update, and uninstall bundles.
 - **Modules** - The layer that defines how a bundle can import and export code.
 - **Security** - The layer that handles the security aspects.
 - **Execution Environment** - Defines what methods and classes are available in a specific platform.

What is a Bundle?

- In Java terms, a **bundle** is a plain old JAR file.
- In standard Java everything in a JAR is completely visible to all other JARs.
- But OSGi **hides everything** in that JAR **unless explicitly exported**.
- **Reason for hiding** is to maintain multiple versions of the same library.
- **By default**, there is **no sharing**.

Practical Example

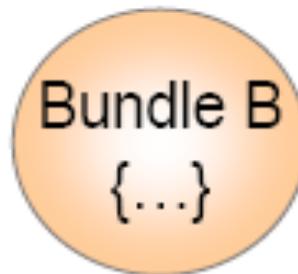
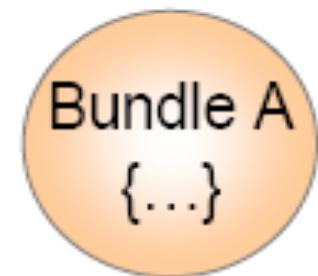
Eg :- I have an application that is interacting with underlying MySQL database and after few month I found that MySQL team has fixed a major bug in their new version of mysql-connector library release so in order to incorporate this new library in my traditional application I have to **stop my application** and **re-package it (or just replace the older one)**

But, with **OSGI** we **don't need to stop the whole application** because **everything is exposed** either as a component or as a service therefore we **just need to install new component/service in OSGI container.**

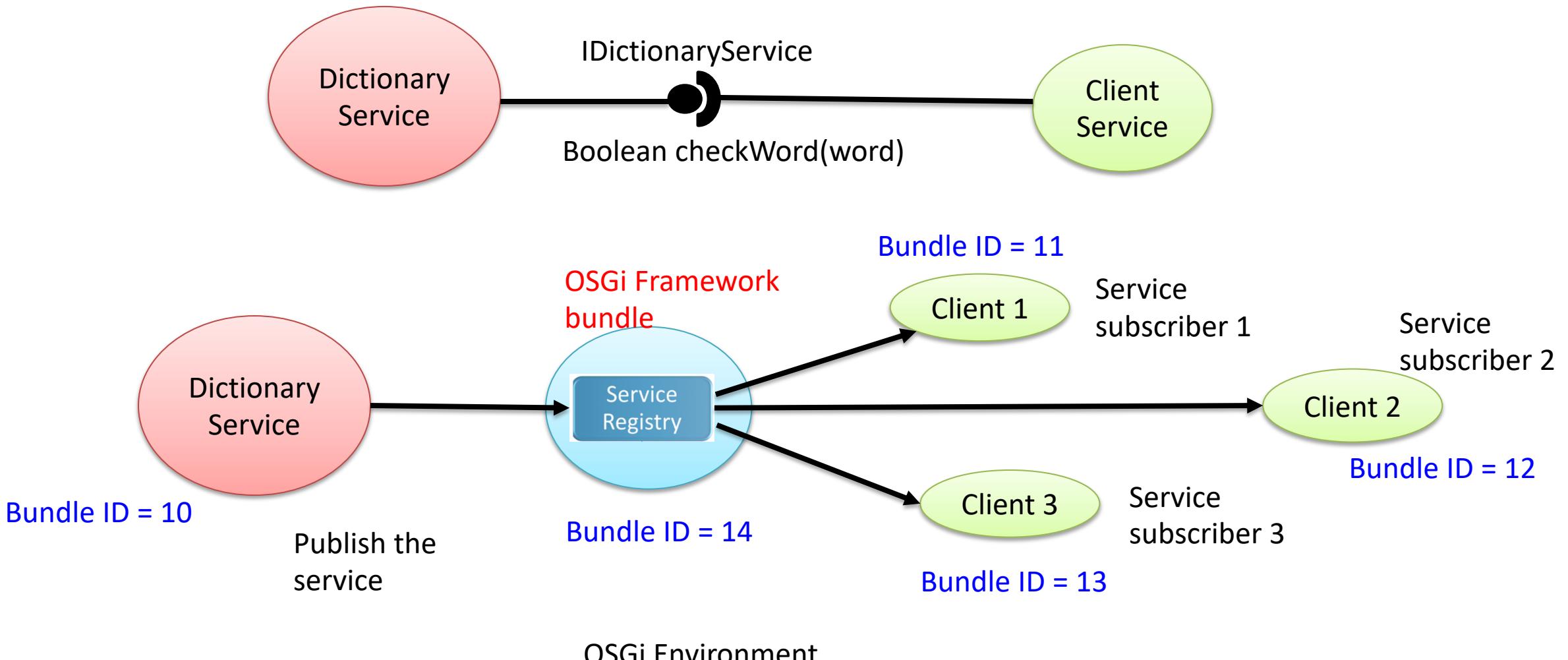
when the **services/components** are updated in OSGI container there are **various event listeners** that **propagate the service/component update event** to service/component consumers.

And accordingly consumers adapts themselves to use new version of web service (on the consumer side **we need to listen for various events** so that consumers can decide whether to respond for change or not.

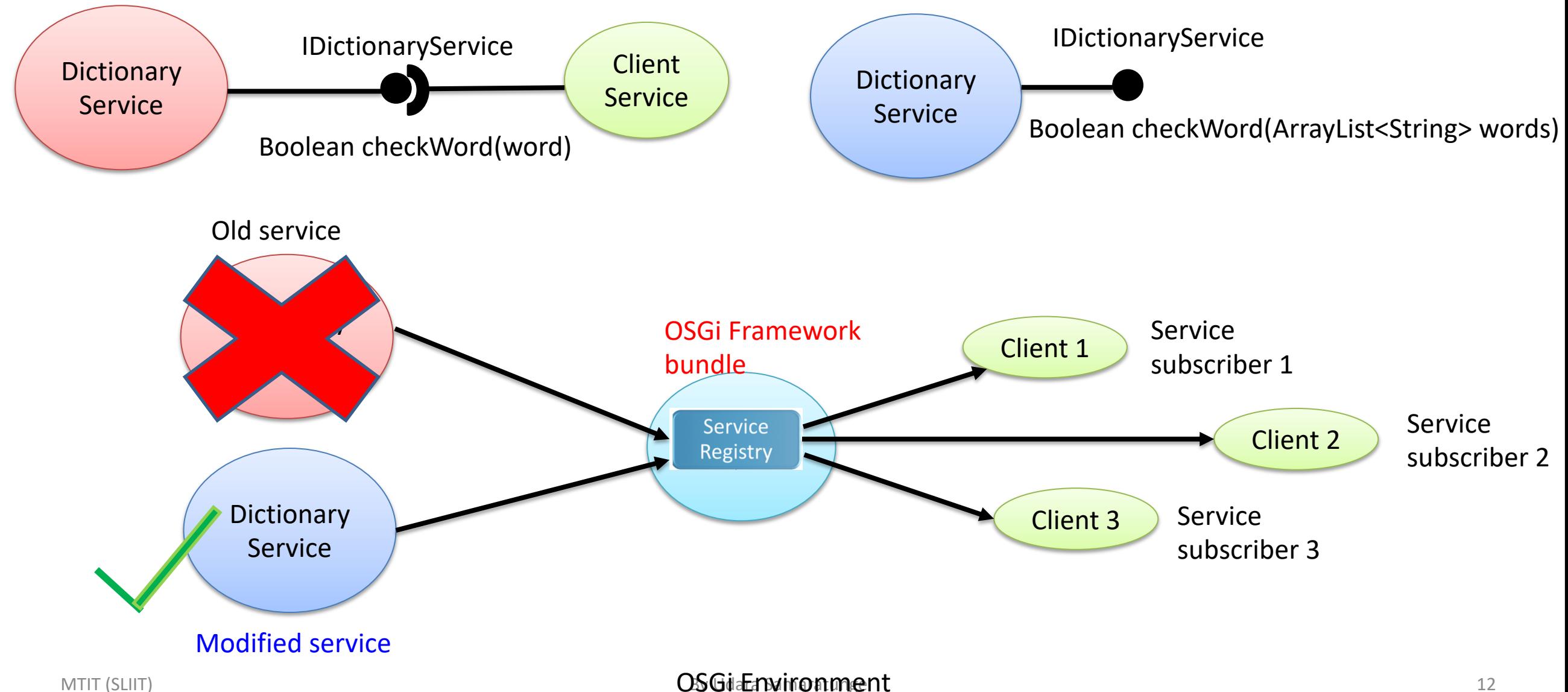
- A *bundle* is the deliverable application
 - Like a Windows EXE file
 - Content is a JAR file
- A bundle registers zero or more services
 - A service is specified in a Java interface and may be implemented by multiple bundles
 - Services are bound to the bundle life-cycle
- Searches can be used to find services registered by other bundles
 - Query language



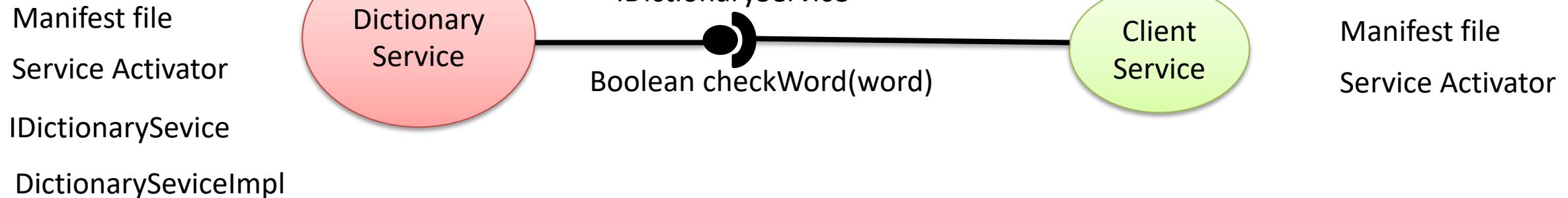
Practical Example



Practical Example



OSGi Environment



Dictionary Service Manifest file

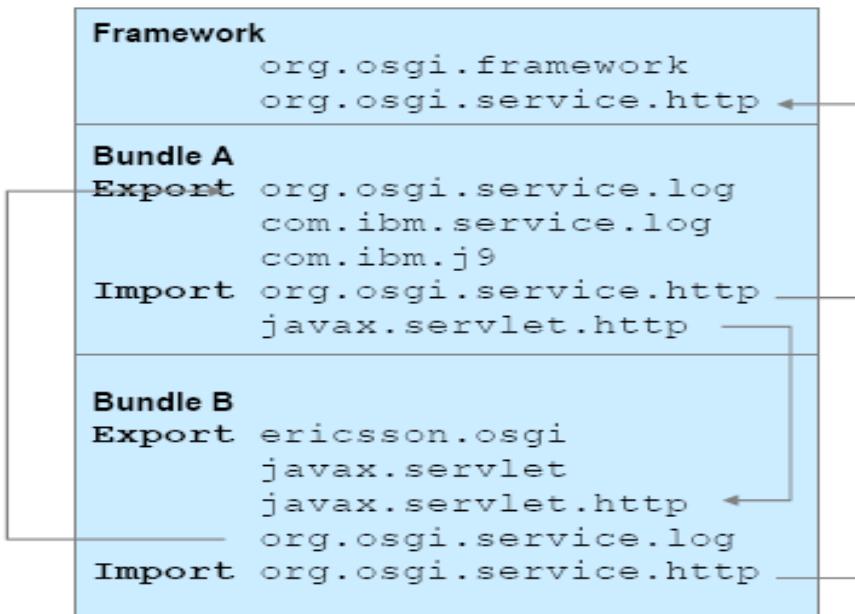
```
Bundle-Name: English dictionary
Bundle-Description: A bundle that registers an English dictionary service
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Bundle-Activator: tutorial.example2.Activator
Export-Package: tutorial.example2.service
Import-Package: org.osgi.framework
```

Client Manifest file

```
Bundle-Name: Service Tracker-based dictionary client
Bundle-Description: A dictionary client using the Service Tracker.
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Bundle-Activator: tutorial.example5.Activator
Import-Package: org.osgi.framework, org.osgi.util.tracker, tutorial.example2.service
```

Bundle Deployment

- Bundles are deployed on an *OSGi framework*, the bundle runtime environment.
- This is not a container like Java Application Servers. It is a *collaborative environment*.

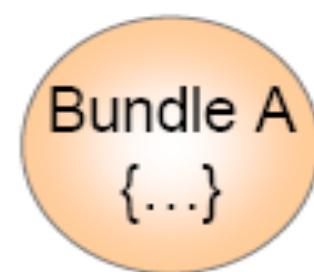


A resolved

B resolved

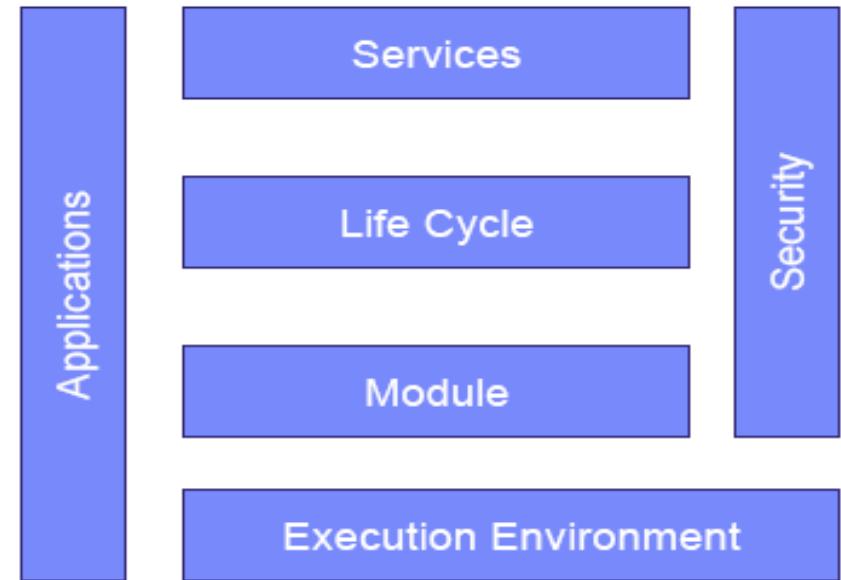
- Bundles run in the same VM and can actually share code.
- The framework uses the *explicit imports* and *exports* to wire up the bundles so they do not have to concern themselves with class loading.

- A **Bundle** contains (normally in a JAR file):
 - Manifest (bundle meta data)
 - Code (classes in packages)
 - Resources (other files in the JAR file)
- The Framework:
 - Reads the bundle's manifest
 - Installs the code and resources
 - Resolves dependencies
 - Controls the bundle life cycle
- During Runtime:
 - Calls the Bundle Activator to start the bundle
 - Manages java class path for the bundle as a network of class loaders
 - Handles the service dependencies
 - Calls the Bundle Activator to stop the bundle
 - Cleans up after the bundle



OSGi Service Platform Layering

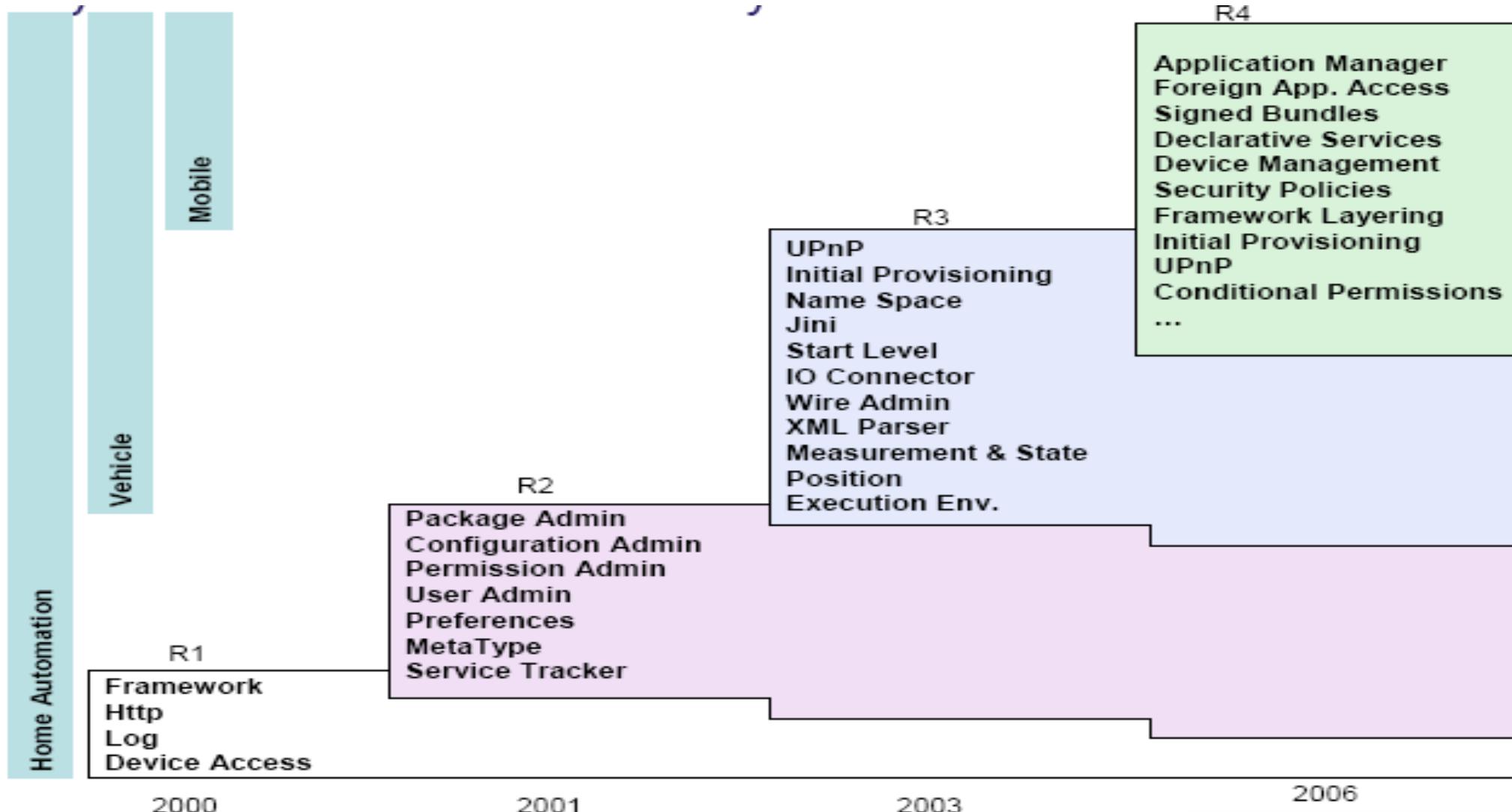
- The OSGi Service Platform is divided in a number of layers.
- Execution Environment provides a defined context for applications.
- The Module layer provides class loading and packaging specifications.
- The Services layer provides a collaboration model.
- The extensive Security layer is embedded in all layers.



OSGi Service Layer

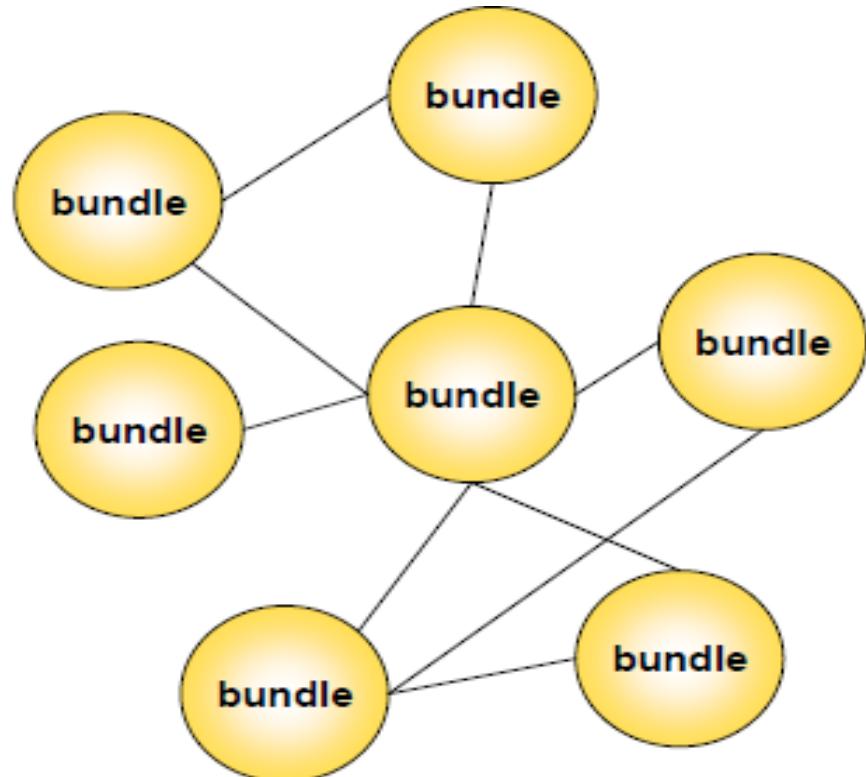
- Provides an inside-VM service model
 - Discover (and get notified about) services based on their interface or properties, no protocol required
 - Bind to one or more services by
 - program control,
 - default rules, or
 - deployment configuration
- Service Oriented Architectures (SOA) Confusion
 - Web services bind and discover over the net
 - The OSGi Service Platform binds and discovers inside a Java VM
- The OSGi Alliance provides many standardized services
- OSGi defines a standard set of services
 - Other organizations can define more (AMI-C, Ertico, JCP)

The OSGi Service Layer Evolution

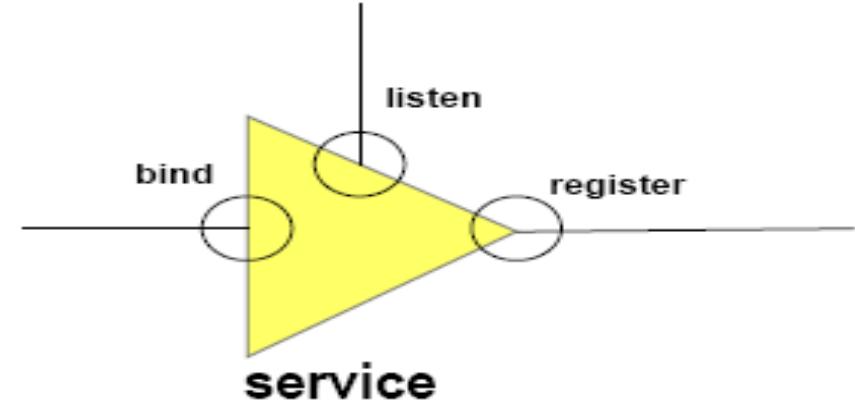
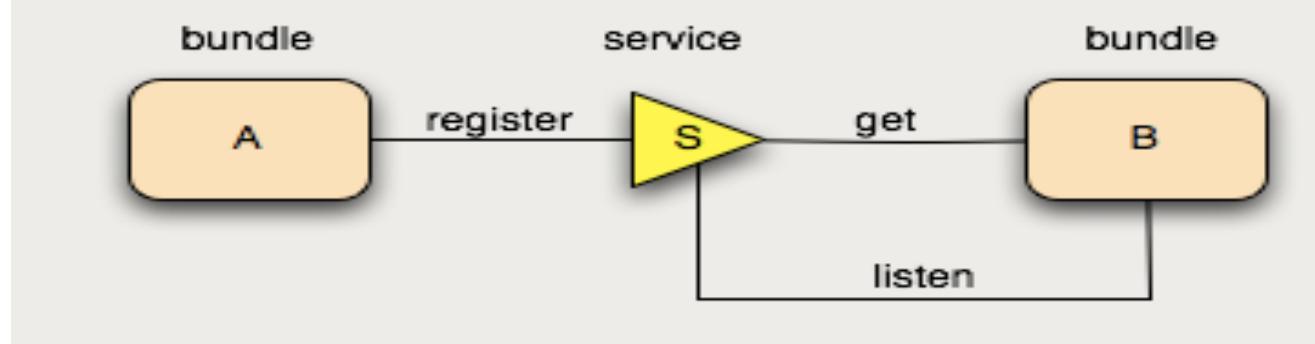


OSGi Module Layer

- Packaging of applications and libraries in Bundles
 - Java has significant deployment issues
- Class Loading modularization
 - Java provides the Class Path as an ordered search list, which makes it hard to control multiple applications
- Protection
 - Java can not protect certain packages and classes from others.
- Versioning
 - Java can not handle multiple versions of the same package in a VM



OSGi Services

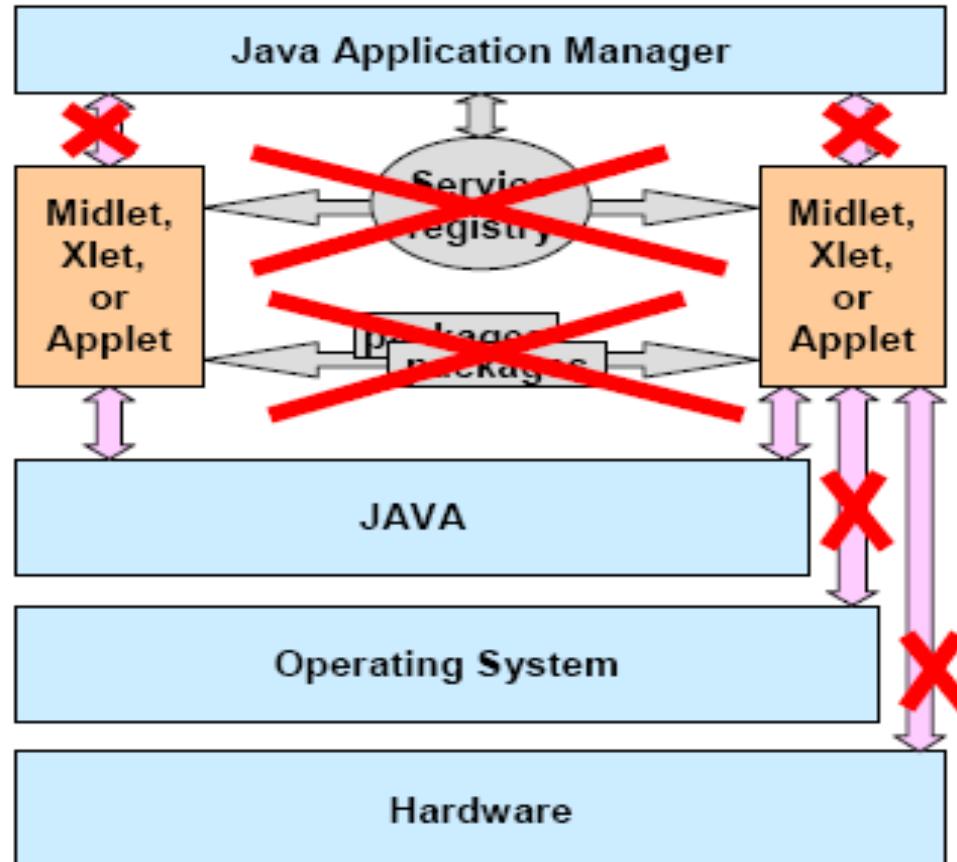
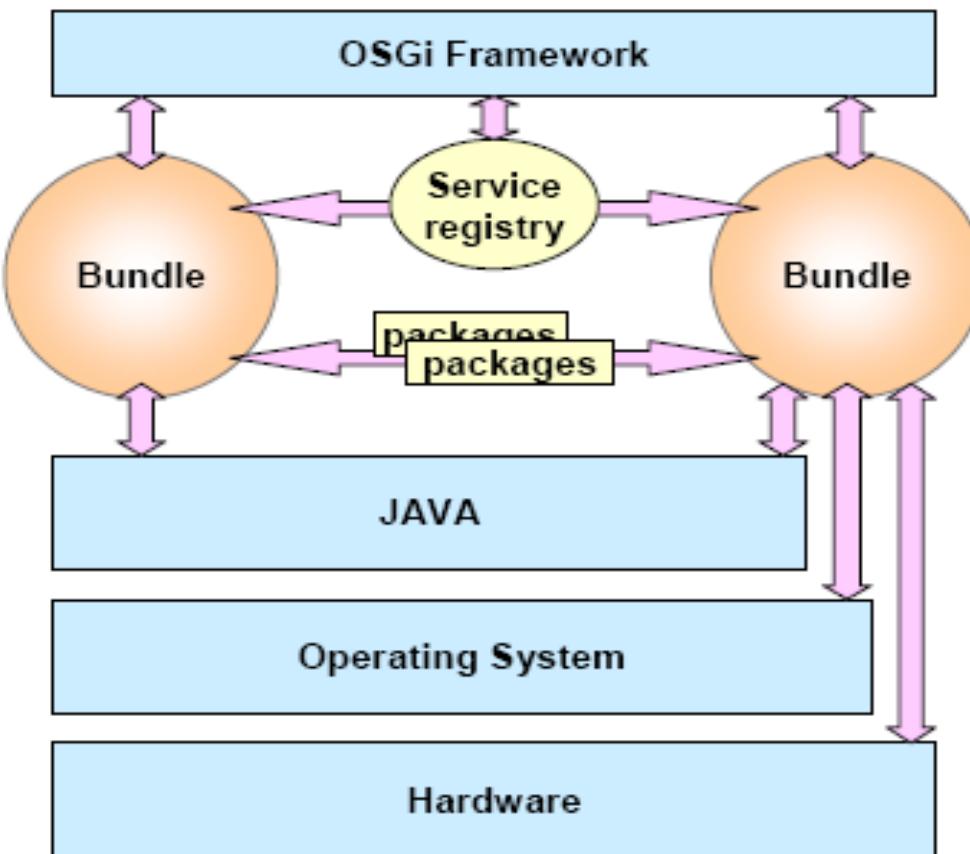


- The Framework **Service Registry** is available to **all bundles** to collaborate with other bundles.
- **Different bundles** (from different vendors) can implement the **same interface**
 - Implementation is not visible to users
 - Allows operator to replace implementations without disrupting service
- A bundle can create an object and register it with the **OSGi service registry** under one or more interfaces.

- Other bundles can go to the **registry** and list all objects that are registered under a specific interfaces or class.
- A bundle can therefore *register a service*, it can *get a service*, and it can *listen for a service* to appear or disappear.
- **Multiple bundles** can register objects under the same interface or class with the same name.
- Services are associated with **properties**
 - Powerful query language to find appropriate service
 - Bundles can update the properties of a service dynamically
- A **bundle** can **decide** to withdraw its service from the **registry** while other bundles are still using this service

- A bundle can use a service (bind to) with any cardinality
 - 1..1, 0..1, 0..n
- A service can be discovered dynamically
 - Active search with query filter
 - Listener interface
- Services are dynamic
 - A bundle can decide to withdraw its service from the registry while other bundles are still using this service. Bundles using such a service must then ensure that they no longer use the service object and drop any references.
- Services can go away at any time! This is very dynamic!

Component interaction and collaboration



- No management bundles
- No collaboration
- No package management (versions!)
- No native code

Functionalities supported by OSGi

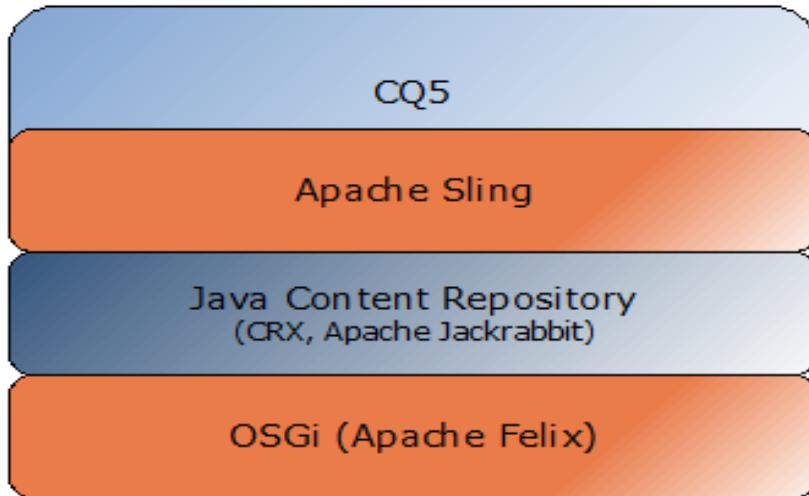
- 1) Reduces the complexity of the system.
- 2) Managing service/component dependencies.
- 3) Makes the components loosely-coupled and easy to manage.
- 4) Increases the performance of the system.

OSGi increase performance.

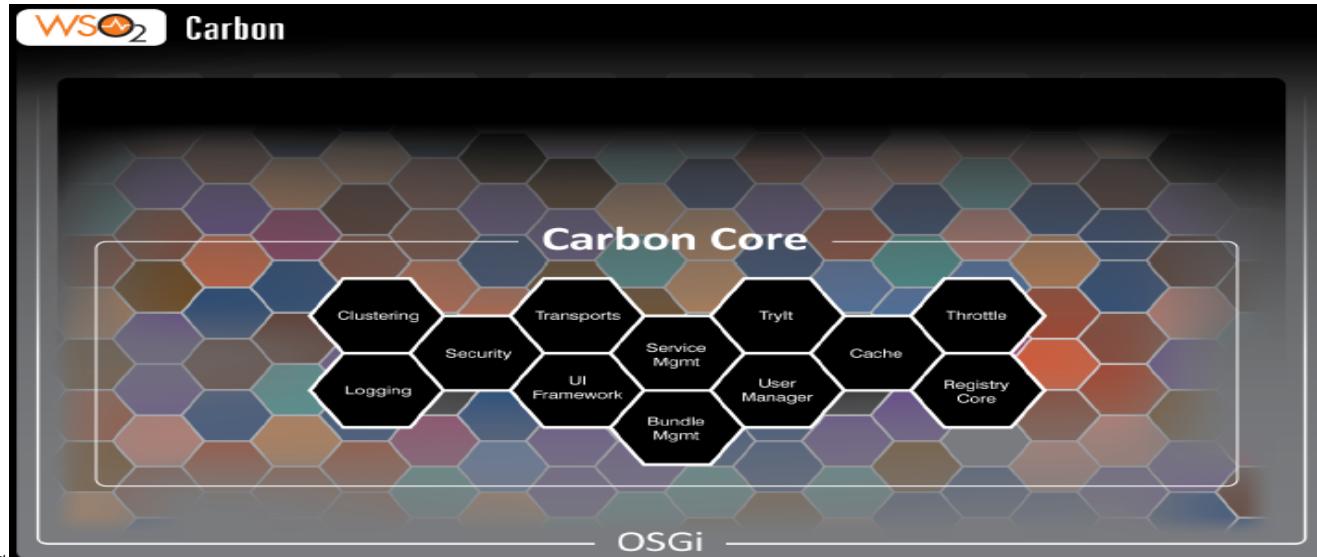
- Consider a scenario where you have a large application which uses a logging framework.
- This **logging framework** can be deployed as an OSGi Bundle, which **can be managed independently**.
- Therefore, it can be started when required by our application and can be stopped when not in use.
- Also the OSGi container makes these **bundles available as services**, which can be subscribed by other parts of application.

Custom products uses OSGi

Adobe CQ (CMS app)



- CQ5, uses the Apache Felix implementation of OSGi.
- Apache Felix is a open-source project to implement the OSGi R4 Service Platform.
- That includes
 - OSGi framework and standard services.
 - OSGi-related technologies.

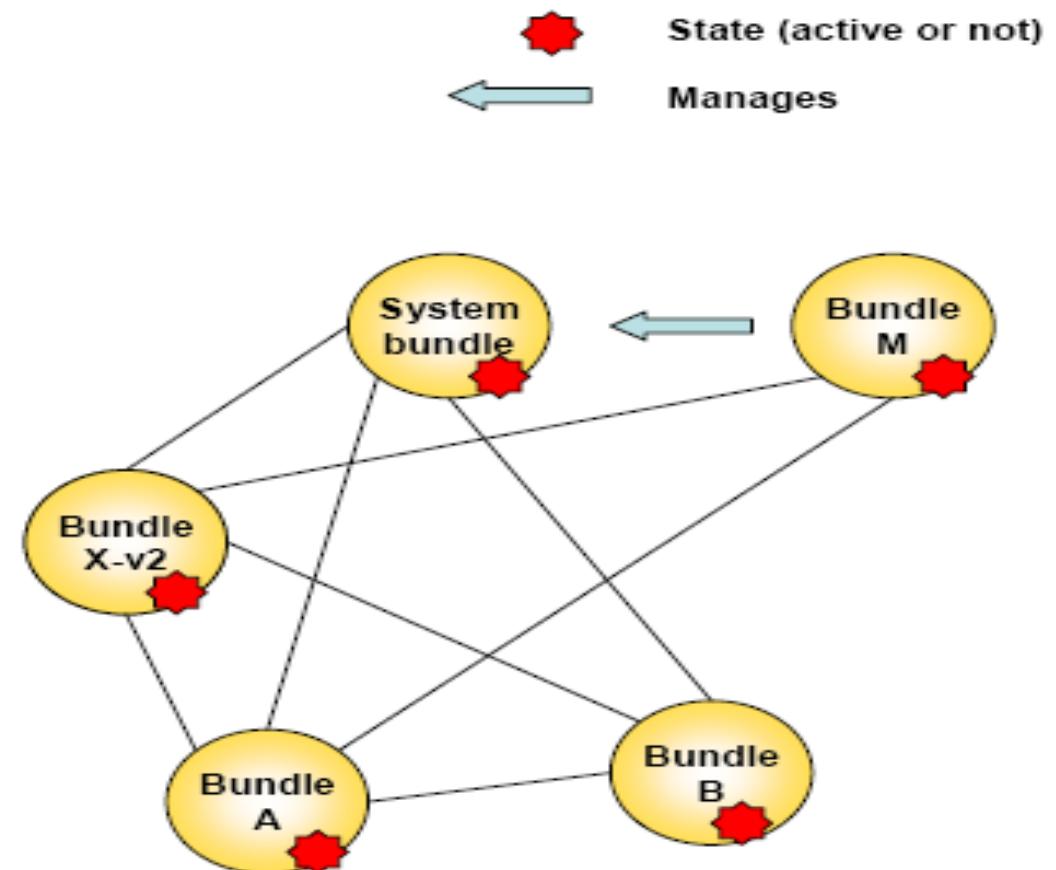


What is a Carbon Component

- A set of OSGi Bundles.
- Lives in the Carbon Framework. Hence should conform to rules define in the Carbon Framework.
- Develop the Carbon component
 - . Back-end component (BE OSGi bundles)
 - . Front-end component (FE OSGi bundles)
 - . Common bundles, if any

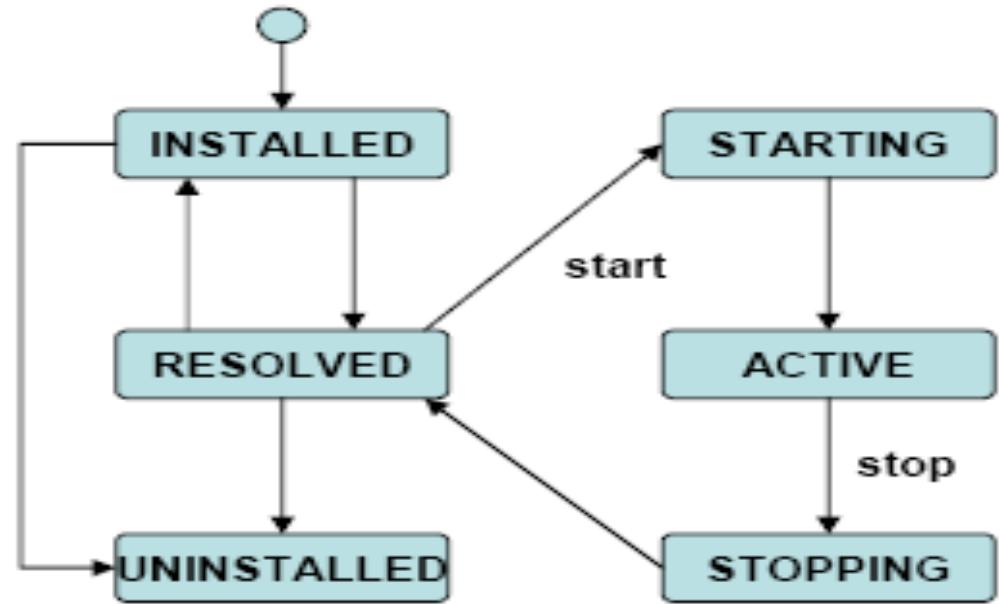
Layers: OSGi Life Cycle Layer

- System Bundle represents the OSGi Framework
- Provides an API for managing bundles
 - Install
 - Resolve
 - Start
 - Stop
 - Refresh
 - Update
 - Uninstall



Layers: OSGi Life Cycle Layer

- Bundle is started by the *Bundle Activator* class
- Header in the JAR manifest file refer to this class
- *Bundle Activator* interface has 2 methods
 - Start: Initialize and return immediate
 - Stop: Cleanup
- The *Bundle Activator* gets a *Bundle Context* that provides access to the OSGi Framework functions.
- The Framework provides the Start Level service to control the start/stop of groups of applications.



OSGI life cycle methods

org.osgi.framework

BundleActivator

- start(BundleContext) : void
- stop(BundleContext) : void

States of bundles

- Installed – finish bundle installation
- Active – start the bundle
- Resolved – stop the bundle

org.osgi.framework

BundleContext

- addBundleListener(BundleListener) : void
- addFrameworkListener(FrameworkListener) : void
- addServiceListener(ServiceListener) : void
- addServiceListener(ServiceListener, String) : void
- createFilter(String) : Filter
- getAllServiceReferences(String, String) : ServiceReference
- getBundle() : Bundle
- getBundle(long) : Bundle
- getBundles() : Bundle[]
- getDataFile(String) : File
- getProperty(String) : String
- getService(ServiceReference) : Object
- getServiceReference(String) : ServiceReference
- getServiceReferences(String, String) : ServiceReference
- installBundle(String) : Bundle
- installBundle(String, InputStream) : Bundle
- registerService(String[], Object, Dictionary) : ServiceRegistration
- registerService(String, Object, Dictionary) : ServiceRegistration
- removeBundleListener(BundleListener) : void
- removeFrameworkListener(FrameworkListener) : void
- removeServiceListener(ServiceListener) : void
- ungetService(ServiceReference) : boolean

Manipulating Services

- The *Bundle Context* provides the methods to manipulate the service registry
- Services registrations are handled by *Service Registration* objects
 - They can be used to unregister a service or modify its properties
- *Service Reference* objects give access to the service as well as to the service's properties
- Access to service objects is through the *getService* method. These services should be returned with the *ungetService* method.

```
ServiceRegistration registerService(  
    String clazz,  
    Object srvc,  
    Dictionary props)  
  
ServiceReference[]  
getServiceReferences(  
    String clazz,  
    String filter)  
  
Object getService(  
    ServiceReference reference)  
  
boolean ungetService(  
    ServiceReference reference);
```

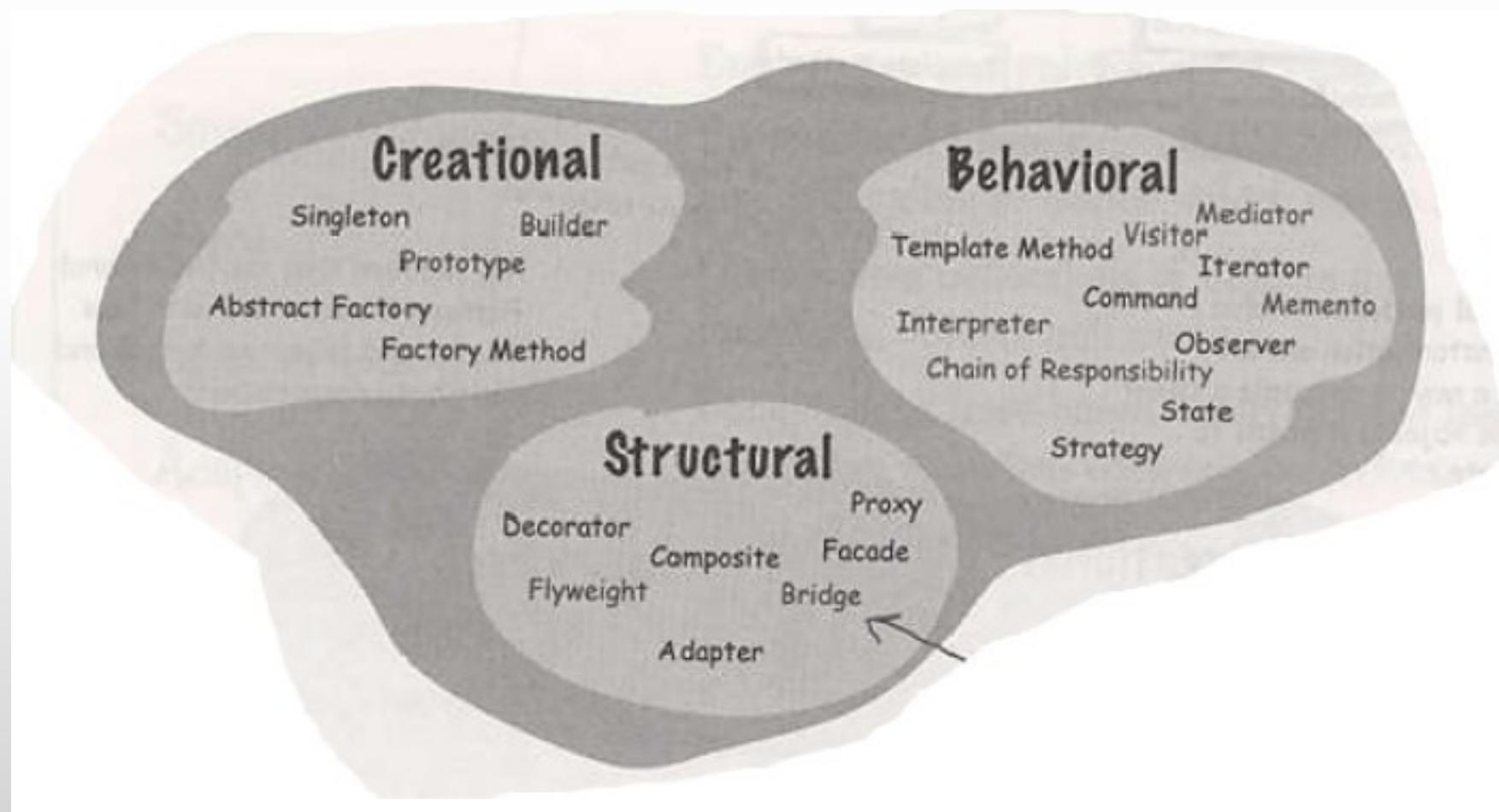
References

- <http://www.osgi.org/Main/HomePage>
- <http://www.osgi.org/Technology/WhatIsOSGi>
- <http://en.wikipedia.org/wiki/OSGi>
- <http://grepcode.com/file/repository.grepcode.com/java/eclipse.org/3.5/org.eclipse osgi/3.5.0/org/osgi/framework/BundleContext.java#BundleContext.getBundle%28long%29>
- <http://felix.apache.org/site/apache-felix-framework-usage-documentation.html>

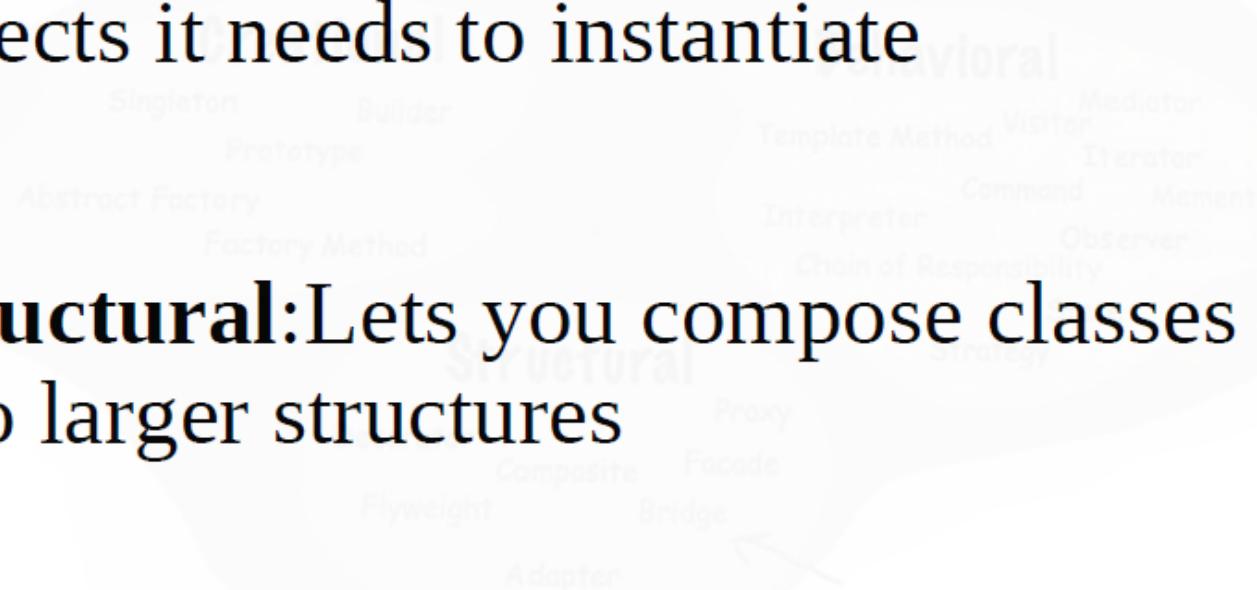
Design Patterns

Software Architecture (SA)
3rd Year – Semester 1
By Udara Samaratunge

Fundamental Design Patterns (Gang Of Four - GOF)



- **Creational:** Involve object initialization and provide a way to decouple client from the objects it needs to instantiate



- **Structural:** Lets you compose classes or objects into larger structures
- **Behavioral:** Concerned with how classes and objects interact or distribute responsibility

Gang of Four Patterns

Creational :

- ⌚ Abstract Factory
- ⌚ Builder
- ⌚ Factory Method
- ⌚ Prototype
- ⌚ Singleton

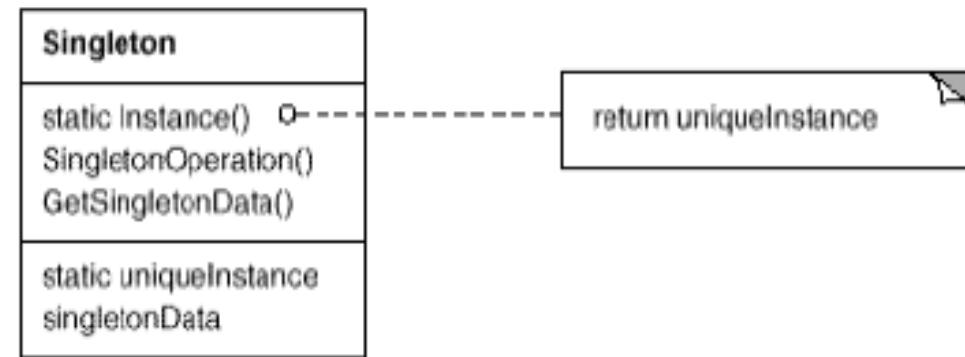
Behavioral :

- ⌚ Strategy
- ⌚ Observer
- ⌚ Command
- ⌚ Interpreter
- ⌚ Iterator
- ⌚ Mediator
- ⌚ Memento
- ⌚ Chain of Responsibility
- ⌚ State
- ⌚ Template Method
- ⌚ Visitor

Structural :

- ⌚ Adapter
- ⌚ Bridge
- ⌚ Composite
- ⌚ Decorator
- ⌚ Facade
- ⌚ Flyweight
- ⌚ Proxy

Singleton Pattern



Ensure a class only has one instance, and provide a global point of access to it

Singleton

```
public class Singleton {  
    private static Singleton uniqueinstance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
  
        if (uniqueinstance == null) {  
            uniqueinstance = new Singleton();  
        }  
        return uniqueinstance;  
    }  
}
```

A static instance

Need to have a private constructor to block multiple instances

Lazy Loading

This ensures only one object instance is ever created.

However, this is good only for a single-threaded application

Singleton

```
public class Singleton {  
    private static Singleton uniqueinstance;  
  
    private Singleton() {}  
  
    public static synchronized Singleton getInstance() {  
  
        if (uniqueinstance == null) {  
            uniqueinstance = new Singleton();  
        }  
        return uniqueinstance;  
    }  
}
```

*This overcomes
the multiple
threading issue.*

However, the synchronization is bit expensive

This way is good if the performance is not an issue
By Udara Samaratunge

Singleton – with Double Check Lock

```
public class Singleton {  
    private volatile static Singleton uniqueinstance;  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        if (uniqueinstance == null) {  
            synchronized (Singleton.class) {  
                if (uniqueinstance == null) {  
                    uniqueinstance = new Singleton();  
                }  
            }  
        }  
        return uniqueinstance;  
    }  
}
```

Double
Check
Locking

Here the object is created and synchronized at the first time only. If the Double Check Locking is not there, two threads can get into the synchronized block one after the other

This way is good if the application is keen on its performance

Thread-safe singleton output

```
public class TestThreadSingleton implements Runnable{  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
  
        new Thread(new TestThreadSingleton()).start();  
  
        for (int i = 0; i < 10; i++) {  
            Singleton.getInstance();  
            ThreadSafeSingleton.getInstance();  
        }  
    }  
  
    /**  
     * Invoke thread  
     */  
    public void run(){  
        for (int i = 0; i < 10; i++) {  
            Singleton.getInstance();  
            ThreadSafeSingleton.getInstance();  
        }  
    }  
}
```

```
<terminated> TestThreadSingleton [Java Application]  
Singleton invocation  
Singleton invocation  
Object created for ThreadSafeSingleton.
```

Factory Pattern

The Factory Method

The factory method pattern encapsulates the object creation by letting subclasses to decide what objects to create

PizzaStore is now abstract (see why below).

```
public abstract class PizzaStore {  
  
    public Pizza orderPizza(String type) {  
        Pizza pizza;  
  
        pizza = createPizza(type);  
  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
  
        return pizza;  
    }  
  
    abstract createPizza(String type);  
}
```

Our "factory method" is now abstract in PizzaStore.

Now createPizza is back to being a call to a method in the PizzaStore rather than on a factory object.

All this looks just the same...

Now we've moved our factory object to this method.

The Creator Classes

```
public abstract class PizzaStore {  
    abstract Pizza createPizza(String item);  
  
    public Pizza orderPizza(String type) {  
        Pizza pizza = createPizza(type);  
        System.out.println("--- Making a " + pizza.getName() + " ---");  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
        return pizza;  
    }  
}  
  
public class ChicagoPizzaStore extends PizzaStore {  
  
    Pizza createPizza(String item) {  
        if (item.equals("cheese")) {  
            return new ChicagoStyleCheesePizza();  
        } else if (item.equals("veggie")) {  
            return new ChicagoStyleVeggiePizza();  
        } else if (item.equals("clam")) {  
            return new ChicagoStyleClamPizza();  
        } else if (item.equals("pepperoni")) {  
            return new ChicagoStylePepperoniPizza();  
        } else return null;  
    }  
}  
  
public class NYPizzaStore extends PizzaStore {  
  
    Pizza createPizza(String item) {  
        if (item.equals("cheese")) {  
            return new NYStyleCheesePizza();  
        } else if (item.equals("veggie")) {  
            return new NYStyleVeggiePizza();  
        } else if (item.equals("clam")) {  
            return new NYStyleClamPizza();  
        } else if (item.equals("pepperoni")) {  
            return new NYStylePepperoniPizza();  
        } else return null;  
    }  
}
```

Factory objects are created through INHERITANCE

This is the "Factory Method"

Simple Factory Vs Factory Method

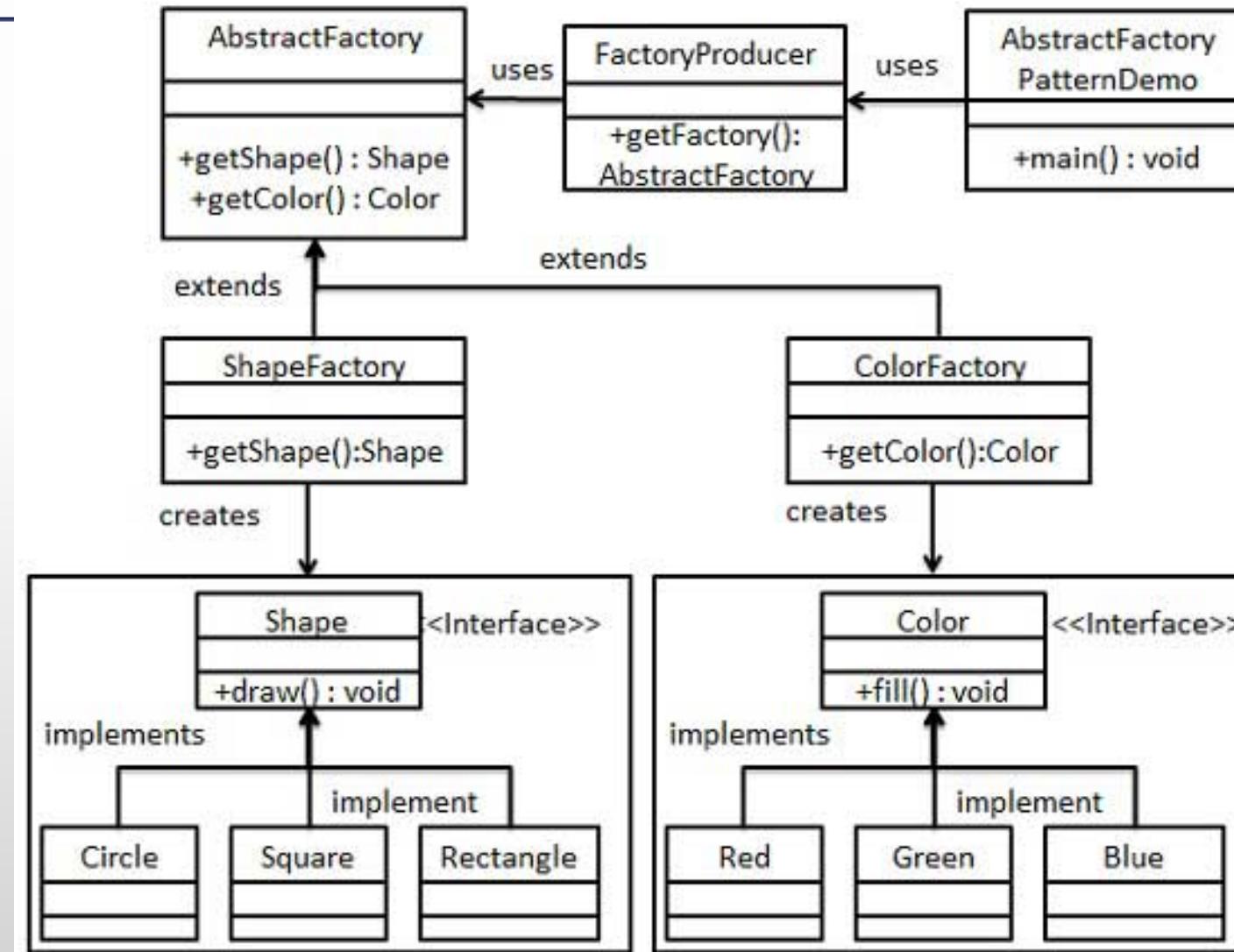
Simple Factory

-  Does not let you vary the product implementations being created

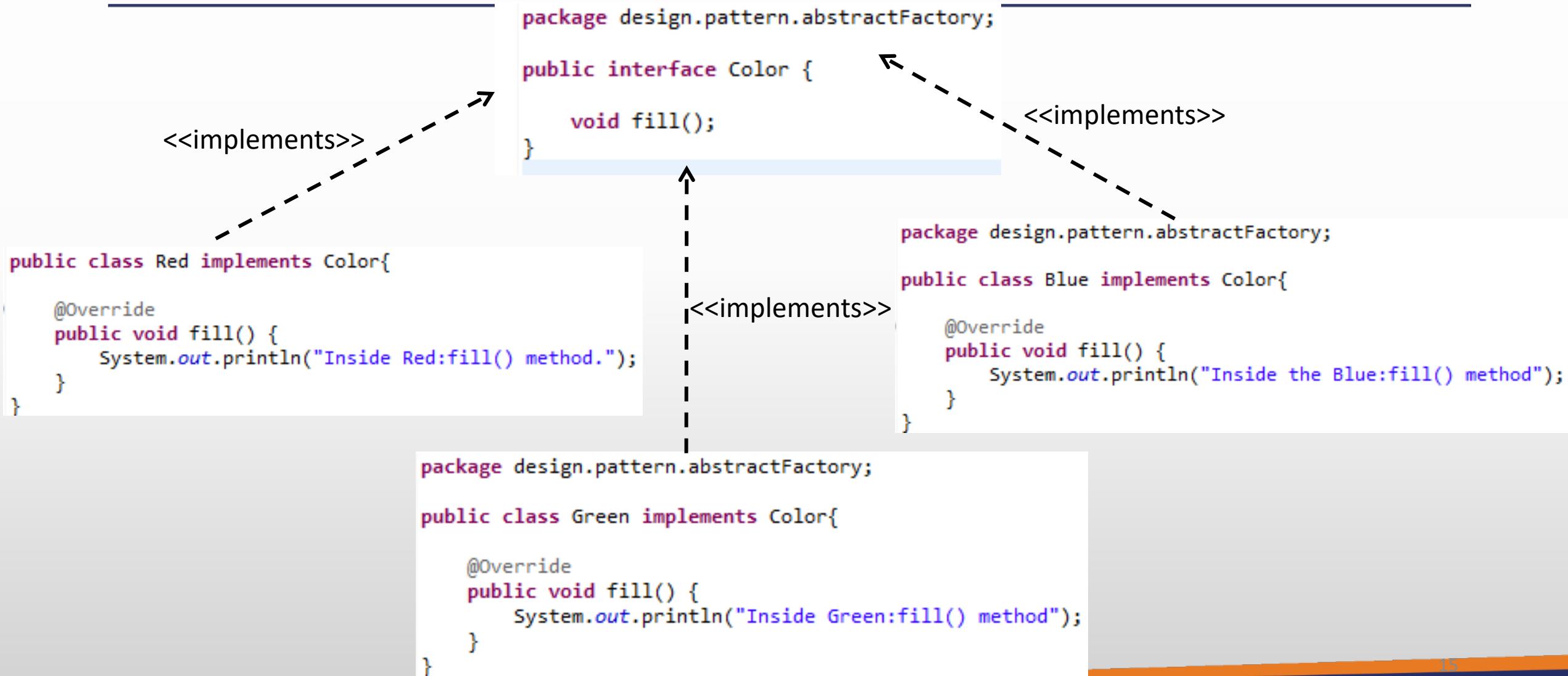
Factory Method

-  Creates a framework that lets the sub classes decides which product implementation will be used

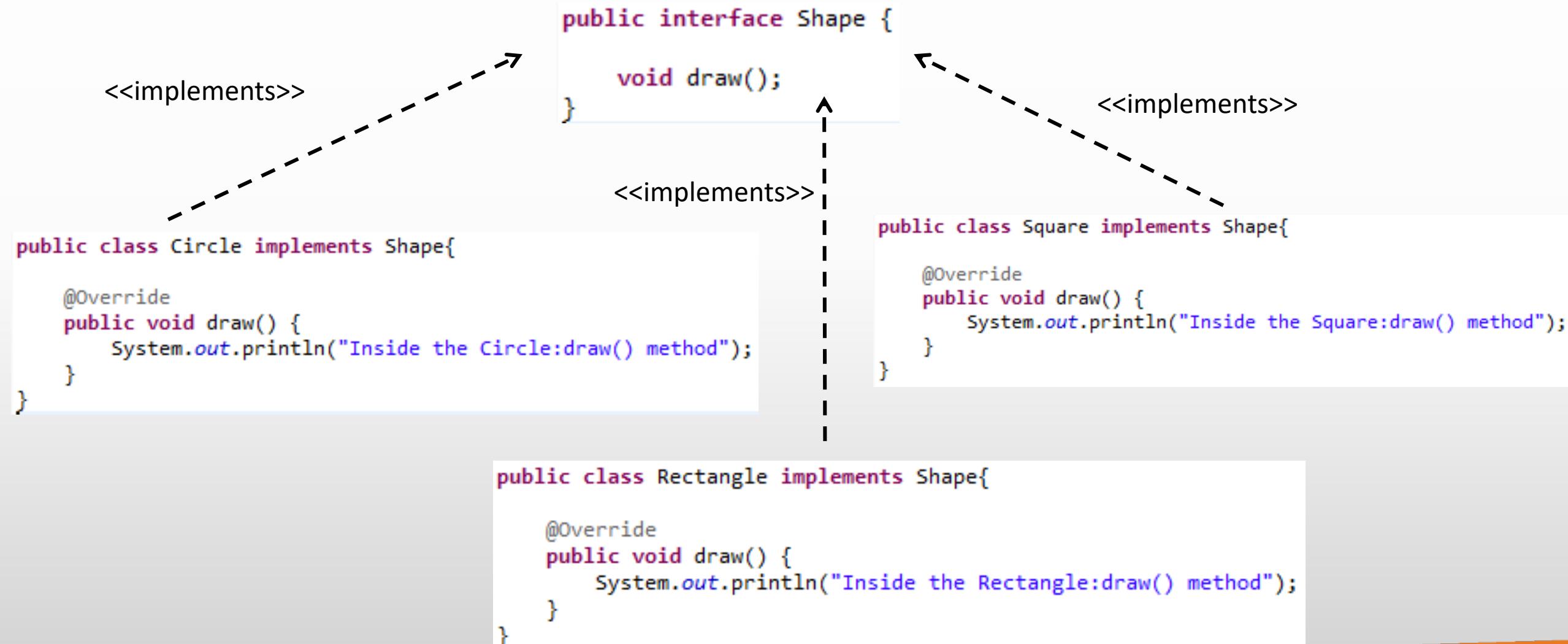
Abstract Factory Pattern



Abstract Factory Pattern



Abstract Factory Pattern



Abstract Factory Pattern

```
public class ColorFactory extends AbstractFactory{  
  
    @Override  
    public Color getColor(String color) {  
  
        if(color.equalsIgnoreCase("RED")){  
            return new Red();  
        }  
        else if(color.equalsIgnoreCase("GREEN")){  
            return new Green();  
        }  
        else if(color.equalsIgnoreCase("BLUE")){  
            return new Blue();  
        }  
        else{  
            return null;  
        }  
    }  
  
    @Override  
    public Shape getShape(String type) {  
        return null;  
    }  
}
```

```
public class FactoryProducer {  
  
    public static AbstractFactory getFactory(String choice){  
  
        if(choice.equalsIgnoreCase("SHAPE")){  
            return new ShapeFactory();  
        }  
        else if(choice.equalsIgnoreCase("COLOR")){  
            return new ColorFactory();  
        }  
        else{  
            return null;  
        }  
    }  
}
```

```
public class ShapeFactory extends AbstractFactory{  
  
    @Override  
    public Shape getShape(String shapeType) {  
  
        if(shapeType == null){  
            return null;  
        }  
        else if(shapeType.equalsIgnoreCase("CIRCLE")){  
            return new Circle();  
        }  
        else if(shapeType.equalsIgnoreCase("RECTANGLE")){  
            return new Rectangle();  
        }  
        else if(shapeType.equalsIgnoreCase("SQUARE")){  
            return new Square();  
        }  
        else{  
            return null;  
        }  
    }  
  
    @Override  
    public Color getColor(String type) {  
        return null;  
    }  
}
```

Abstract Factory Pattern

```
public class ColorFactory extends AbstractFactory{
    @Override
    public Color getColor(String color) {
        if(color.equalsIgnoreCase("RED")){
            return new Red();
        }
        else if(color.equalsIgnoreCase("GREEN")){
            return new Green();
        }
        else if(color.equalsIgnoreCase("BLUE")){
            return new Blue();
        }
        else{
            return null;
        }
    }

    @Override
    public Shape getShape(String type) {
        return null;
    }
}
```

uses

```
public abstract class AbstractFactory {
    public abstract Color getColor(String type);
    public abstract Shape getShape(String type);
}

public class ShapeFactory extends AbstractFactory{
    @Override
    public Shape getShape(String shapeType) {
        if(shapeType == null){
            return null;
        }
        else if(shapeType.equalsIgnoreCase("CIRCLE")){
            return new Circle();
        }
        else if(shapeType.equalsIgnoreCase("RECTANGLE")){
            return new Rectangle();
        }
        else if(shapeType.equalsIgnoreCase("SQUARE")){
            return new Square();
        }
        else{
            return null;
        }
    }

    @Override
    public Color getColor(String type) {
        return null;
    }
}

class FactoryProducer {
    public static AbstractFactory getFactory(String choice){
        if(choice.equalsIgnoreCase("SHAPE")){
            return new ShapeFactory();
        }
        else if(choice.equalsIgnoreCase("COLOR")){
            return new ColorFactory();
        }
        else{
            return null;
        }
    }
}
```

Abstract Factory Pattern

```
package design.pattern.abstractFactory;

public class AbstractFactoryPatternDemo {

    private static final String SHAPE = "SHAPE";
    private static final String CIRCLE = "CIRCLE";
    private static final String RECTANGLE = "RECTANGLE";
    private static final String SQUARE = "SQUARE";

    private static final String COLOR = "COLOR";
    private static final String RED = "RED";
    private static final String GREEN = "GREEN";
    private static final String BLUE = "BLUE";

    public static void main(String[] args) {

        AbstractFactory shapeFactory = FactoryProducer.getFactory(SHAPE);
        Shape shape = shapeFactory.getShape(CIRCLE);
        shape.draw();

        FactoryProducer.getFactory(SHAPE).getShape(RECTANGLE).draw();
        FactoryProducer.getFactory(SHAPE).getShape(SQUARE).draw();

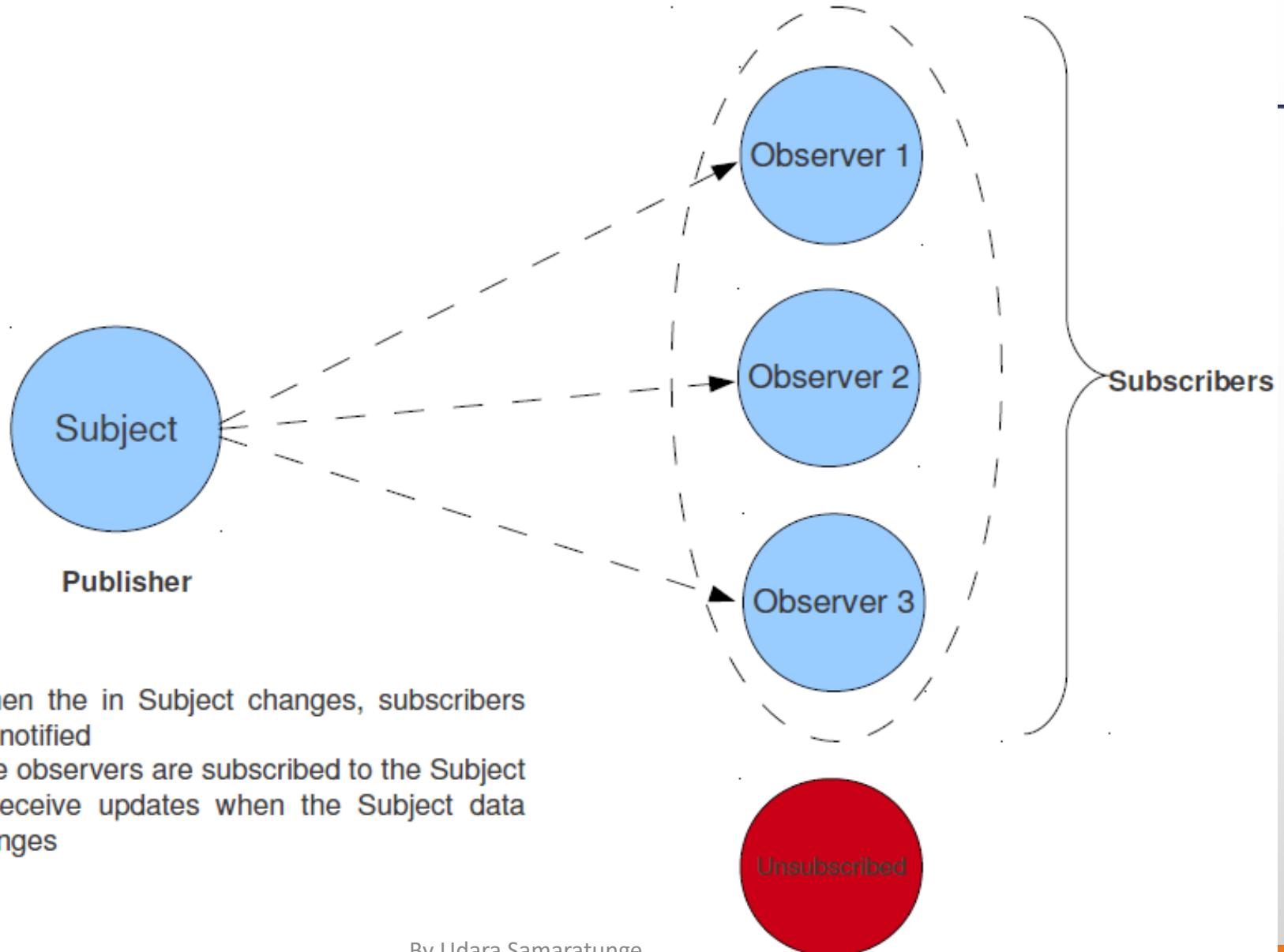
        FactoryProducer.getFactory(COLOR).getColor(RED).fill();
        FactoryProducer.getFactory(COLOR).getColor(GREEN).fill();
        FactoryProducer.getFactory(COLOR).getColor(BLUE).fill();
    }
}
```

Problems Console @ Javadoc Declaration Search Progress Cross Ref

```
<terminated> AbstractFactoryPatternDemo [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\ja
Inside the Circle:draw() method
Inside the Rectangle:draw() method
Inside the Square:draw() method
Inside Red:fill() method.
Inside Green:fill() method
Inside the Blue:fill() method
```

Observer Pattern

The Observer Pattern



Design Principles covered - (1)

④ *Design Principle 4*

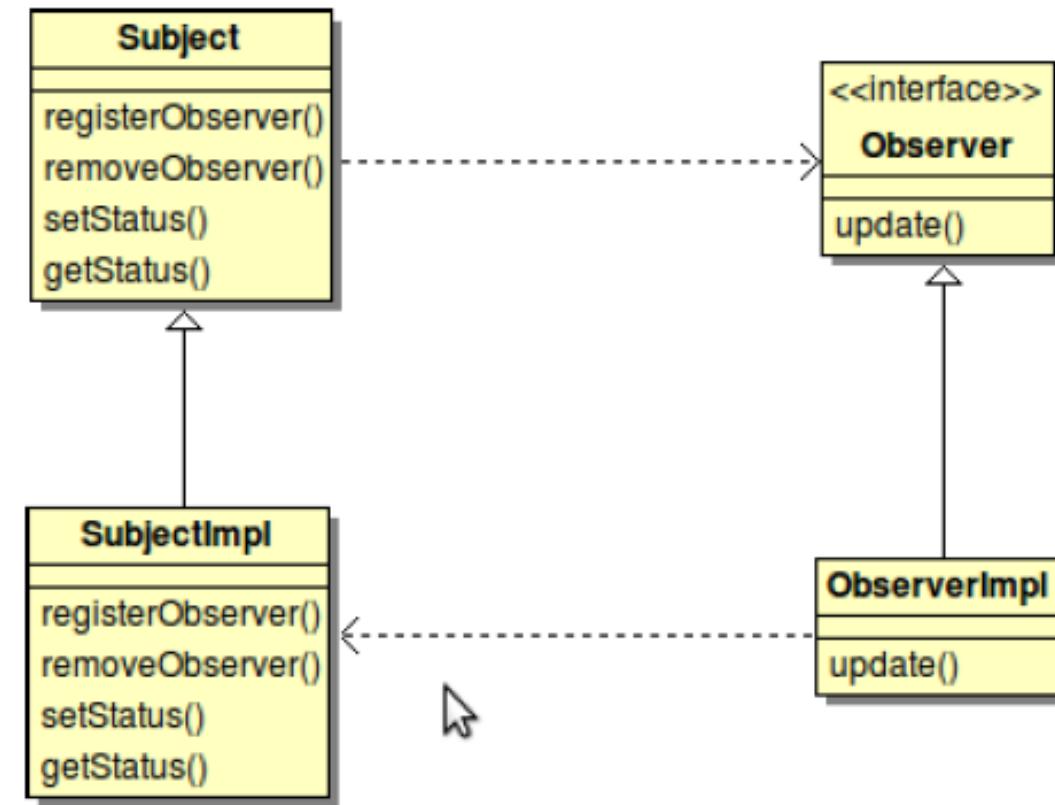
Strive for loosely coupled designs between objects that interact

The Observer Pattern

- ➊ The subject and the observers are having a *one to many relationship*
- ➋ The observers are *dependent* on the subject
- ➌ When the subject state changes, the observers get notified
- ➍ In the observer pattern, (About the state)
 - ➎ Subject is the object that contains the state and it owns it
 - ➏ Observers use it but do not own it

The Observer Pattern - Explained

- ➊ **Subject:** Maintains a list of Observer references. Subject also provides an interface for attaching and detaching Observer objects.
- ➋ **Observer:** Defines an updating interface for objects that should be notified of changes in a subject.
- ➌ **ConcreteSubject:** Stores state of interest to ConcreteObserver objects and sends notifications to its observers upon state changes.
- ➍ **ConcreteObserver:** Maintains a reference to a ConcreteSubject object and a state that should stay consistent with the subject's.



```
public interface Subject {  
    public void registerObserver(Observer o);  
    public void removeObserver(Observer o);  
    public void setState(String state);  
    public String getState();  
}
```

```
public interface Observer {  
    public void update(Subject subject);  
}
```

```
/**  
 * @author udara.s  
 */  
  
public class SubjectImpl implements Subject{  
  
    private String state;  
    List<Observer> observerList = new ArrayList<Observer>();  
  
    @Override  
    public void registerObserver(Observer observer) {  
        observerList.add(observer);  
    }  
  
    @Override  
    public void removeObserver(Observer observer) {  
        observerList.remove(observer);  
    }  
  
    @Override  
    public void setStatus(String status) {  
        this.state = status;  
        notifyObservers();  
    }  
  
    @Override  
    public String getStatus() {  
        return this.state;  
    }  
  
    /**  
     * Notify for all observers  
     */  
    public void notifyObservers(){  
        Iterator<Observer> iterator = observerList.iterator();  
        while (iterator.hasNext()) {  
            Observer observer = (Observer)iterator.next();  
            observer.update(this);  
        }  
    }  
}
```

Observer Pattern

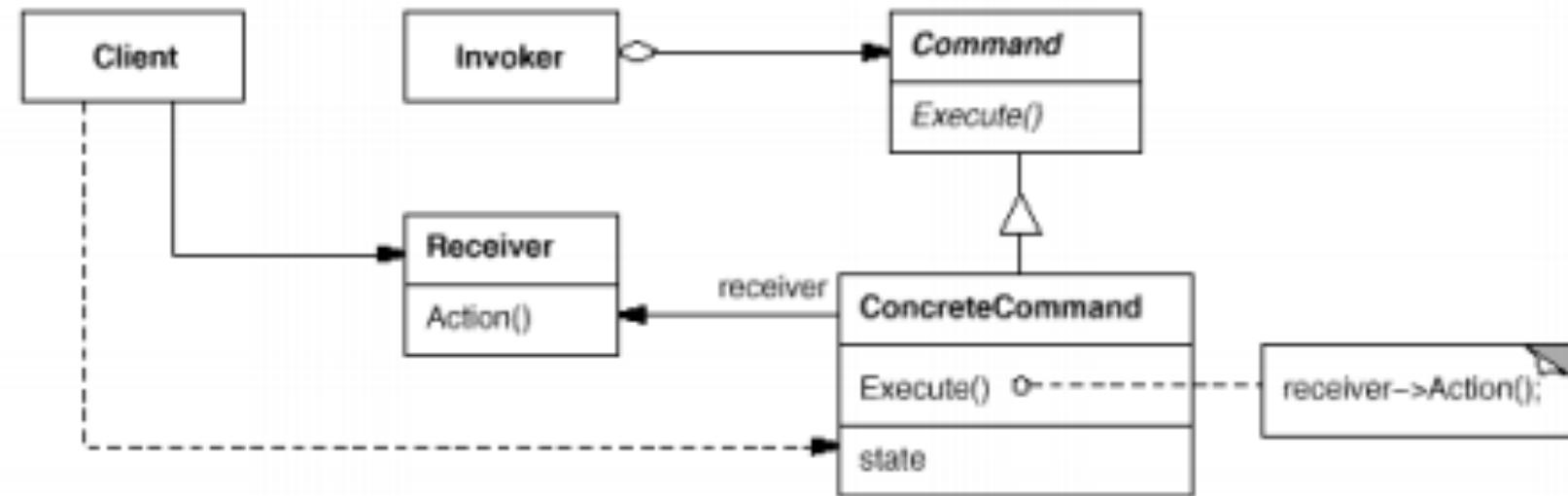
```
public class ObserverImpl implements Observer {  
  
    private String id;  
    private String state;  
  
    public ObserverImpl(String id) {  
        this.id = id;  
    }  
  
    @Override  
    public void update(Subject subject) {  
  
        this.state = subject.getStatus();  
        System.out  
            .println("Observer received state change of subject ID is = "  
                + this.id + " Status = " + this.state);  
    }  
}
```

Observer Pattern Output

```
public class TestObserver {  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
  
        Observer observer1 = new ObserverImpl("Observer 1");  
        Observer observer2 = new ObserverImpl("Observer 2");  
        Observer observer3 = new ObserverImpl("Observer 3");  
        Observer observer4 = new ObserverImpl("Observer 4");  
        Observer observer5 = new ObserverImpl("Observer 5");  
  
        Subject subject = new SubjectImpl();  
  
        subject.registerObserver(observer1);  
        subject.registerObserver(observer2);  
        subject.registerObserver(observer3);  
        subject.registerObserver(observer4);  
        subject.registerObserver(observer5);  
  
        subject.setStatus("status modified");  
    }  
}
```

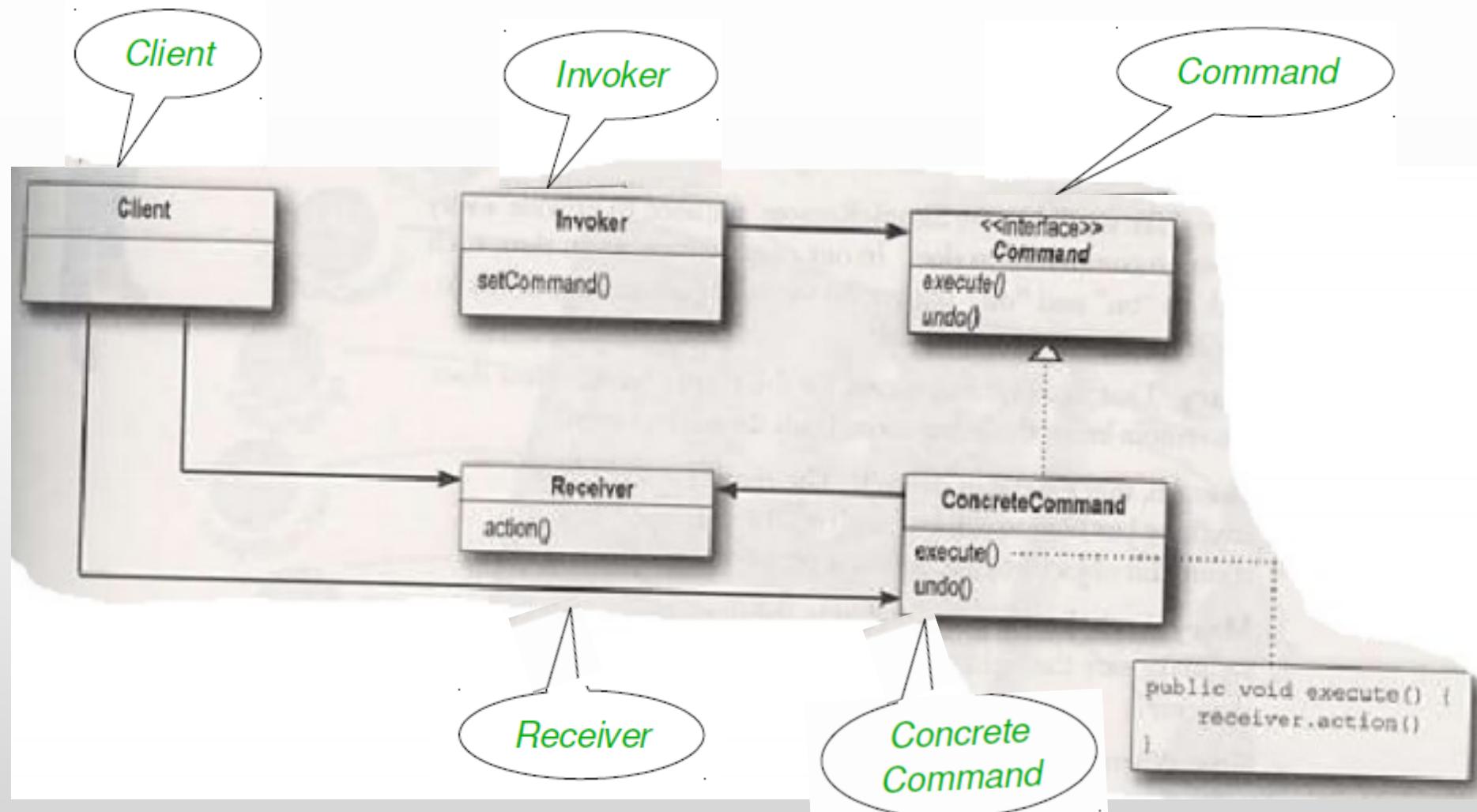
```
<terminated> TestObserver [Java Application] C:\Program Files\Java\jdk1.7.0_71\bin\javaw.exe (Oct 23, 2017 4:36:  
Observer received state change of subject ID is = Observer 1 Status = status modified  
Observer received state change of subject ID is = Observer 2 Status = status modified  
Observer received state change of subject ID is = Observer 3 Status = status modified  
Observer received state change of subject ID is = Observer 4 Status = status modified  
Observer received state change of subject ID is = Observer 5 Status = status modified
```

Command Pattern

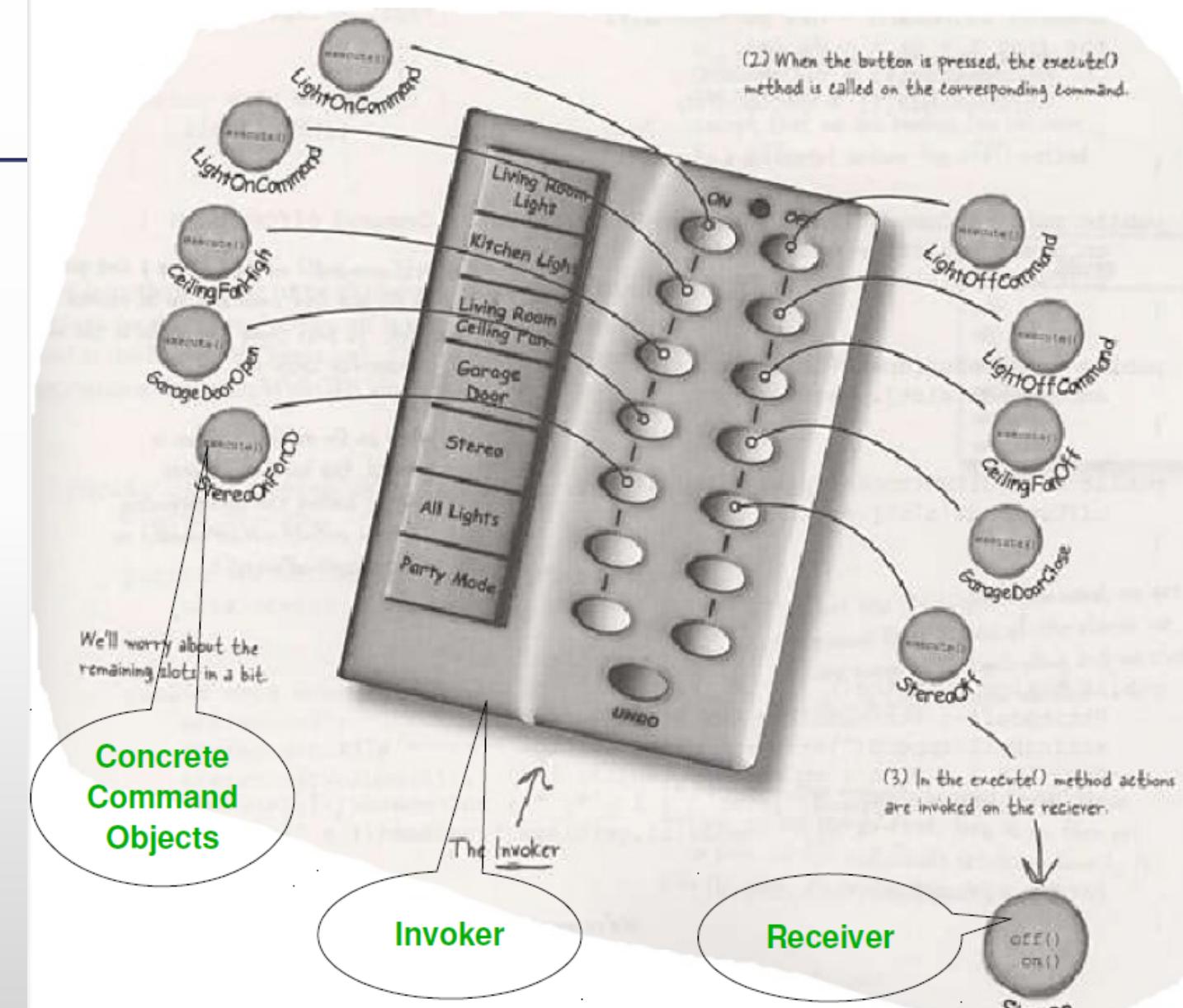


Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations

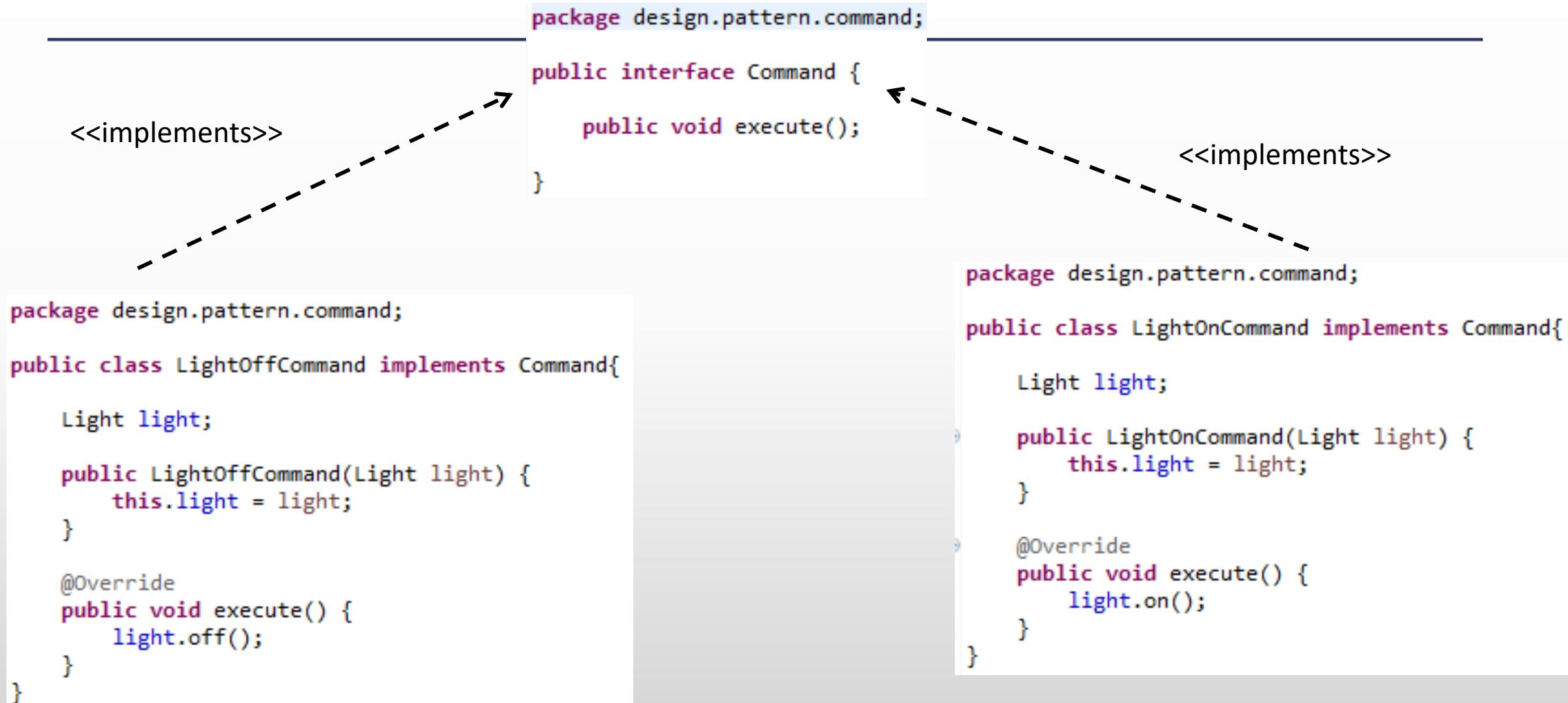
Command Pattern



Command Pattern



Command & Concrete Command



Receiver

```
public class Light {  
  
    private String location;  
  
    public Light(String location) {  
        this.location = location;  
    }  
  
    public void on(){  
        System.out.println(location + " light is on.");  
    }  
  
    public void off(){  
        System.out.println(location + " light is off.");  
    }  
}
```

Command Pattern

```

package design.pattern.command;

public class RemoteController {
    Command [] onCommands;
    Command [] offCommands;

    public RemoteController() {
        onCommands = new Command[7];
        offCommands = new Command[7];

        for (int i = 0; i < 7; i++) {
            onCommands[i] = null;
            offCommands[i] = null;
        }
    }

    public void setCommand(int slot, Command onCommand, Command offCommand){
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
    }

    public void onButtonWasPushed(int slot){
        onCommands[slot].execute();
    }

    public void offButtonWasPushed(int slot){
        offCommands[slot].execute();
    }
}

```

```

package design.pattern.command;

public interface Command {
    public void execute();
}

public class Light {
    private String location;

    public Light(String location) {
        this.location = location;
    }

    public void on(){
        System.out.println(location + " light is on.");
    }

    public void off(){
        System.out.println(location + " light is off.");
    }
}

```

Invoker



The screenshot shows the Eclipse IDE interface with several tabs at the top: sample1.java, Client.java, RemoteContr..., LightOnComm..., Command, Problems, Console, and JavaDoc. The Client.java tab is active, displaying the following Java code:

```
package design.pattern.command;

public class Client {
    /**
     * @param args
     */
    public static void main(String[] args) {
        RemoteController remoteController = new RemoteController();

        Light livingRoomLight = new Light("Living Room Light");
        Light kitchenLight = new Light("Kitchen Light");

        LightOnCommand onLivingRoomLight = new LightOnCommand(livingRoomLight);
        LightOffCommand offLivingRoomLight = new LightOffCommand(livingRoomLight);
        LightOnCommand onKitchenLight = new LightOnCommand(kitchenLight);
        LightOffCommand offKitchenLight = new LightOffCommand(kitchenLight);

        remoteController.setCommand(0, onLivingRoomLight, offLivingRoomLight);
        remoteController.setCommand(1, onKitchenLight, offKitchenLight);

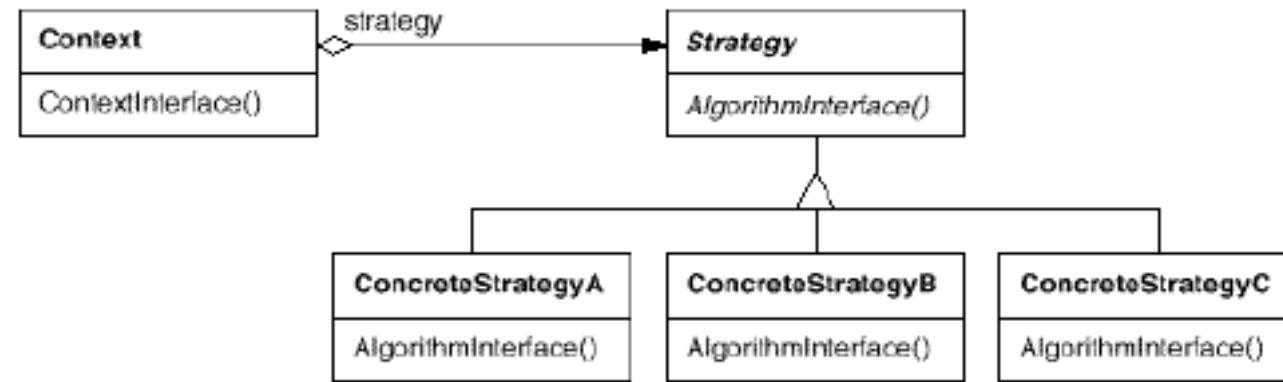
        remoteController.onButtonWasPushed(0);
        remoteController.offButtonWasPushed(0);
        remoteController.onButtonWasPushed(1);
        remoteController.offButtonWasPushed(1);
    }
}
```

The Problems tab shows the output of the application's execution:

```
<terminated> Client [Java Application] C:\Java\workspace\sample1\bin
Living Room Light light is on.
Living Room Light light is off.
Kitchen Light light is on.
Kitchen Light light is off.
```

Strategy Pattern

Strategy Pattern



Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

By Udara Samaratunge

Design Principles covered - (3)

④ *Design Principle 1*

Identify the aspects of your application that vary and separate them from what stays the same

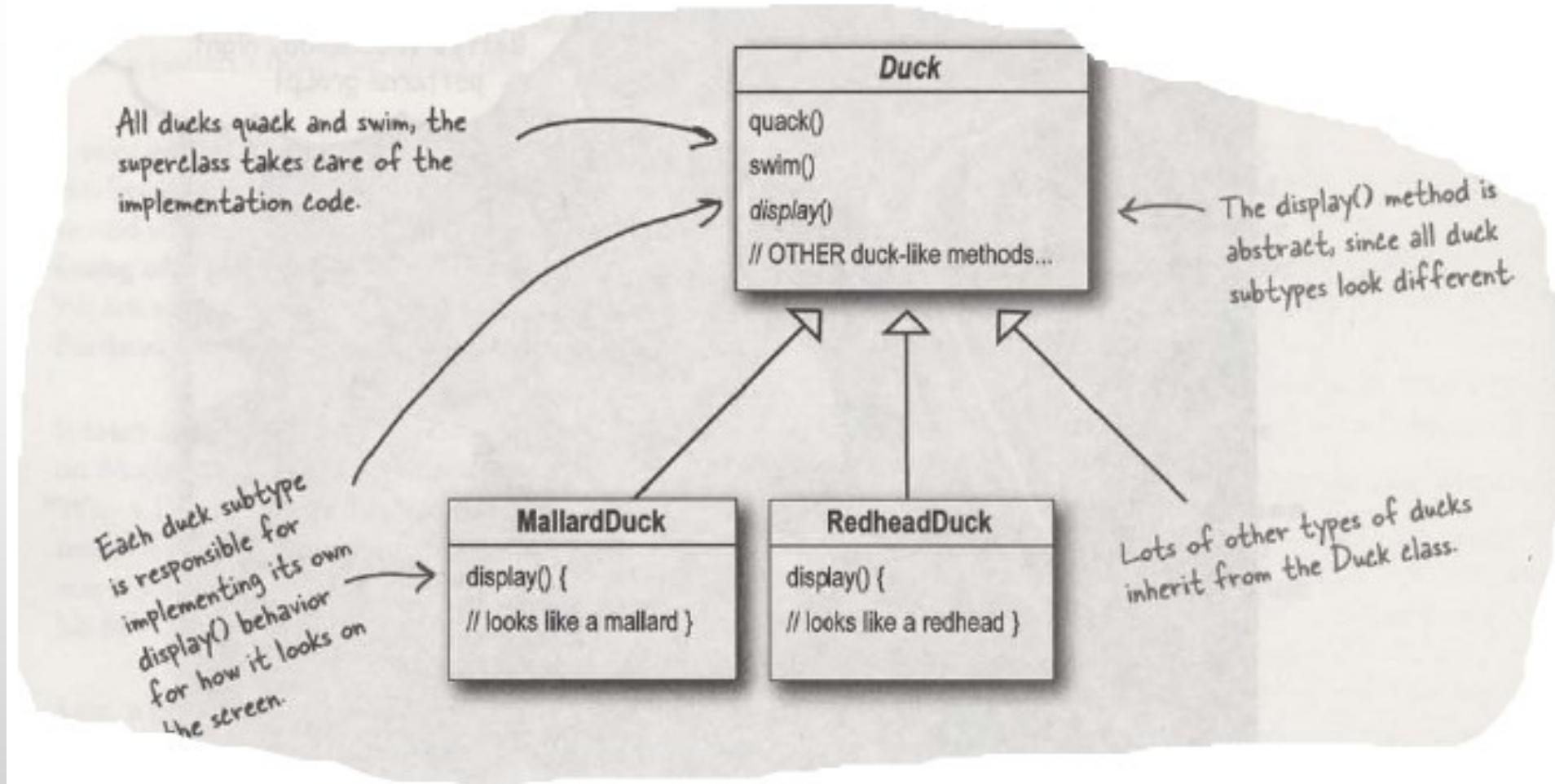
④ *Design Principle 2*

Program to an interface not to an implementation

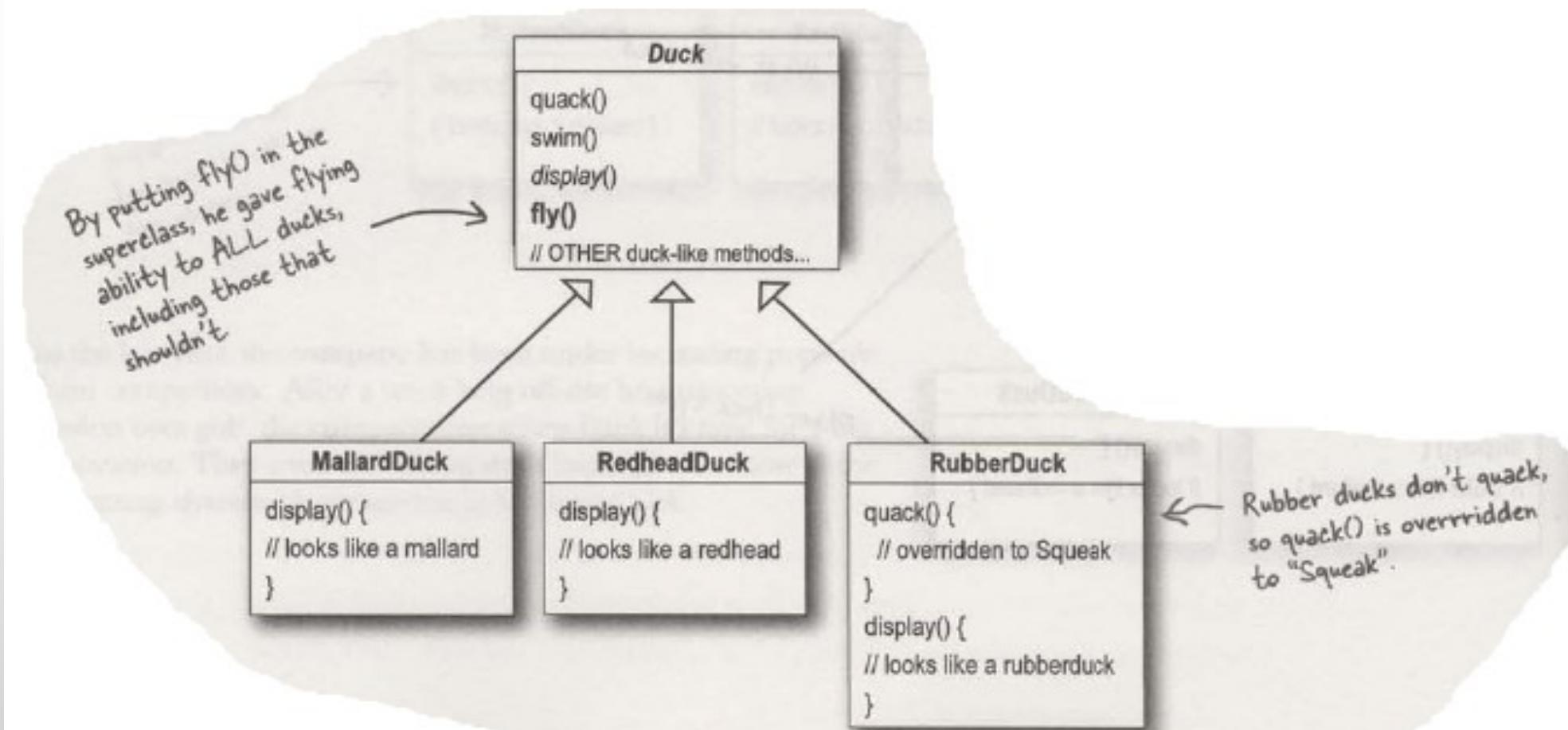
④ *Design Principle 3*

Favor composition over inheritance

The Duck Simulation

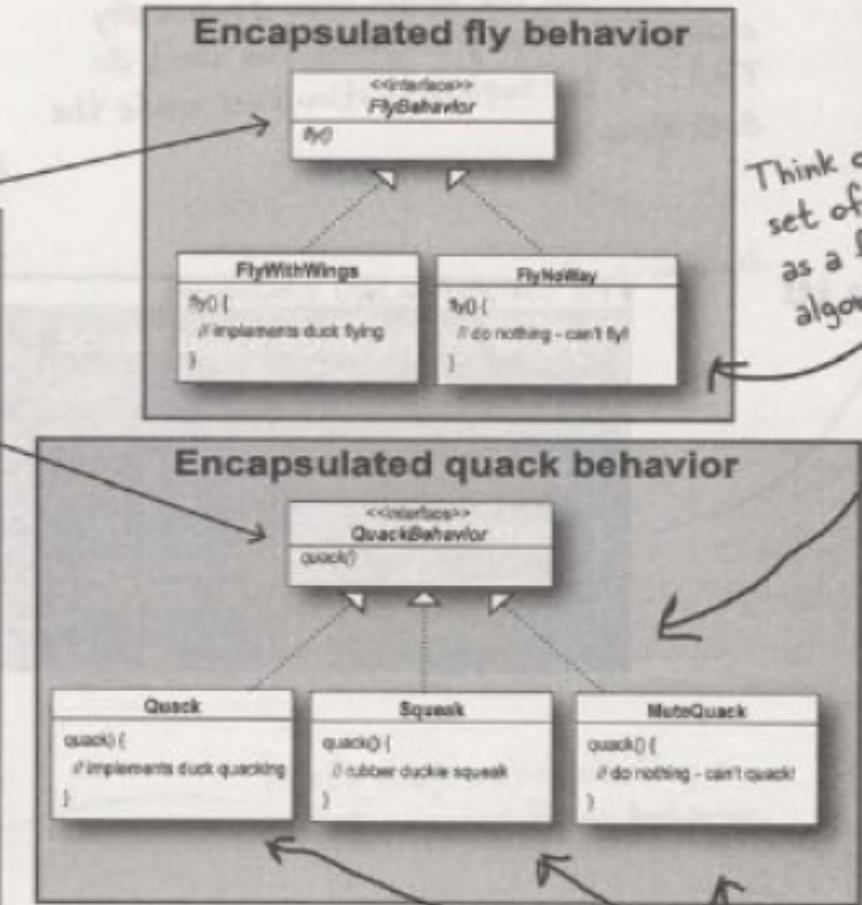
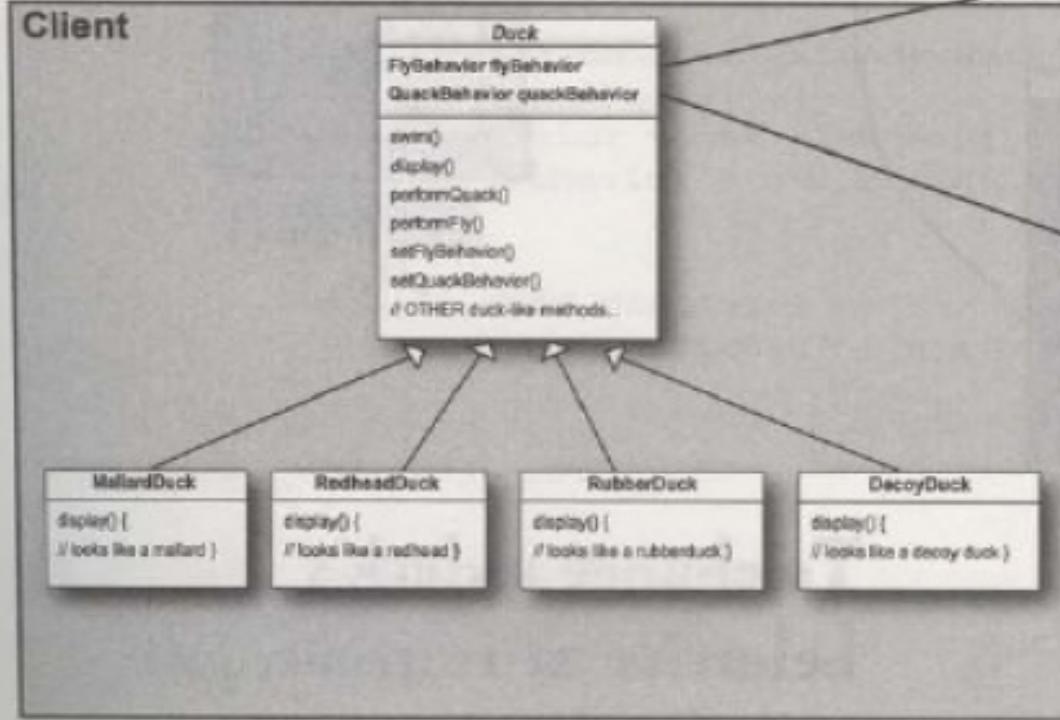


How about Inheritance?



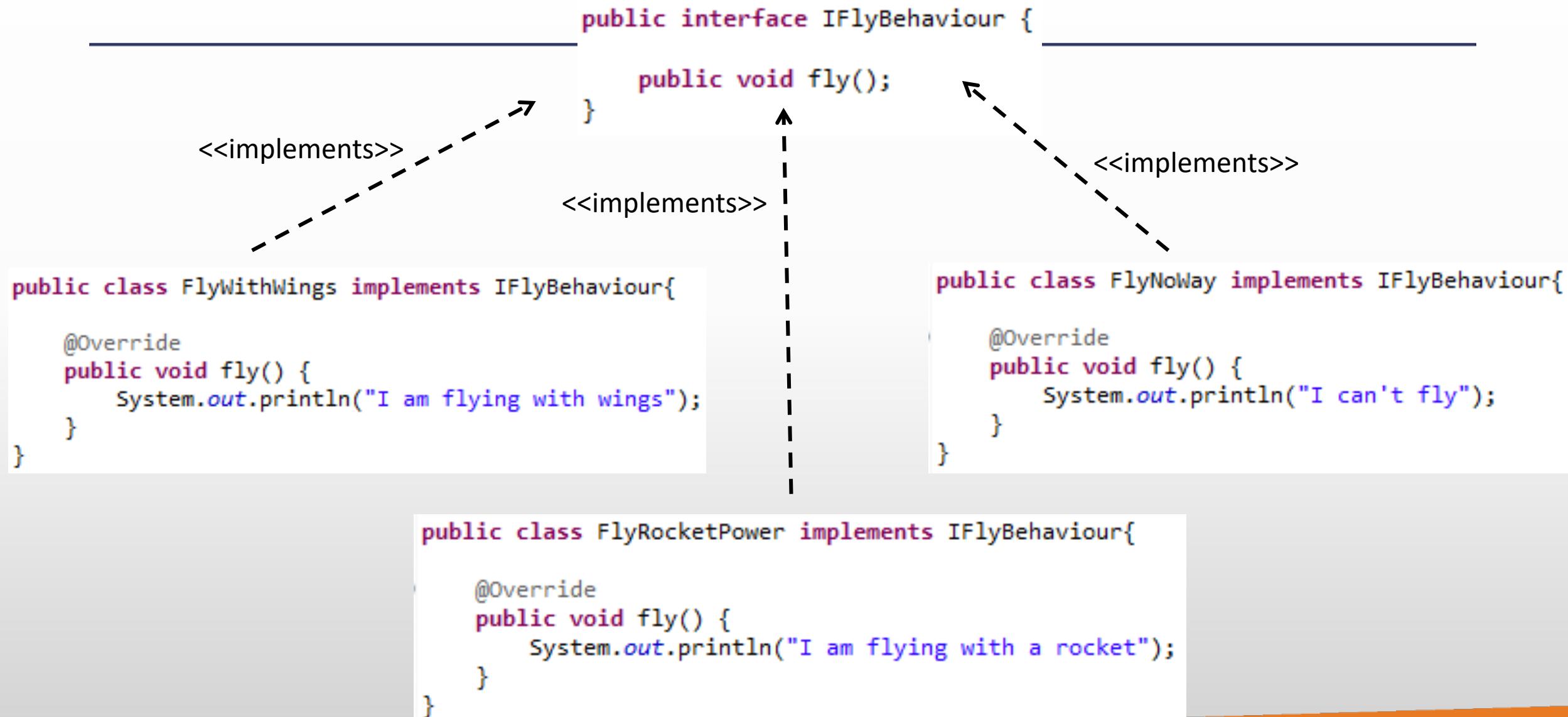
Setting the behavior dynamically

Client makes use of an encapsulated family of algorithms for both flying and quacking.



These behavioral "algorithms" are interchangeable.

Strategy Pattern Implementation



Strategy Pattern Implementation

```
public interface IQuackBehaviour {  
    public void quack();  
}
```

<<implements>>

```
public class Quack implements IQuackBehaviour{  
    @Override  
    public void quack() {  
        System.out.println("Quack..Quack...");  
    }  
}
```

<<implements>>

```
public class ModelQuack implements IQuackBehaviour{  
    @Override  
    public void quack() {  
        System.out.println("Quack Model duck");  
    }  
}
```

```
public abstract class Duck {
    IFlyBehaviour flyBehaviour;
    IQuackBehaviour quackBehaviour;

    public abstract void display();

    public void performFly(){
        flyBehaviour.fly();
    }

    public void performQuack(){
        quackBehaviour.quack();
    }

    public void swim(){
        System.out.println("All ducks float even Decoy");
    }

    public void setFlyBehaviour(IFlyBehaviour flyBehaviour) {
        this.flyBehaviour = flyBehaviour;
    }

    public void setQuackBehaviour(IQuackBehaviour quackBehaviour) {
        this.quackBehaviour = quackBehaviour;
    }
}
```

```
public interface IFlyBehaviour {
    public void fly();
}

public interface IQuackBehaviour {
    public void quack();
}
```

```
public class ModelDuck extends Duck{
    public ModelDuck() {
        quackBehaviour = new Quack();
        flyBehaviour = new FlyNoWay();
    }

    @Override
    public void display() {
        System.out.println("I am a model Duck");
    }
}
```

```
public class MollardDuck extends Duck{
    public MollardDuck() {
        quackBehaviour = new Quack();
        flyBehaviour = new FlyWithWings();
    }

    @Override
    public void display() {
        System.out.println("I am a real Mollard Duck.");
    }
}
```

Strategy Pattern Implementation

```
package design.pattern.strategy;

public class TestDuck {

    /**
     * @param args
     */
    public static void main(String[] args) {

        System.out.println("Start Mollard Duck");
        System.out.println("=====");
        Duck mollard = new MollardDuck();
        mollard.performFly();
        mollard.performQuack();

        System.out.println("Start Model Duck");
        System.out.println("=====");
        Duck model = new ModelDuck();

        model.performFly();
        model.setFlyBehaviour(new FlyRocketPower());
        model.performFly();

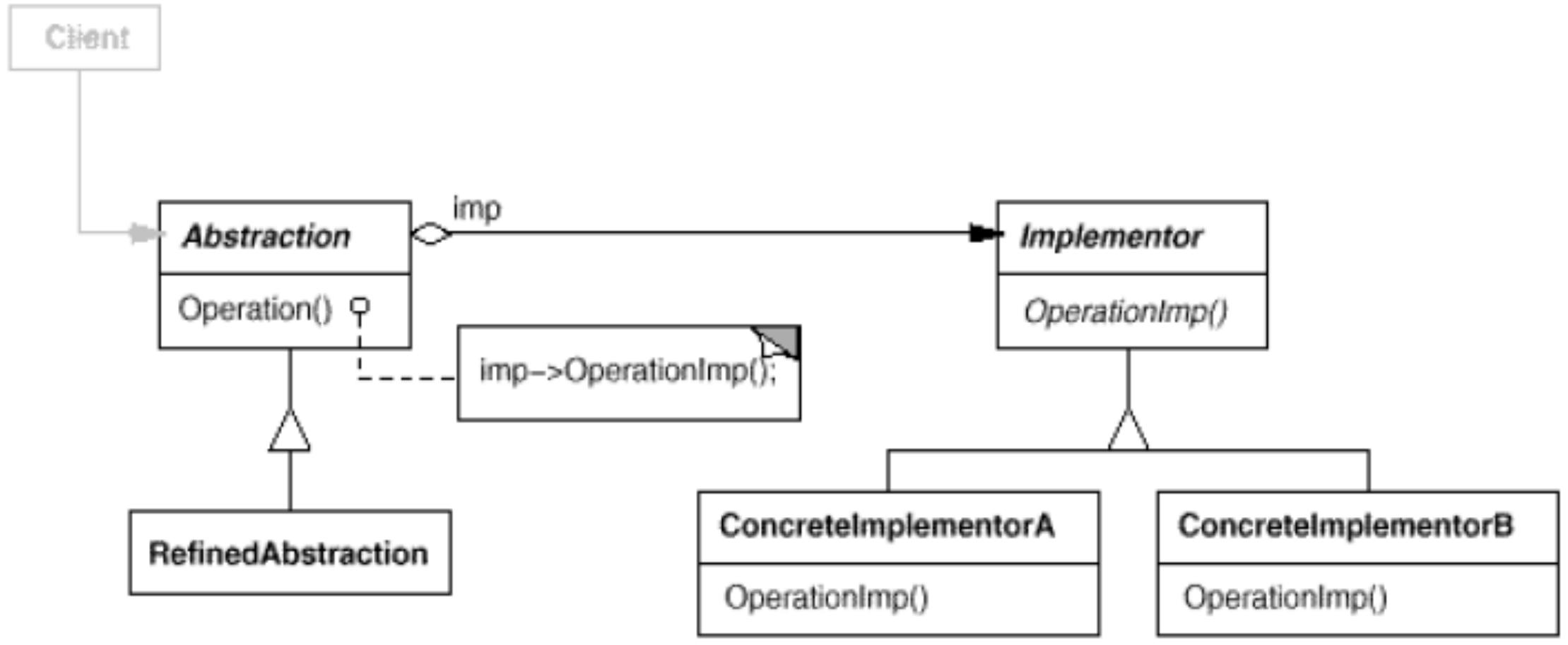
        model.performQuack();
        model.setQuackBehaviour(new ModelQuack());
        model.performQuack();

    }
}
```

```
<terminated> TestDuck [Java Application] C:\P
Start Mollard Duck
=====
I am flying with wings
Quack..Quack...
Start Model Duck
=====
I can't fly
I am flying with a rocket
Quack..Quack...
Quack Model duck
```

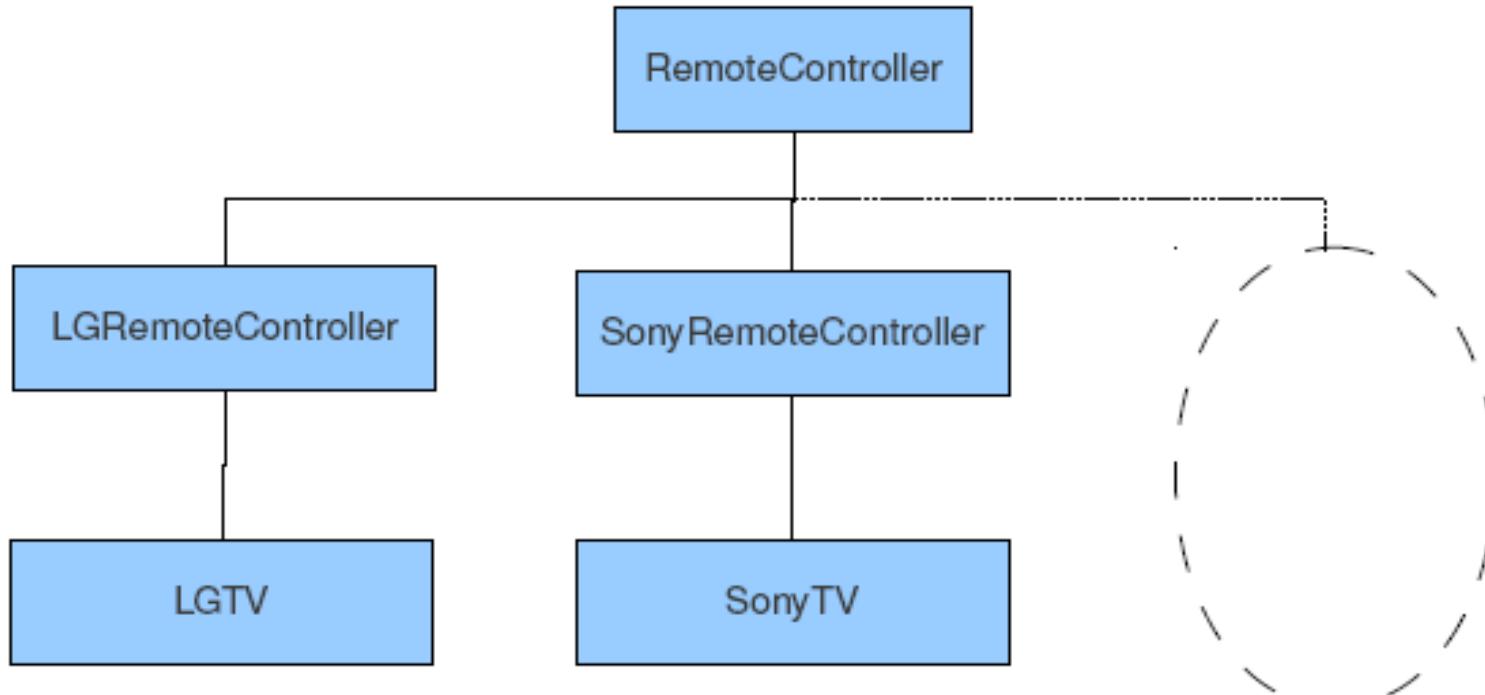
Bridge Pattern

Bridge Pattern



Example for Bridge Pattern

- There are two brands of TVs (Sony and LG) in your living room. So there are two remote controllers for each one. (See below diagram) Just assume a single remote controller can be used to **switch on, switch off** and **tune channels** of both TVs. Think of a design pattern that can solve this.



Bridge Pattern

```
public interface TV {  
    void on();  
    void off();  
    void tune(int channel);  
}  
  
public class LGTV implements TV{  
    @Override  
    public void on() {  
        System.out.println("Switch on LG TV");  
    }  
  
    @Override  
    public void off() {  
        System.out.println("Switch off LG TV");  
    }  
  
    @Override  
    public void tune(int channel) {  
        System.out.println("Switch on channel in LG TV is: " + channel);  
    }  
}
```

Diagram illustrating the Bridge Pattern:

```
graph TD; TV[TV] -->|<<implements>>| LGTV[LGTV]; LGTV -->|<<implements>>| SonyTV[SonyTV]
```

The diagram shows the TV interface at the top, with a dashed arrow pointing down to the LGTV class, labeled with '<<implements>>'. Another dashed arrow points up from the SonyTV class to the TV interface, also labeled with '<<implements>>'.

```
public class SonyTV implements TV{  
    @Override  
    public void on() {  
        System.out.println("Switch on Sony TV");  
    }  
  
    @Override  
    public void off() {  
        System.out.println("Switch off Sony TV");  
    }  
  
    @Override  
    public void tune(int channel) {  
        System.out.println("Switch on channel in Sony TV is: " + channel);  
    }  
}
```

```

public interface RemoteController {
    void on();
    void off();
    void tune(int channel);
}

public class RemoteControllerImpl implements RemoteController{
    TV tv;

    public RemoteControllerImpl(TV tv) {
        this.tv = tv;
    }

    @Override
    public void on() {
        tv.on();
    }

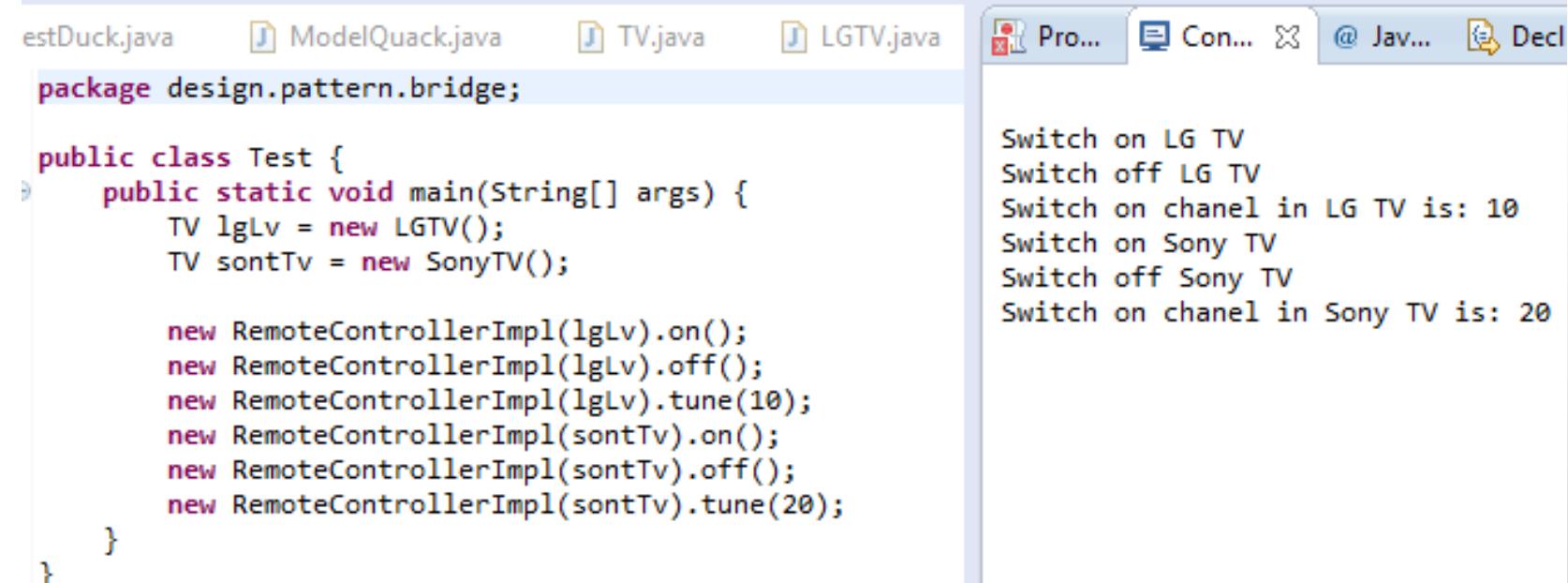
    @Override
    public void off() {
        tv.off();
    }

    @Override
    public void tune(int channel) {
        tv.tune(channel);
    }
}

```

↑
| <<implements>>
|

Bridge Pattern



```

package design.pattern.bridge;

public class Test {
    public static void main(String[] args) {
        TV lgLv = new LGTV();
        TV sonyTv = new SonyTV();

        new RemoteControllerImpl(lgLv).on();
        new RemoteControllerImpl(lgLv).off();
        new RemoteControllerImpl(lgLv).tune(10);
        new RemoteControllerImpl(sonyTv).on();
        new RemoteControllerImpl(sonyTv).off();
        new RemoteControllerImpl(sonyTv).tune(20);
    }
}

```

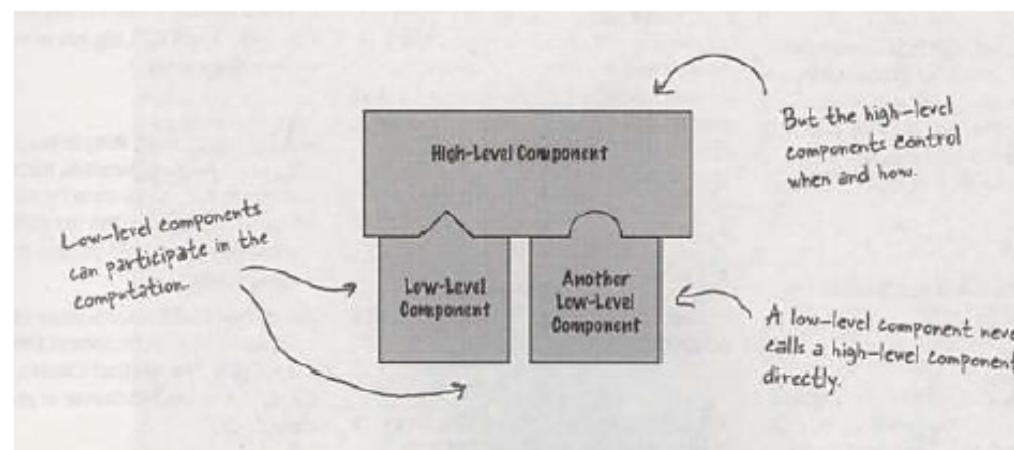
Template method pattern

Design Principles covered - (1)

④ *Design Principle*

“The Hollywood Principle” - *Don’t call us, we will call you*

Allows low level components to hook themselves into a system. But the high-level components determine when they are needed and how.



④ The *Template Method*,

Is a method, which serves as a **template** for an algorithm

- ④ In the template,
 - ④ Each step of the algorithm is represented by a method
(These are called as “hooks”)
 - ④ Some methods are handled by this class.
 - ④ Some methods are handled by the sub class.
 - ④ The methods, that need to be supplied by a subclass
are declared *abstract*

Template method pattern

The template method defines the steps of an algorithm and follows subclasses to provide the implementation for one more steps

Template method pattern

An Example: Servlets

- ⌚ The servlet container invokes our servlet code
- ⌚ HttpServlet defines a *Template Method service()*, which takes care of general purpose handling of HTTP requests by calling **doGet()** and **doPost()** methods
- ⌚ We can extend the HttpServlet by overriding the steps of the algorithm, **doGet()** and **doPost()** methods to provide meaningful results

An Example: Servlets

Servlet Containers Hollywood Principle

Don't call me I will call you (servlet), whenever I hear from a browser

Servlet's Template Method

*Let me have the control of the algorithm and let me deal with HTTP.
You (Developer) just respond with some meaningful action when I
call your methods*

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    String method = req.getMethod();

    if (method.equals(METHOD_GET)) {
        long lastModified = getLastModified(req);
        if (lastModified == -1) {
            // servlet doesn't support if-modified-since, no reason
            // to go through further expensive logic
            doGet(req, resp);
        } else {
            long ifModifiedSince = req.getDateHeader(HEADER_IFMODSINCE);
            if (ifModifiedSince < (lastModified / 1000 * 1000)) {
                // If the servlet mod time is later, call doGet()
                // Round down to the nearest second for a proper compare
                // A ifModifiedSince of -1 will always be less
                maybeSetLastModified(resp, lastModified);
                doGet(req, resp);
            } else {
                resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
            }
        }
    } else if (method.equals(METHOD_HEAD)) {
        long lastModified = getLastModified(req);
        maybeSetLastModified(resp, lastModified);
        doHead(req, resp);

    } else if (method.equals(METHOD_POST)) {
        doPost(req, resp);

    } else if (method.equals(METHOD_PUT)) {
        doPut(req, resp);
```

Template
Method

```
public abstract class CaffeineBeverage {
```

```
    void final prepareRecipe() {
```

```
        boilWater();
```

```
        brew();
```

```
        pourInCup();
```

```
        addCondiments();
```

```
}
```

```
abstract void brew();
```

```
abstract void addCondiments();
```

```
void boilWater() {  
    // implementation  
}
```

```
void pourInCup() {  
    // implementation  
}
```

```
}
```

prepareRecipe() is our template method
Why?

Because:

(1) It is a method, after all.

(2) It serves as a template for an algorithm, in this case, an algorithm for making caffeinated beverages.

In the template, each step of the algorithm is represented by a method.

Some methods are handled by this class...

...and some are handled by the subclass.

The methods that need to be supplied by a subclass are declared abstract.

Template method pattern

```
public abstract class Beverage {  
  
    final void prepareRecepie(){  
        boilWater();  
        brew();  
        addCondiments();  
        pourInCup();  
    }  
  
    abstract void brew();  
  
    abstract void addCondiments();  
  
    void boilWater(){  
        System.out.println("Boiling water.");  
    }  
  
    void pourInCup(){  
        System.out.println("Pour into cup.");  
    }  
}
```

```
public class Tea extends Beverage {  
  
    @Override  
    void brew() {  
        System.out.println("Steeping the Tea.");  
    }  
  
    @Override  
    void addCondiments() {  
        System.out.println("Adding Lemon.");  
    }  
}
```

```
public class Coffie extends Beverage {  
  
    @Override  
    void addCondiments() {  
        System.out.println("Add sugar and milk.");  
    }  
  
    @Override  
    void brew() {  
        System.out.println("Stripping coffee through filter.");  
    }  
}
```

Template method pattern

```
package design.pattern.templateMethod;

public class TestTemplateMethod {

    static Beverage beverage = null;

    public static void main(String[] args) {

        System.out.println("=====Tea===== \n");
        Beverage tea = new Tea();
        tea.prepareRecepie();

        System.out.println("=====Coffie===== \n");
        Beverage coffie = new Coffie();
        coffie.prepareRecepie();
    }
}
```

```
<terminated> TestTemplateMethod [Java A]
=====
Tea=====
```

Boiling water.
Steeping the Tea.
Adding Lemon.
Pour into cup.
=====Coffie=====

Boiling water.
Stripping coffie through filter.
Add suger and milk.
Pour into cup.

References

- Head First Design Patterns: *by Eric Freeman & Elisabeth Freeman*
- Design Patterns: Elements of Reusable Object Oriented Software: *Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides (GOF)*
<http://www.hillside.net>
- Core Security Patterns: Best Practices and Strategies J2EE Web Services and Identity Management: *Chris Steel, Ramesh Nagappan, Ray Lai, Sun Microsystems*
- Core J2EE Patterns, Best Practices and Design Strategies: *Deepak Alur, John Crupi, Dan Malks, 2nd Edition, Prentice Hall/ Sun Microsystems, 2003*
- <http://www.javaworld.com/jw-11-1998/jw-11-techniques.html>

The End





Architectural Activities & Design Process

**Software Architecture
3rd Year – Semester 1
Lecture 4**

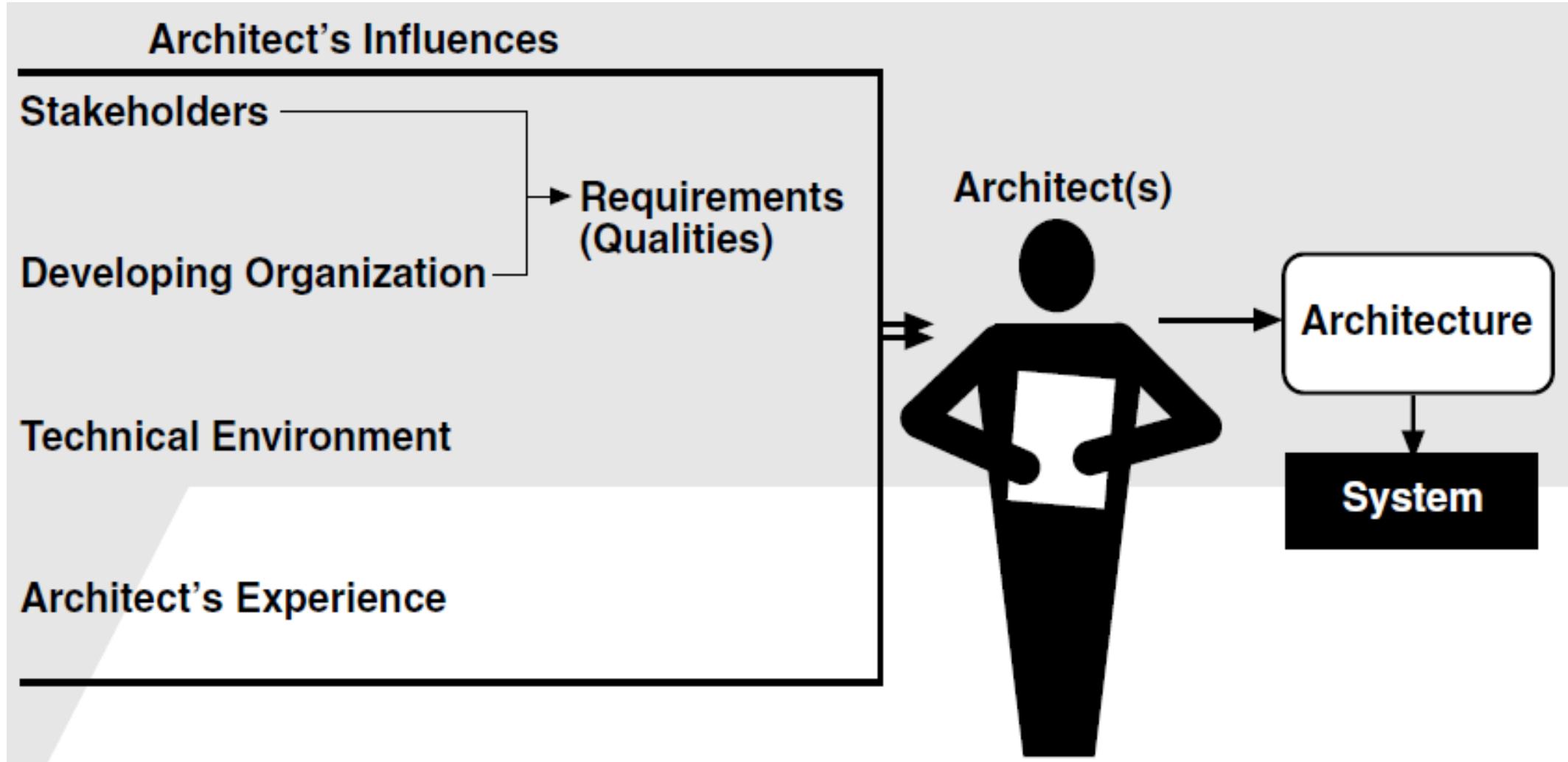
Architectural Process & the ABC

- What activities are involved in creating a Software Architecture?
- Using the selected Architecture how to realize a Design and then Implement or Manage the Evolution of a target system or application?

Where do Architectures come from?

- An architecture is the result of a set of business and technical decisions
- Influenced by:
 - Stakeholders
 - Management: Low Cost | Marketing: Time to Market | End User: UX, Security
 - Developing Organization
 - Investments already made | Future Investments | Organizational Structure
 - Experience & Background of the Architect(s)
 - Technical Skills | Domain Knowledge
 - Technical Environment
 - Software Engineering Techniques, practices & processes

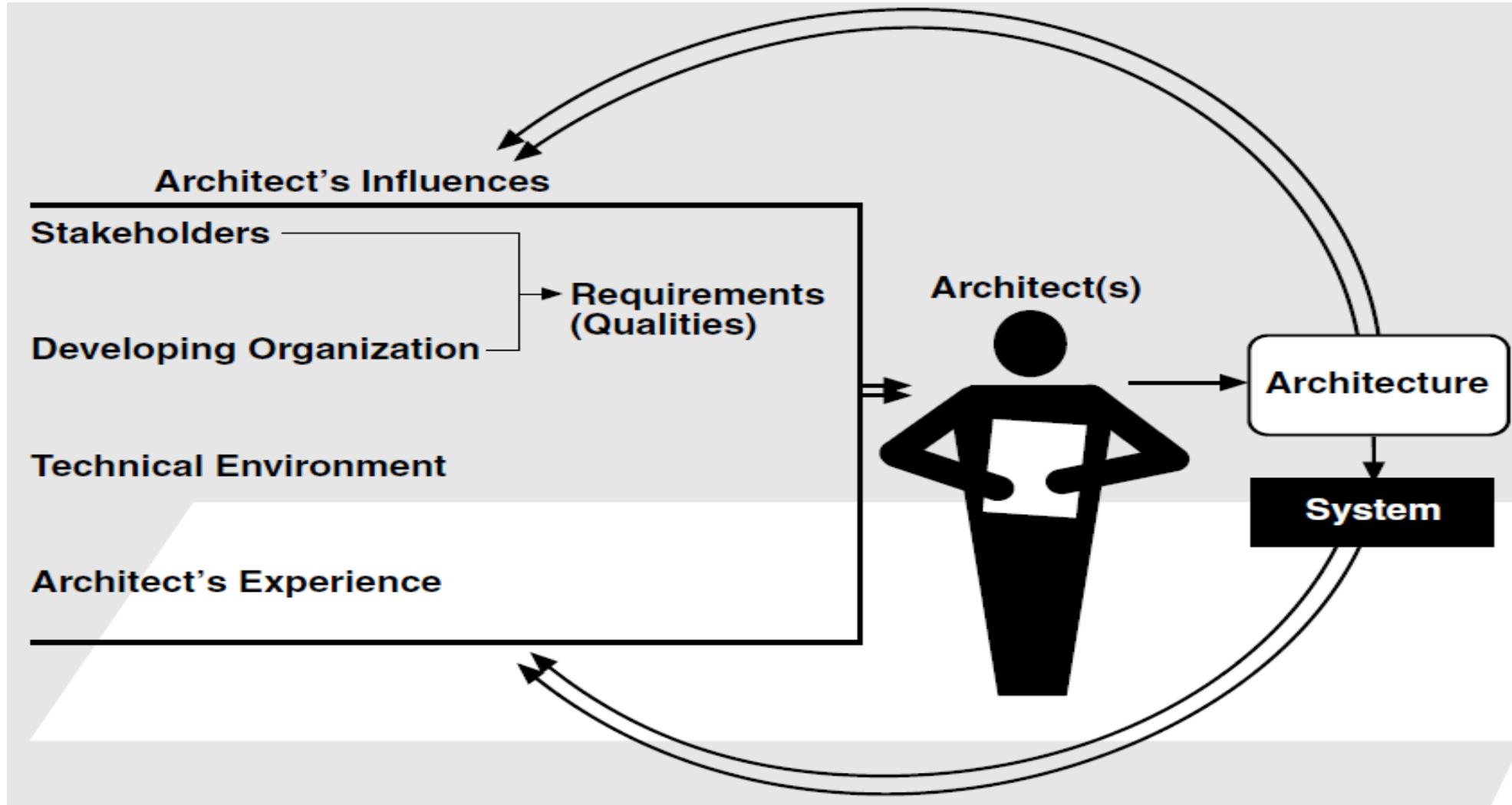
Architect's Influence



Architecture Business Cycle (ABC)

- Software architecture is a result of technical, business, and social influences.
- Its existence in turn affects the technical, business, and social environments that subsequently influence future architectures.
- We call this cycle of influences, from the environment to the architecture and back to the environment, the Architecture Business Cycle (ABC)

The Architecture Business Cycle



Ramifications of Architecture

- Affects the structure of the developing organization
 - Team Structure, Specialized Skills for Teams, etc.
- Affects the Goals of development organization
 - The organization may adjust its goals depending on the result of the current system to explore the future markets
- Affects future customer requirements
- Affects the Architect
 - Architect also will gain more experience based on the results
- Affects the Development Process & Culture

Architectural Activities

- Creating the Business Case for the System
- Understanding the Requirements
- Creating or Selecting the Architecture
- Documenting and Communicating the Architecture
- Analyzing or Evaluating the Architecture
- Implementing the system based on the Architecture
- Ensuring that the implementation Conforms to the Architecture

Creating the Business Case for the System

- Assess the market need for a system
- How much should the product cost?
- What is its targeted market?
- What is its targeted time to market?
- Will it need to interface with other systems?
- Are there system limitations that it must work within?

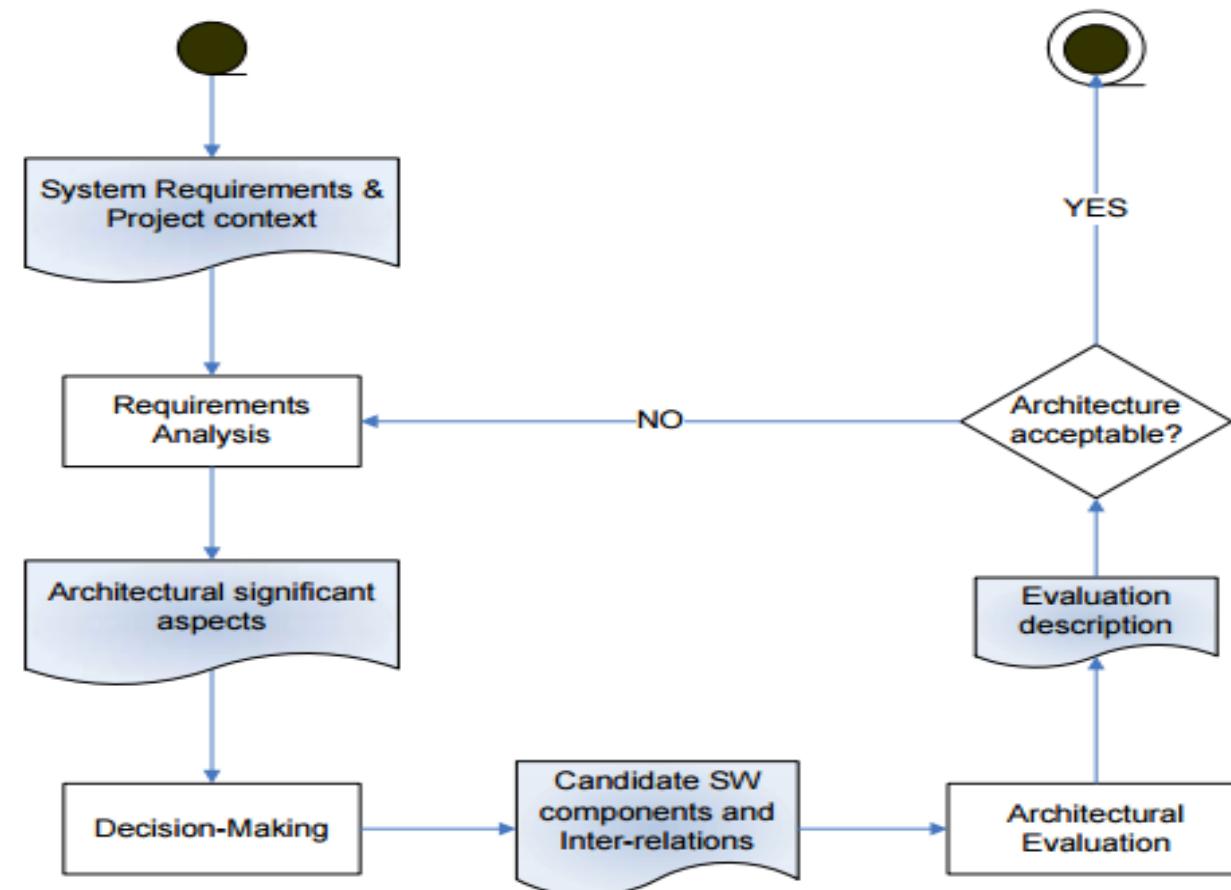
Note: Above cannot be decided solely by an architect, but if an architect is not consulted in the creation of the business case, it may be impossible to achieve the business goals!

Understanding the Requirements

- Various methods exists for capturing functional requirements;
 - OOAD uses Scenarios or Use Cases
 - Safety Critical Systems uses Finite-state-machine models or Formal specification language
- For Non-Functional requirements;
 - Quality Attribute Scenarios
- Prototyping
- Domain Modeling

Creating or Selecting the Architecture

- The process of how to create an architecture to achieve its behavioral and quality requirements



Documenting and Communicating the Architecture

- For the architecture to be effective as the backbone of the project's design, it must be communicated clearly and unambiguously to all of the stakeholders
 - Developers: must understand the work assignments it requires of them
 - Testers: must understand the task structure it imposes on them
 - Management: must understand the scheduling implications it suggests
- Architectural documentation should be informative, unambiguous, and readable by many people with varied backgrounds

Analyzing or Evaluating the Architecture

- In any design process there will be multiple candidate designs considered, some will be rejected immediately and others will contend for primacy
- Choosing among these competing designs in a rational way is one of the architect's greatest challenges
- Methods:
 - Architecture Tradeoff Analysis Method (ATAM) is the most mature methodology
 - Cost Benefit Analysis Method (CBAM) focuses more on economic implications of architectural decisions

Implementing the system based on the Architecture

- Ensure the developers are kept faithful to the structures and interaction protocols constrained by the architecture
- The Environment should assist developers in creating and maintaining the architecture
- Architect's involvements:
 - Technology & Technical Infrastructure Selection & Setup
 - Setting up Appropriate Software Engineering Processes
 - Creating the Team
 - Identify Technical specialists
 - Identify Skill Gaps and mitigate (trainings, etc.)

Ensuring that the implementation Conforms to the Architecture

- Constant vigilance is required to ensure that the actual architecture and its representation remain faithful to each other in the maintenance phase

What makes a Good Architecture?

- No such thing as an inherently good or bad architecture
- Architectures are more or less fit for some stated purpose
- Architectures can be evaluated
 - One of the great benefits of paying attention to them
 - Should be evaluated only in the context of specific goals
- Rules of Thumb (from knowledge & experience):
 - Process Recommendations:
 - Functional Requirements & identifying Quality Attributes which are of high priority
 - Analyze & formally evaluate before it is too late to change
 - Product Recommendations:
 - Software Principles e.g. Information hiding, Separation of concerns
 - Write tasks or processes to allow easy reallocation, perhaps at runtime

Architectural Design Process

- Objectives of the Design Process:
 - Creativity
 - Enhance your skillset
 - Provide new tools
 - Method
 - Focus on highly effective techniques
 - Develop judgment: when to develop novel solutions, and when to follow established method

Engineering Design Process

- Feasibility:
 - Identifying a set of feasible concepts for the design as a whole
- Preliminary design:
 - Selection and development of the best concept
- Detailed design:
 - Development of engineering descriptions of the concept
- Planning:
 - Evaluating and altering the concept to suit the requirements of production, distribution, consumption and product retirement

Potential Problems

- If the designer is unable to produce a set of feasible concepts, progress stops
- As problems and products increase in size and complexity, the probability that any one individual can successfully perform the first steps decreases
- The standard approach does not directly address the situation where system design is at stake, i.e. when relationship between a set of products is at issue
- As complexity increases or the experience of the designer is not sufficient, alternative approaches to the design process must be adopted

Alternative Design Strategies

- Standard
 - Linear model
- Cyclic
 - Process can revert to an earlier stage
- Parallel
 - Independent alternatives are explored in parallel
- Adaptive (“lay tracks as you go”)
 - The next design strategy of the design activity is decided at the end of a given stage
- Incremental
 - Each stage of development is treated as a task of incrementally improving the existing design

Identifying a Viable Strategy

- Use fundamental design tools: abstraction and modularity
 - But how?
- Inspiration, where inspiration is needed. Predictable techniques elsewhere.
 - But where is creativity required?
- Applying own experience or experience of others

Tools & Patterns of Software Engineering

- Abstraction
 - Abstraction(1): look at details, and abstract “up” to concepts
 - Abstraction(2): choose concepts, then add detailed substructure, and move “down”
- Separation of Concerns
- Architectural Patterns & Styles
 - Layered
 - Model View Controller (MVC)
 - Client-Server

Controlling the Design Strategy

- Manage the Activity: Exploring diverse approaches to the problem demands that some care be used in managing the activity
- Review: Identify and review critical decisions
- Cost: Relate the costs of research and design to the penalty for taking wrong decisions
- Enforce: Insulate uncertain decisions
- Cross Check with Requirements: Continually re-evaluate system requirements in light of what the design exploration yields

Insights from Requirements

- In many cases new architectures can be created based upon experience with and improvement to pre-existing architectures
- Requirements can use a vocabulary of known architectural choices and therefore reflect experience
- The interaction between past design and new requirements means that many critical decisions for a new design can be identified or made as a requirement

Insights from Implementation

- Constraints on the implementation activity may help shape the design
- Externally motivated constraints might dictate
 - Use of a middleware
 - Use of a particular programming language
 - Software reuse
- Design and implementation may proceed cooperatively
- Initial partial implementation activities may yield critical performance or feasibility information

References

- Software Architecture in Practice, 2nd Ed., by Len Bass, Paul Clements, Rick Kazman
- <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch01.html>
- <http://www.ics.uci.edu/~taylor/classes/211/DesignAndArchitecture.pdf>

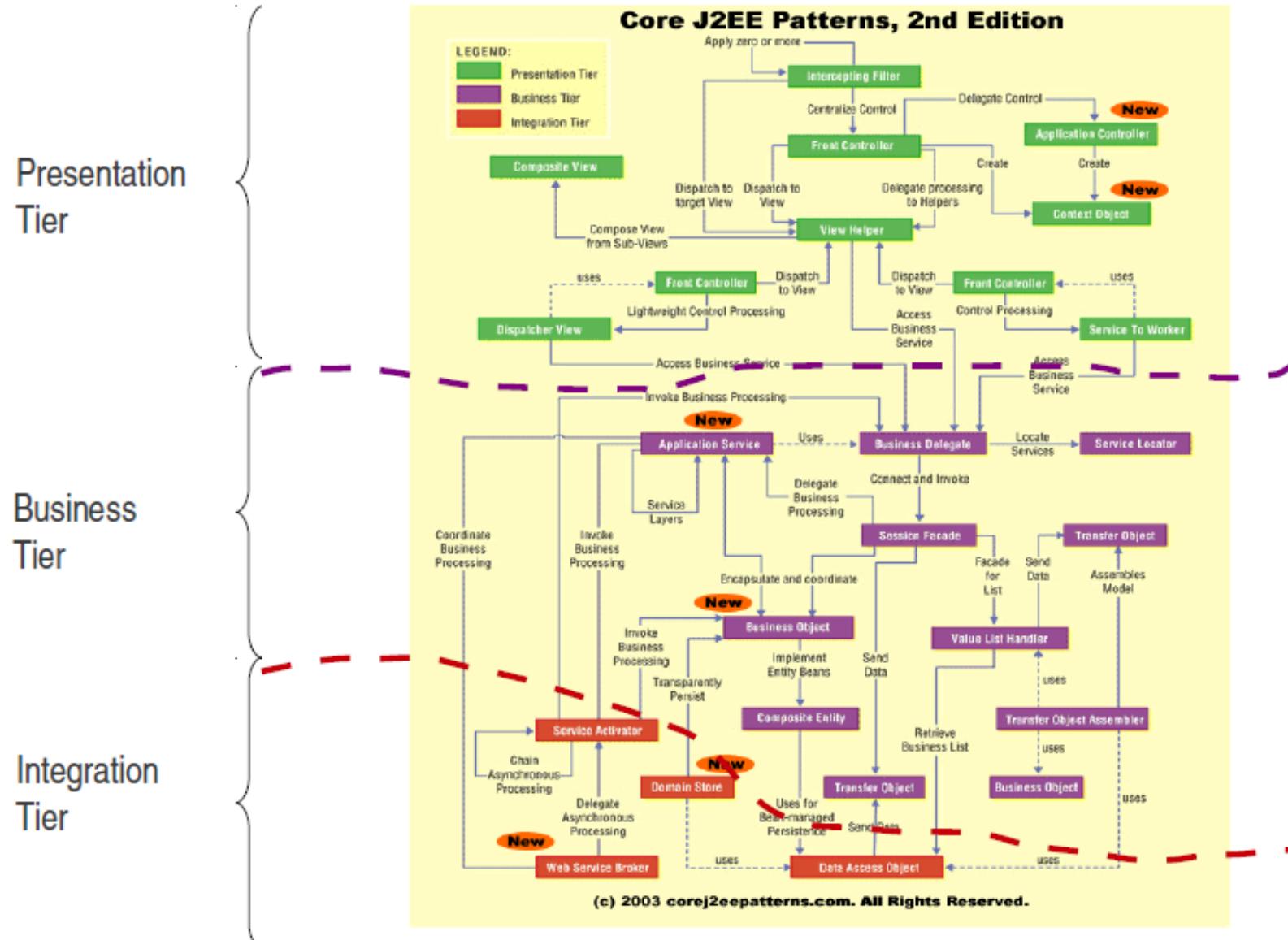


Presentation Layer Patterns

Lecture 03

by Udara Samaratunge

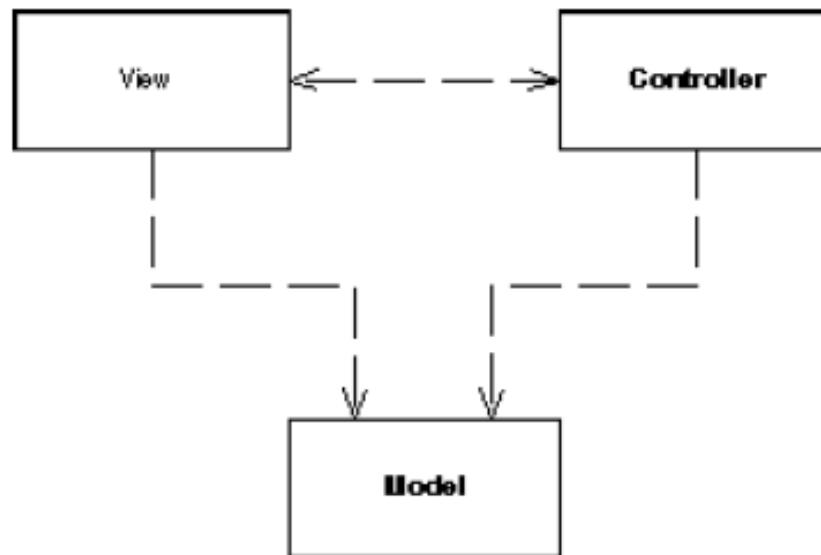
J2EE Architecture Blueprint



Web Presentation Design Patterns

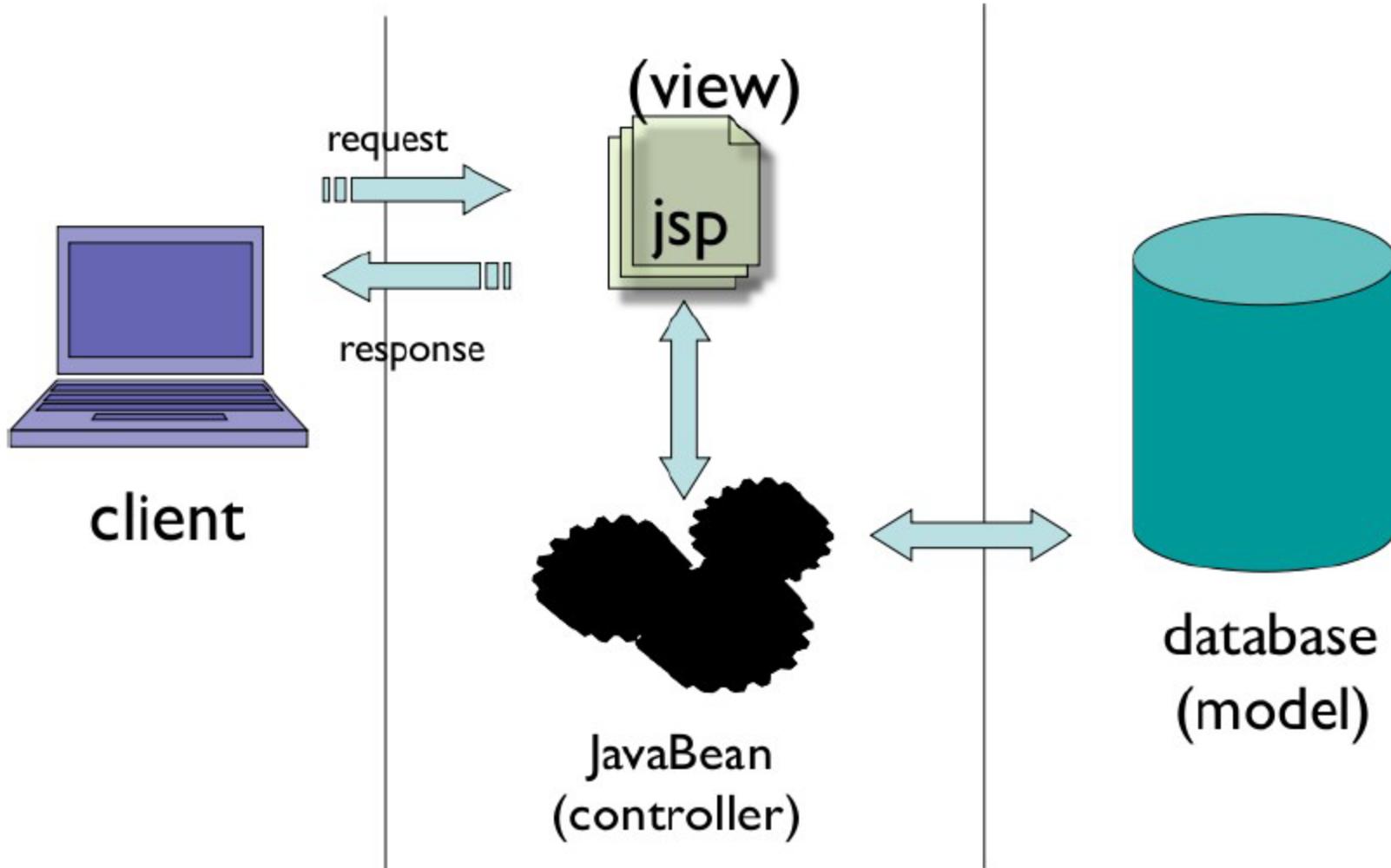
- ⌚ Model View Controller (MVC)
- ⌚ Intercepting Filter Pattern
- ⌚ Front Controller
- ⌚ View Helper
- ⌚ Composite View
- ⌚ Dispatcher View
- ⌚ Service to Worker

Model-View Controller

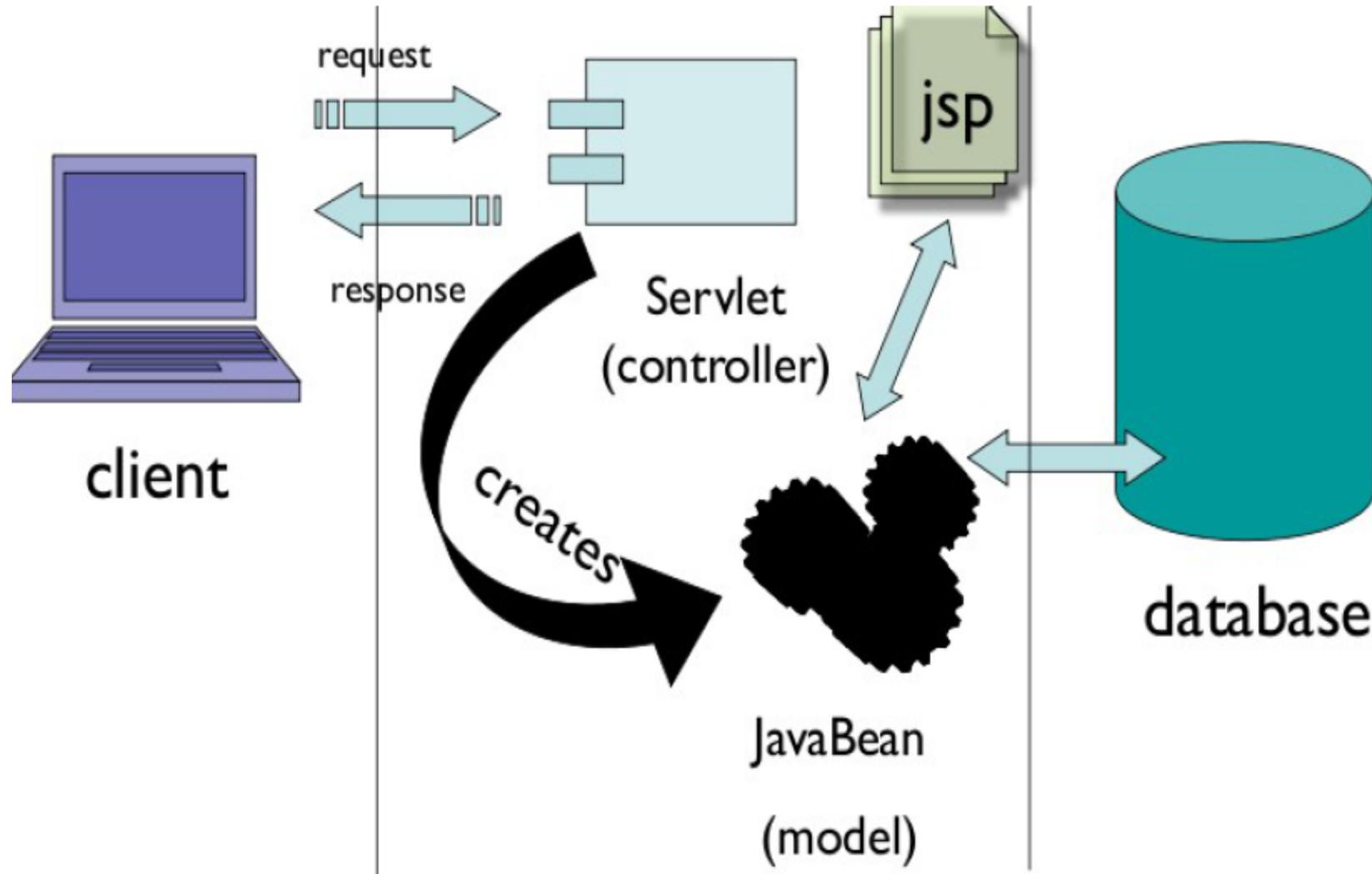


Splits user interface interaction into three distinct roles

Model 1 Architecture



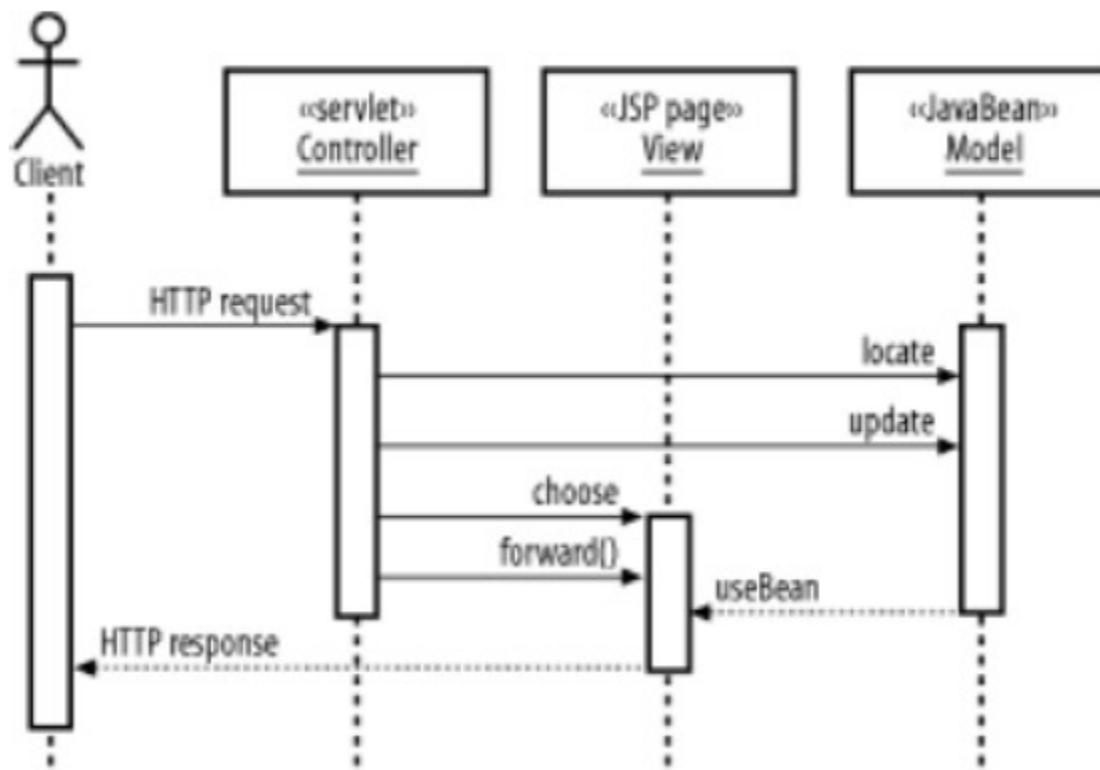
Model 2 Architecture



Model View Controller - (MVC)

- ⌚ The MVC pattern provides the basic structure for a web application
- ⌚ In J2EE, the following components define the structure of the web application
 - The JavaBeans provides the *Model*
 - The JSPs provide the *View*
 - The controller Servlet provides the *Controller*

Model View Controller - (MVC)



MVC Interaction in J2EE (Reference: *J2ee Design Patterns*)

Model View Controller - (MVC)

⌚ Model:

Holds the *data*, *state* and *application logic*

Can send notification of *state* changes to observers

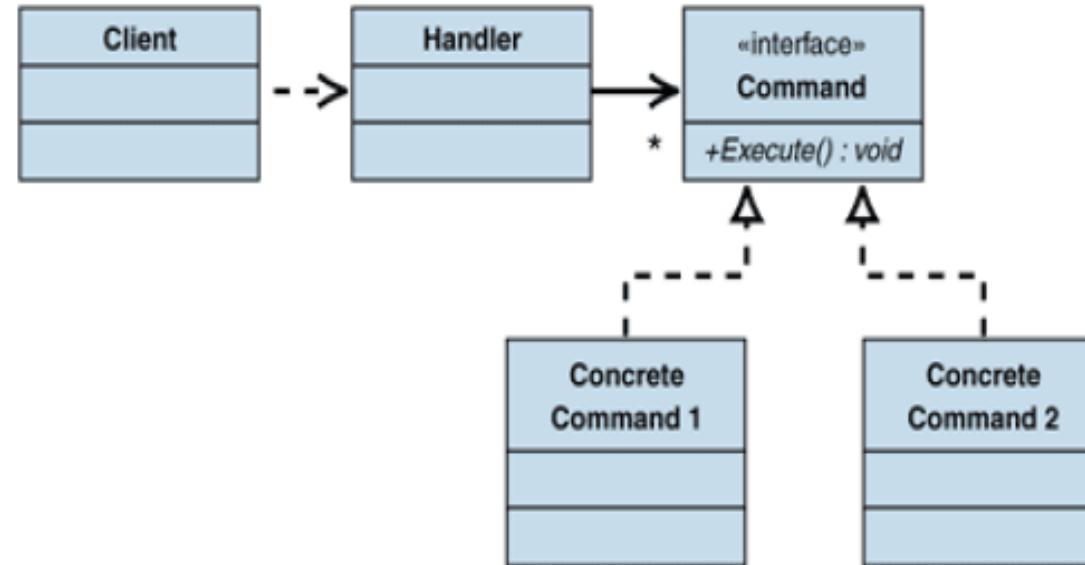
⌚ View:

Gives you the presentation of the model. This is
stateless

⌚ Controller:

Takes user input and figure out what it means to the model

Front Controller



A controller that handles all the requests for the web application

Exercise 01 - In class Activity

- Assume that you have a **light** in both **Living Room** and **Kitchen** you use a **Remote Controller** to switch **on** and **off** lights. Command and Light interfaces are given with the Test class including final Output. Implement the **OnCommand**, **OffCommand**, **LivingRoomLight**, **KitchenLight**, and **RemoteController** classes.

```
public interface Light {  
    public void on();  
    public void off();  
}
```

```
public interface Command {  
    public void execute();  
}
```

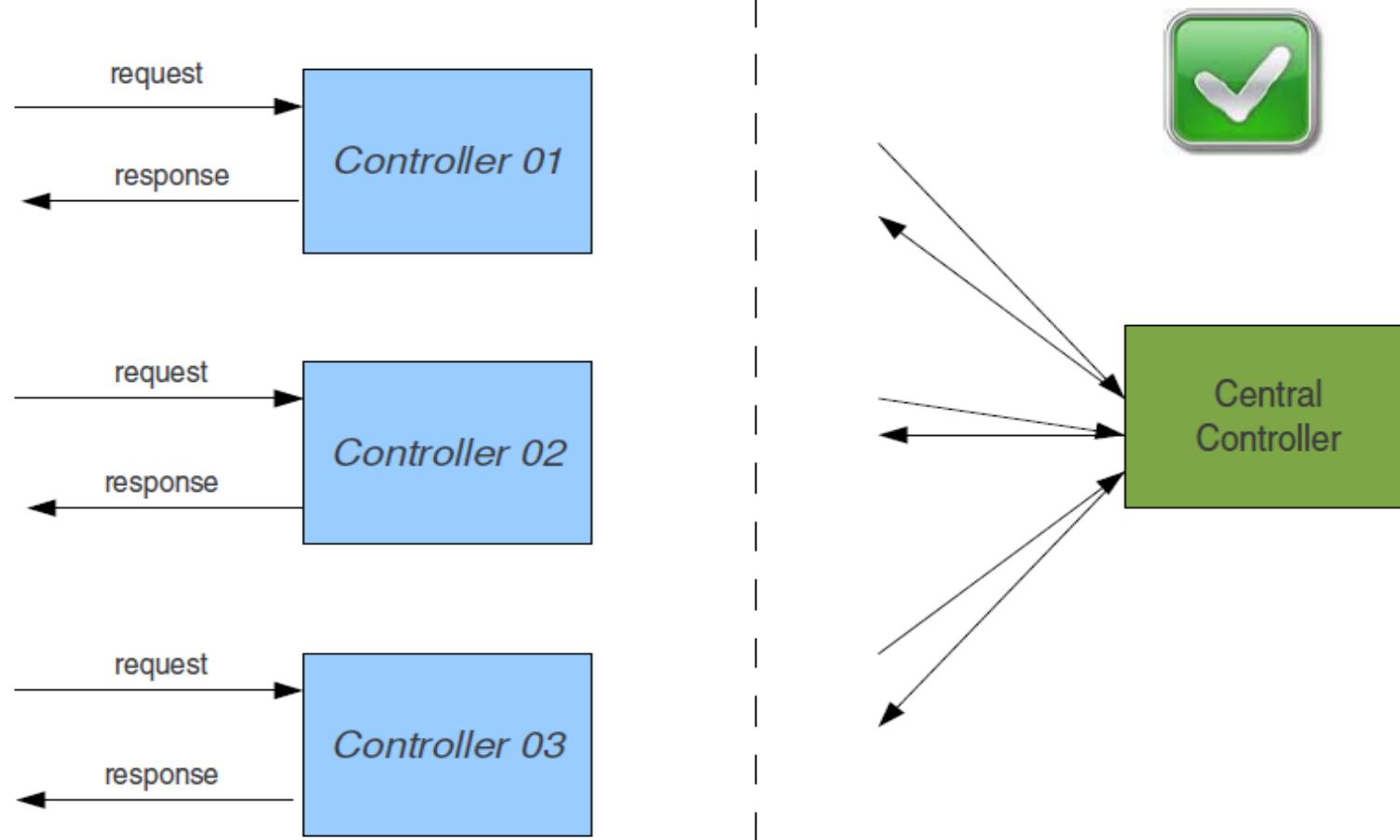
```
package design.pattern.command;  
  
public class Test {  
    public static void main(String[] args) {  
        Light livingRoomLight = new LivingRoomLight();  
        Light kitchenLight = new KitchenLight();  
  
        RemoteController remoteController = new RemoteController();  
  
        Command lightOnCommand = new LightOnCommand(livingRoomLight);  
        Command lightOffCommand = new LightOffCommand(livingRoomLight);  
        remoteController.setCommand(lightOnCommand, lightOffCommand);  
        remoteController.onButtonWasPushed();  
        remoteController.offButtonWasPushed();  
  
        Command lightOnCommand1 = new LightOnCommand(kitchenLight);  
        Command lightOffCommand1 = new LightOffCommand(kitchenLight);  
        remoteController.setCommand(lightOnCommand1, lightOffCommand1);  
        remoteController.onButtonWasPushed();  
        remoteController.offButtonWasPushed();  
    }  
}
```

```
<terminated> Test (7) [Java Application]  
Switch on() Living Room Light  
Switch off() Living Room Light  
Swich on() Kitchen Light  
Swich off() Kitchen Light
```

Front Controller

- ⌚ Front Controller is the central controlling point of a web application
- ⌚ In a complex web site, there are many tasks need to be followed when handling a request. For example:
 - Authentication/ Authorization
 - Delegating to business Processors
 - Providing different view to the application, etc
- ⌚ If we duplicate the input controller behavior (entry behavior) to all these tasks, the **behavior can be duplicated** across

Front Controller

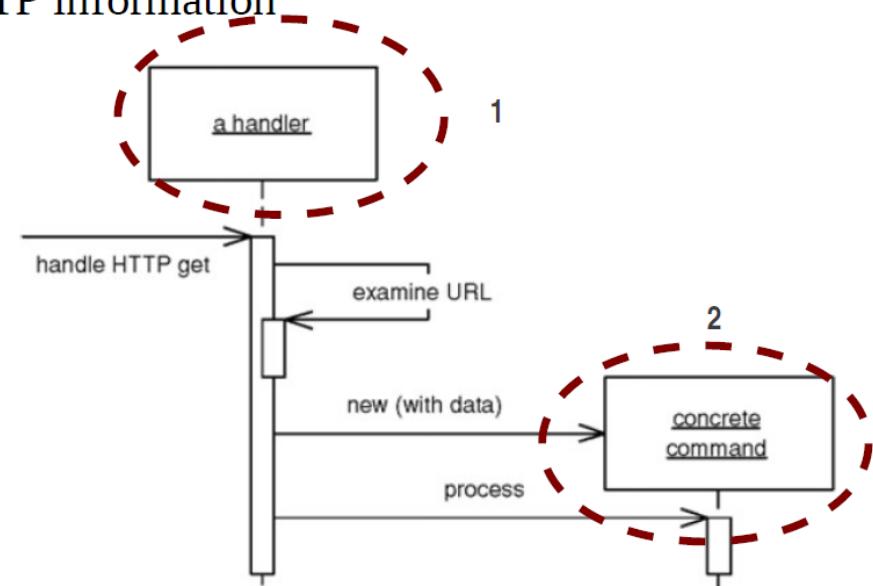


Front Controller - Benefits

- Provides a central entry point that controls and manages the web request
- Centralizing control in the controller and reducing business logic in the view promotes code reuse across requests
- Coordinates the request dispatching – Dispatchers are responsible for view management navigation

Front Controller

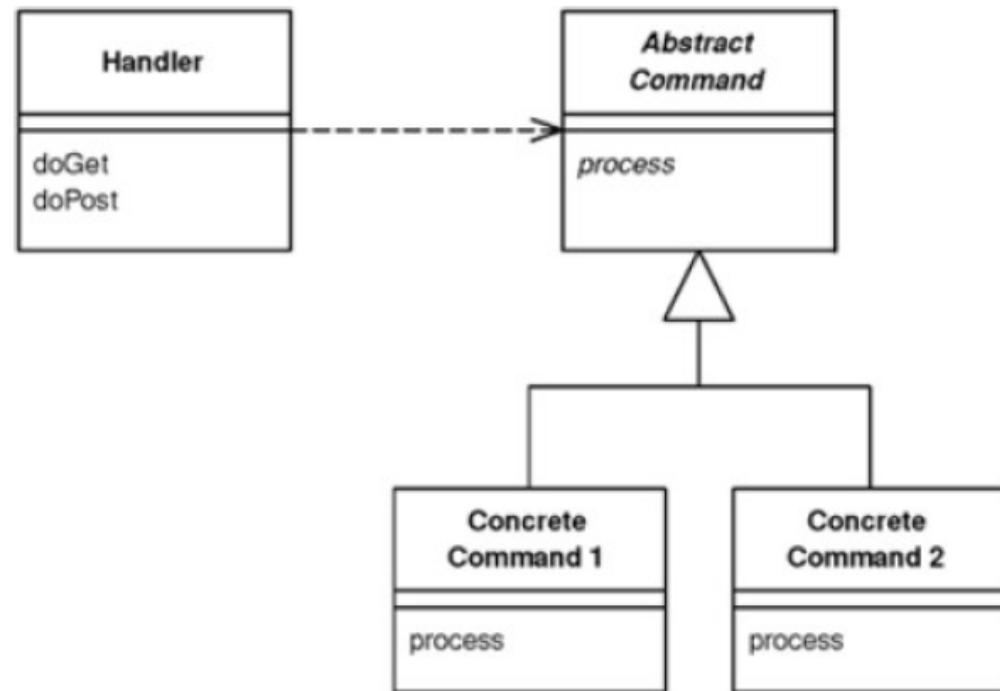
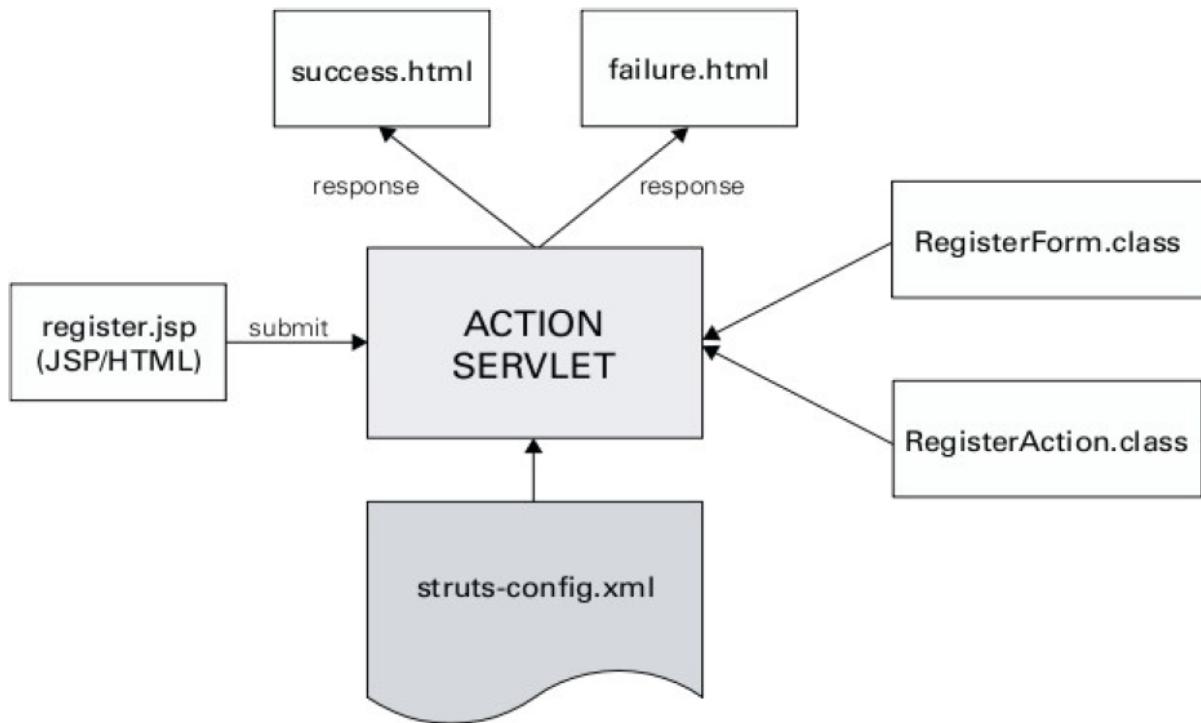
- The *Web Handler* is fairly a simple class that does nothing other than deciding which command to run
- The *commands* are also classes that are often passed with HTTP information



Front Controller – In Struts

The Front Controller consolidates all request handling by channeling requests through a single handler object.

- In Struts, *ActionServlet* is the Front Controller



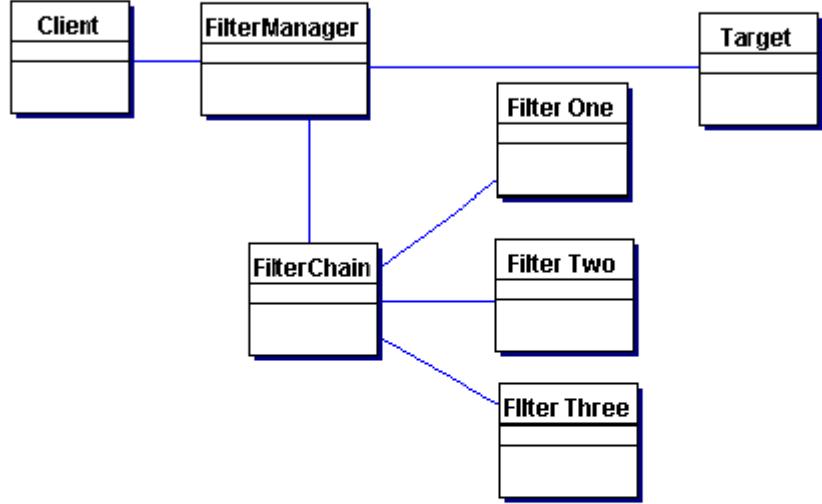
Front Controller

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    FrontCommand command = getCommand(request);
    command.init(getServletContext(), request, response);
    command.process();
}

private FrontCommand getCommand(HttpServletRequest request) {
    try {
        return (FrontCommand) getCommandClass(request).newInstance();
    } catch (Exception e) {
        throw new ApplicationException(e);
    }
}

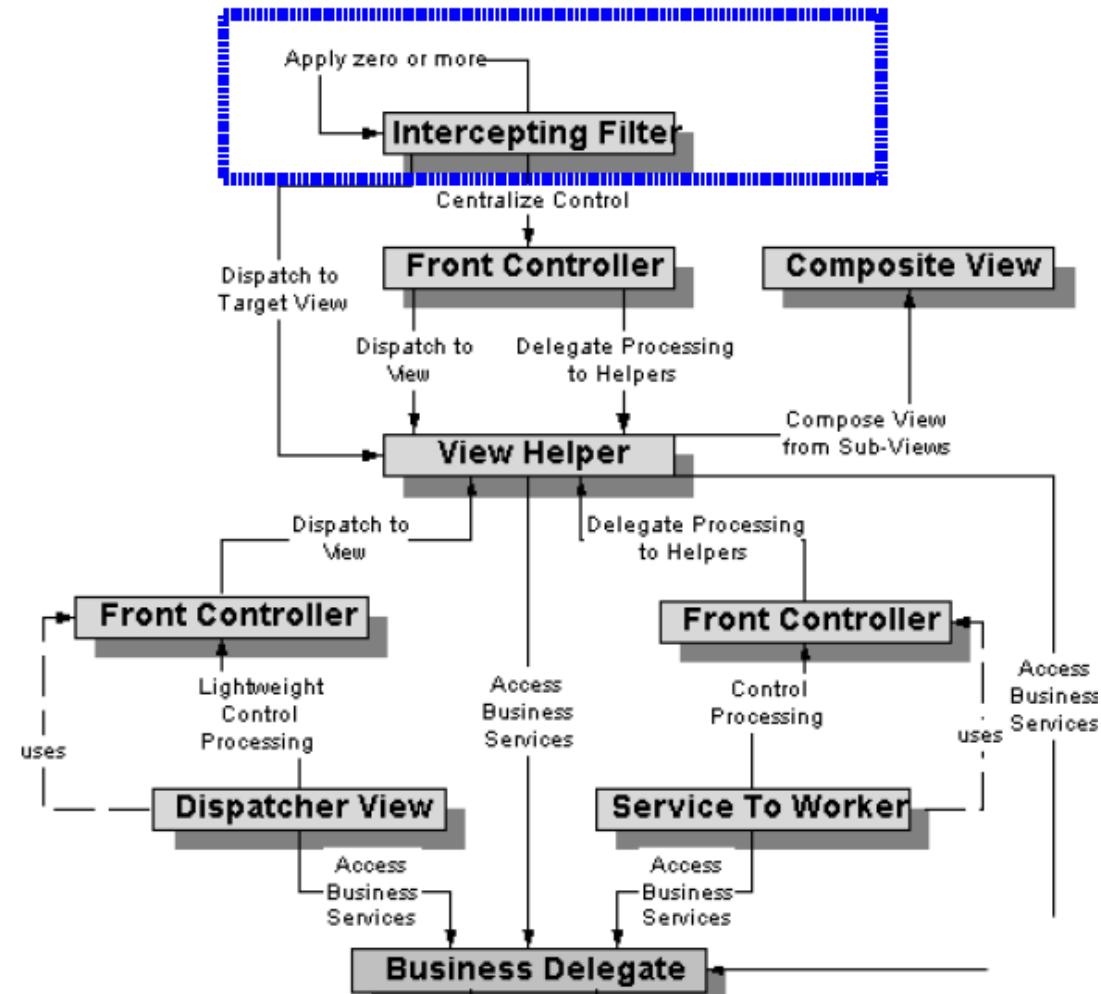
private Class getCommandClass(HttpServletRequest request) {
    Class result;
    final String commandClassName =
        "frontController." + (String) request.getParameter("command") + "Command";
    try {
        result = Class.forName(commandClassName);
    } catch (ClassNotFoundException e) {
        result = UnknownCommand.class;
    }
    return result;
}
```

Intercepting Filter



Create pluggable filters to process common services in a standard manner without requiring changes to core request processing code. The filters intercept incoming requests and outgoing responses, allowing preprocessing and post-processing. We are able to add and remove these filters unobtrusively, without requiring changes to our existing code

Intercepting Filter Pattern



Intercepting Filter Pattern

- ⌚ This is a presentation tier web pattern and is designed using several GOF design patterns

Java => (Core J2EE Patterns)

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html>

.NET => (MSDN)

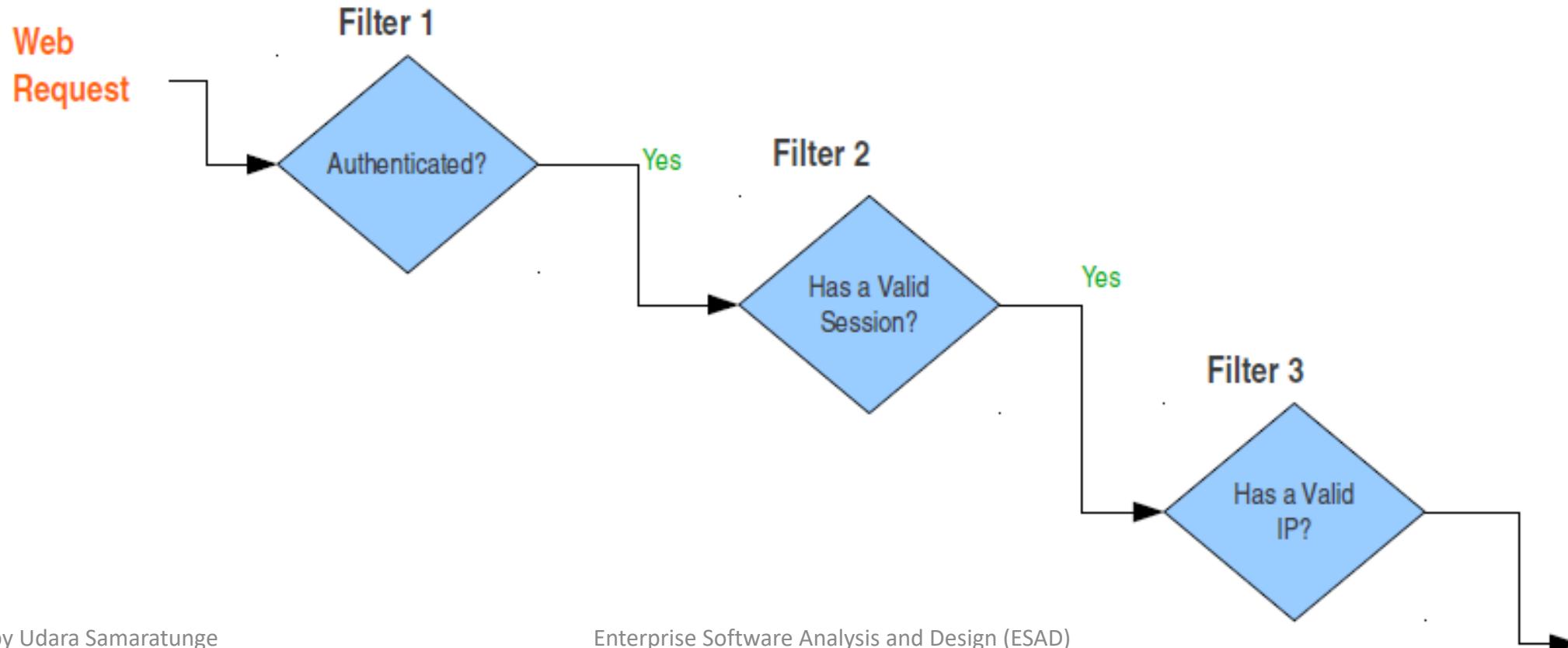
<http://msdn.microsoft.com/en-us/library/ms978727.aspx>

Intercepting Filter Pattern

- When a request enters a Web application, it often needs to pass a several **entrance tests** prior to the main processing stage. For example:
 - Has the client been authenticated?
 - Does the client have a valid session?
 - Is the client's IP address from a trusted network?
 - Does the request path violate any constraints?
 - What encoding does the client use to send the data?
 - Do we support the browser type of the client?

Intercepting Filter Pattern

The Solution: To have a simple mechanism to add or remove processing components(Filters), in which each component does a certain filtering.



Intercepting Filter Pattern

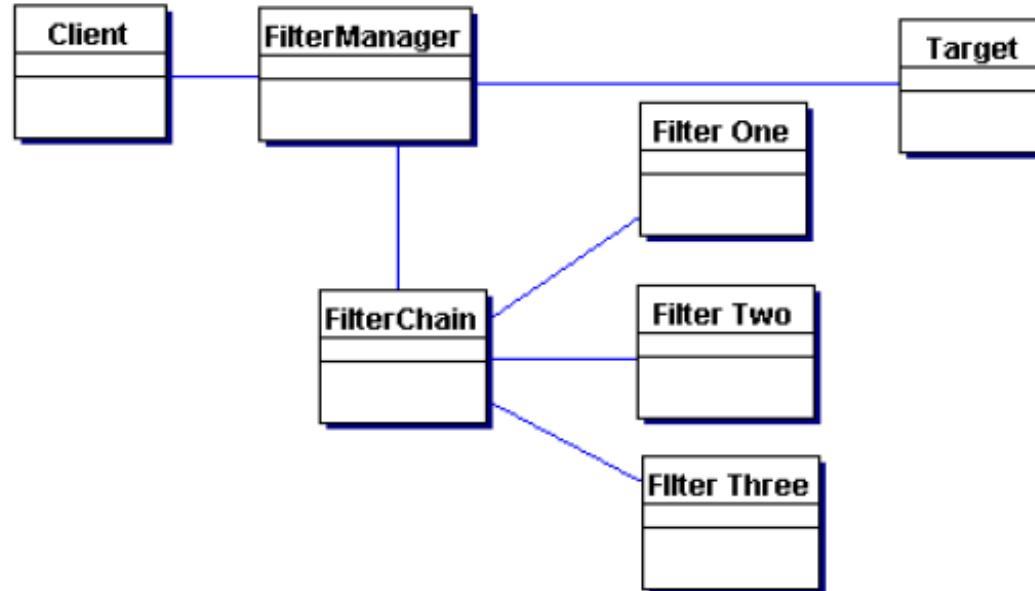
Creates pluggable filters to process common services in a standard manner without requiring changes to core request processing code.

The filters intercept incoming requests and outgoing responses, allowing preprocessing and post-processing.

We are able to add and remove these filters, without doing much changes to our existing code.

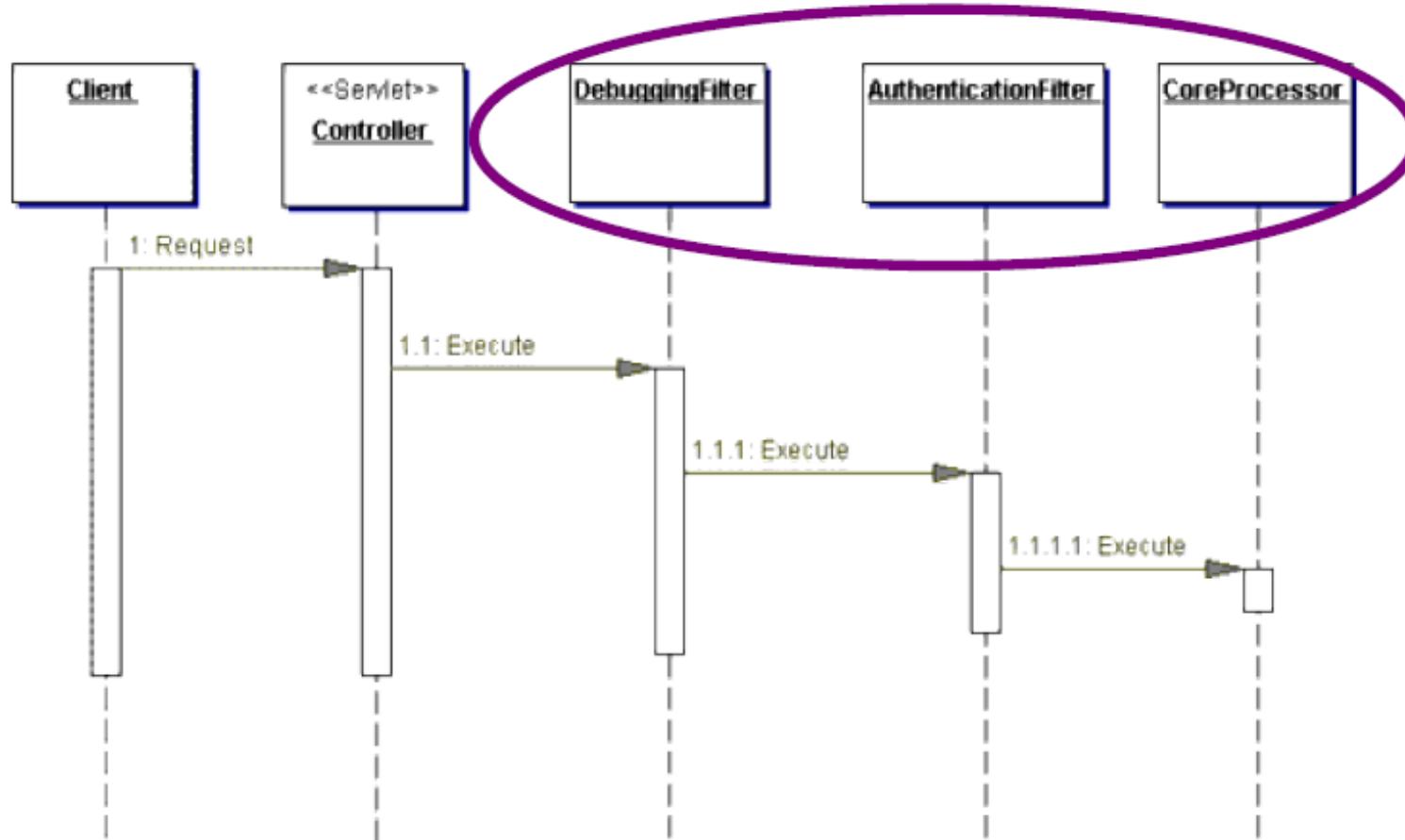
Intercepting Filter Pattern

These filters are components which are totally independent from the application code. They may be added or removed declaratively



Intercepting Filter Pattern (Custom Filters – Decorator implementation)

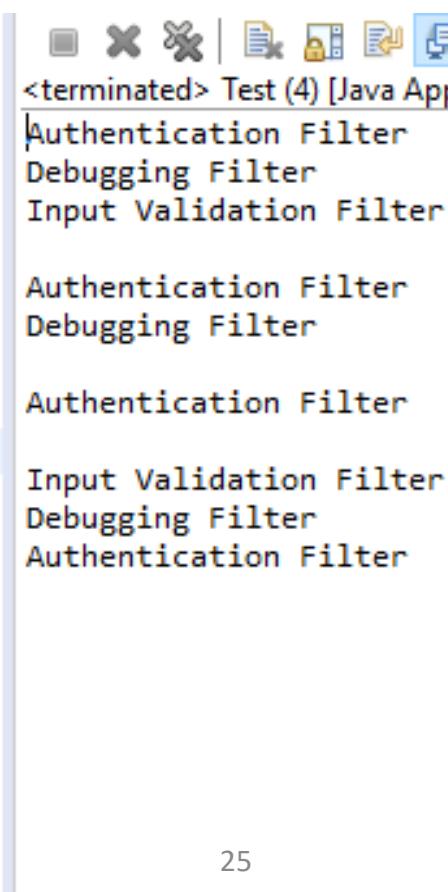
The Decorator Pattern [GoF] is used here



Exercise 02 - In class Activity

- Write the code for three filter classes **AuthenticationFilter**, **InputValidationFilter**, and **LoggingFilter**. You should implement the interface **IFilter** and override the method. As per the displayed output modify your filter classes accordingly.

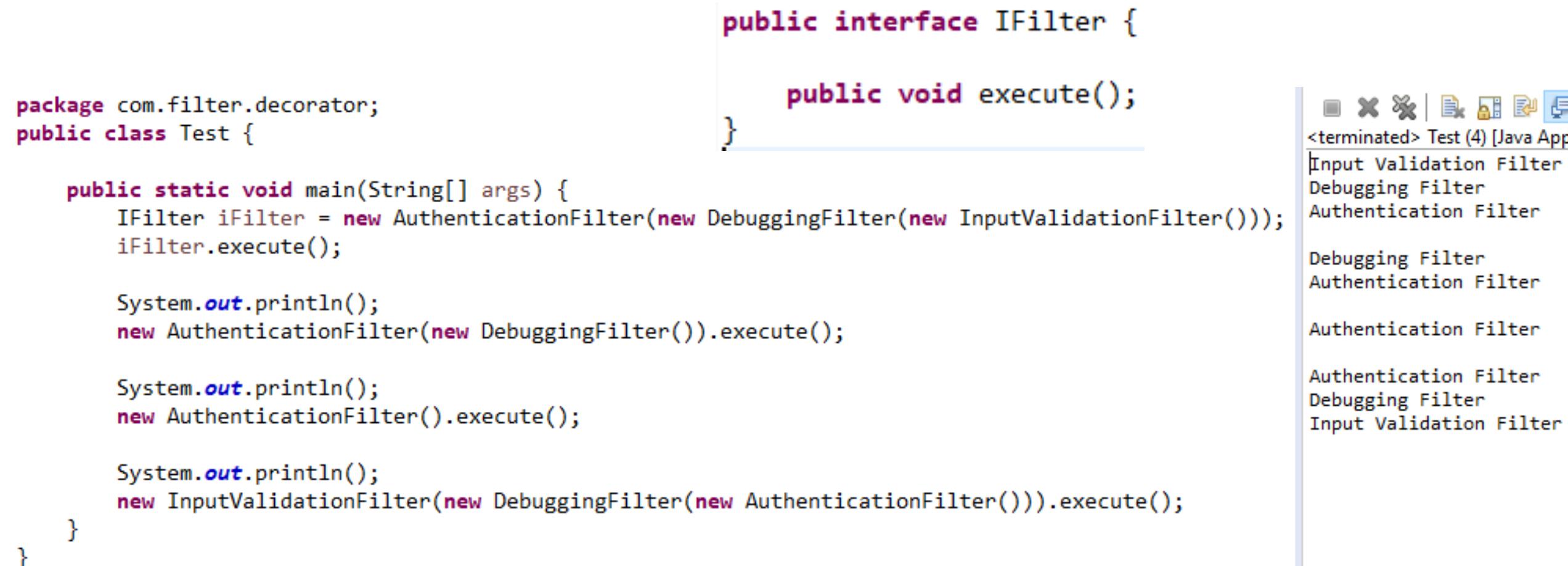
```
public interface IFilter {  
  
    public void execute();  
}  
  
package com.filter.decorator;  
  
public class Test {  
  
    public static void main(String[] args) {  
        IFilter iFilter = new AuthenticationFilter(new DebuggingFilter(new InputValidationFilter()));  
        iFilter.execute();  
  
        System.out.println();  
        new AuthenticationFilter(new DebuggingFilter()).execute();  
  
        System.out.println();  
        new AuthenticationFilter().execute();  
  
        System.out.println();  
        new InputValidationFilter(new DebuggingFilter(new AuthenticationFilter())).execute();  
    }  
}
```



Exercise 03 - In class Activity

- Remodify the above program it should start printing inner object to outer object as depicted in the console output.

```
public interface IFilter {  
    public void execute();  
}  
  
package com.filter.decorator;  
public class Test {  
  
    public static void main(String[] args) {  
        IFilter iFilter = new AuthenticationFilter(new DebuggingFilter(new InputValidationFilter()));  
        iFilter.execute();  
  
        System.out.println();  
        new AuthenticationFilter(new DebuggingFilter()).execute();  
  
        System.out.println();  
        new AuthenticationFilter().execute();  
  
        System.out.println();  
        new InputValidationFilter(new DebuggingFilter(new AuthenticationFilter())).execute();  
    }  
}
```



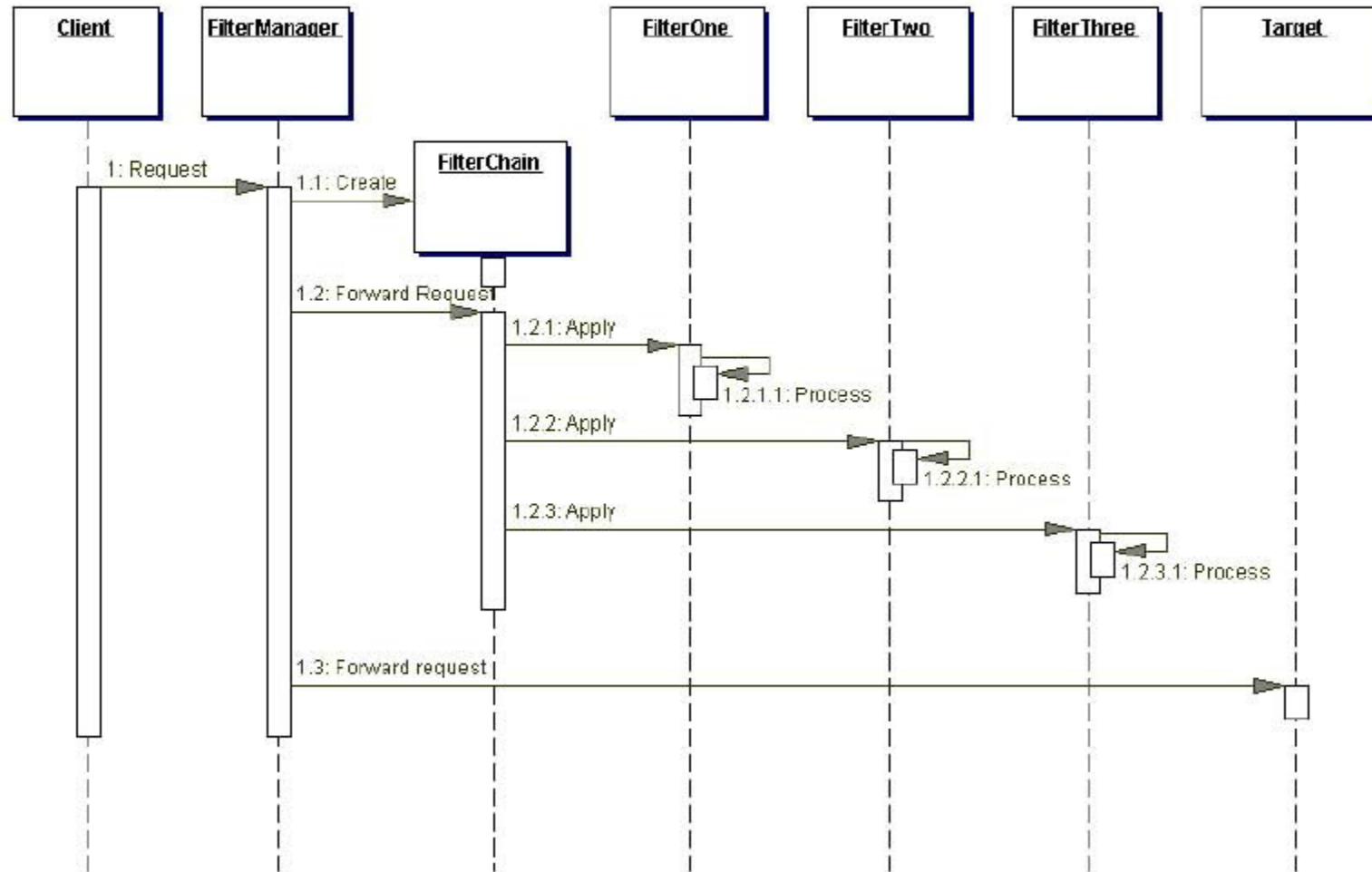
The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the provided Java code. The terminal window shows the execution output:

```
<terminated> Test (4) [Java App]  
Input Validation Filter  
Debugging Filter  
Authentication Filter  
  
Debugging Filter  
Authentication Filter  
  
Authentication Filter  
  
Authentication Filter  
Debugging Filter  
Input Validation Filter
```

Intercepting Filter Non-Decorator Implementation

Intercepting Filter Pattern

(Custom Filters – Non-Decorator implementation)



Source: Core J2EE Patterns

Exercise 04 - In class Activity

- Remodify the above three filters according to the Non-Decorator Filter chain as per the below output. You should implement the same classes (**AuthenticationFilter**, **InputValidationFilter**, and **LoggingFilter**) and implement the interface **IFilter** and override the method. You can maintain chain of filters as **ArrayList** or **Vector** and use **FilterManager** class to invoke the Filter chain process. Filter Manager class is given below implement the **FilterChain** class. As per the displayed output modify your filter classes accordingly.

```
package com.filter.chain;

public class FilterManager {

    public static void main(String[] args) {

        FilterChain filterChain = new FilterChain();
        filterChain.addFilter(new DebuggingFilter());
        filterChain.addFilter(new AuthenticationFilter());
        filterChain.addFilter(new InputValidationFilter());
        filterChain.processFilter();

        filterChain.addFilter(new AuthenticationFilter());
        filterChain.addFilter(new DebuggingFilter());
        filterChain.processFilter();
    }
}
```

```
<terminated> FilterManager [Java]
Debugging Filter
Authentication Filter
Input Validation Filter
```

```
Debugging Filter
Authentication Filter
Input Validation Filter
Authentication Filter
Debugging Filter
```

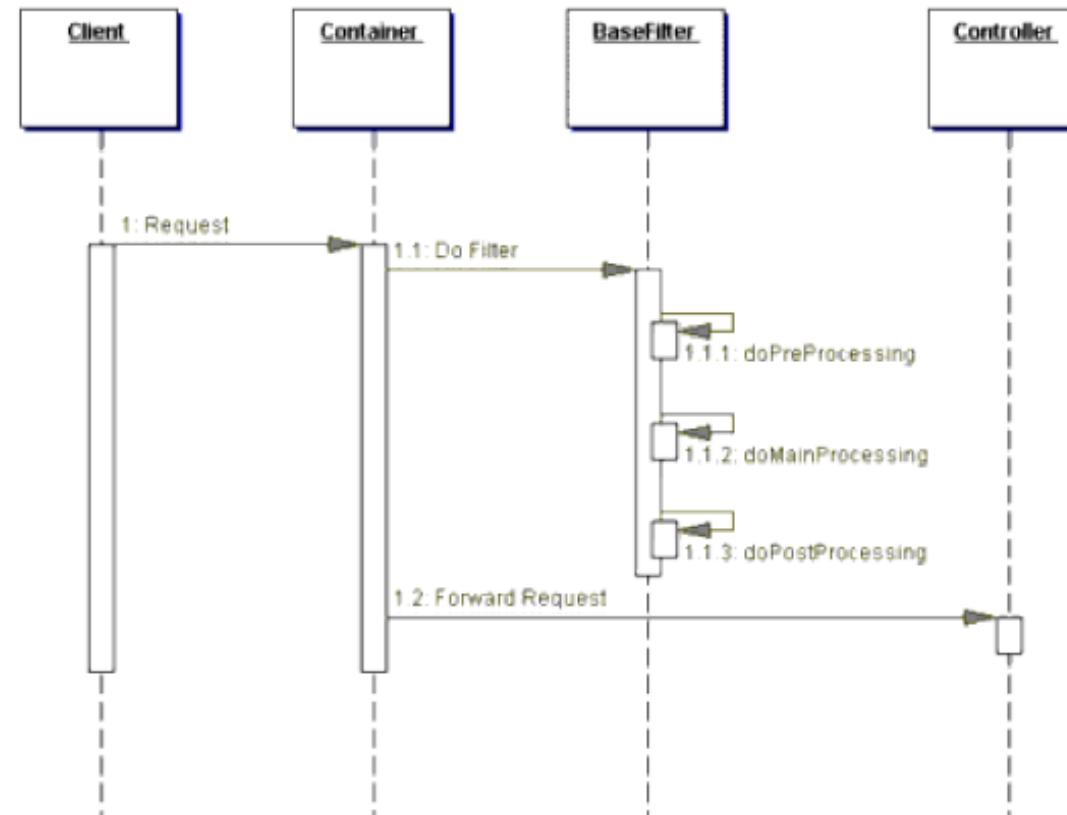


```
public class FilterChain {  
    // filter chain  
    private Vector myFilters = new Vector();  
  
    // Creates new FilterChain  
    public FilterChain() {  
        // plug-in default filter services for example  
        // only. This would typically be done in the  
        // FilterManager, but is done here for example  
        // purposes  
        addFilter(new DebugFilter());  
        addFilter(new LoginFilter());  
        addFilter(new AuditFilter());  
    }  
    public void processFilter(  
        javax.servlet.http.HttpServletRequest request,  
        javax.servlet.http.HttpServletResponse response)  
throws javax.servlet.ServletException,  
        java.io.IOException {  
        Filter filter;  
  
        Iterator filters = myFilters.iterator();  
        while (filters.hasNext())  
        {  
            filter = (Filter)filters.next();  
            // pass request & response through various  
            // filters  
            filter.execute(request, response);  
        }  
    }  
    public void addFilter(Filter filter) {  
        myFilters.add(filter);  
    }  
}
```



*The wrapping is handled by
HttpServletRequestWrapper
implemented by the Custom
Filter*

Intercepting Filter Pattern *(Template Filters)*



```
public abstract class TemplateFilter implements  
    javax.servlet.Filter {  
    private FilterConfig filterConfig;  
  
    public void setFilterConfig(FilterConfig fc) {  
        filterConfig=fc;  
    }  
  
    public FilterConfig getFilterConfig() {  
        return filterConfig;  
    }  
  
    public void doFilter(ServletRequest request,  
        ServletResponse response, FilterChain chain)  
        throws IOException, ServletException {  
        // Common processing for all filters can go here  
        doPreProcessing(request, response, chain);  
  
        // Common processing for all filters can go here  
        doMainProcessing(request, response, chain);  
  
        // Common processing for all filters can go here  
        doPostProcessing(request, response, chain);  
  
        // Common processing for all filters can go here  
  
        // Pass control to the next filter in the chain or  
        // to the target resource  
        chain.doFilter(request, response);  
    }  
    public void doPreProcessing(ServletRequest request,  
        ServletResponse response, FilterChain chain) {  
    }  
  
    public void doPostProcessing(ServletRequest request,  
        ServletResponse response, FilterChain chain) {  
    }  
  
    public abstract void doMainProcessing(ServletRequest  
        request, ServletResponse response, FilterChain  
        chain);  
}
```

Template
Method

Intercepting Filter Pattern (*Template Filters*)

```
public class DebuggingFilter extends TemplateFilter {  
    public void doPreProcessing(ServletRequest req,  
        ServletResponse res, FilterChain chain) {  
        //do some preprocessing here  
    }  
  
    public void doMainProcessing(ServletRequest req,  
        ServletResponse res, FilterChain chain) {  
        //do the main processing;  
    }  
}
```

This defines a specific processing by overriding the abstract *doMainProcessing* method and, optionally, *doPreProcessing* and *doPostProcessing*

Intercepting Filter Pattern

(Custom Filters – Non-Decorator implementation)



Limitations:

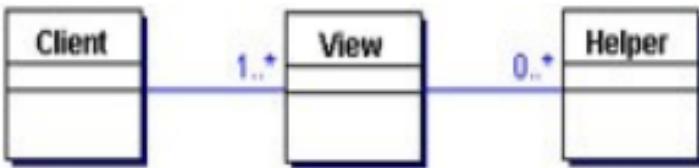
- ⌚ Filters can only be added or removed programmatically
- ⌚ Not possible to wrap the request and response objects

Intercepting Filter Pattern (Standard Filters)

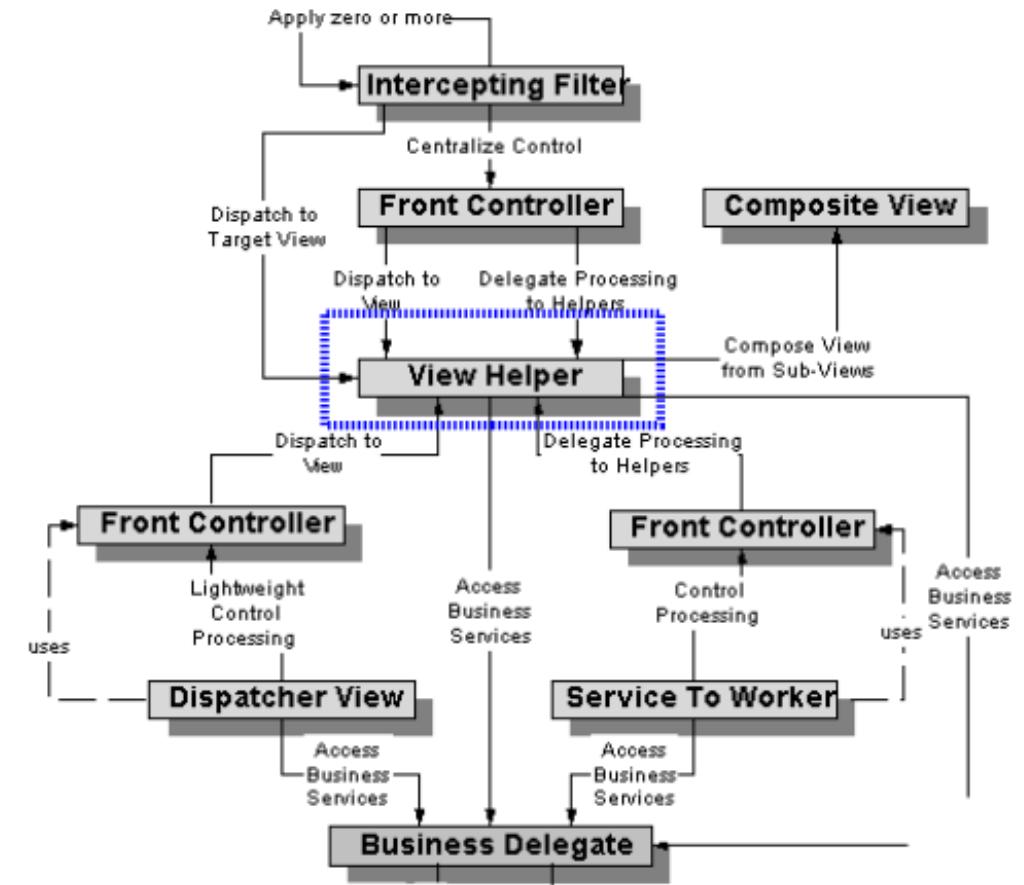
- Filters are controlled declaratively using a deployment descriptor (web.xml)

```
<filter>
    <filter-name>StandardEncodeFilter</filter-name>
    <display-name>StandardEncodeFilter</display-name>
    <description></description>
    <filter-class> corepatterns.filters.encodefilter.
        StandardEncodeFilter</filter-class>
</filter>
<filter>
    <filter-name>MultipartEncodeFilter</filter-name>
    <display-name>MultipartEncodeFilter</display-name>
    <description></description>
    <filter-class>corepatterns.filters.encodefilter.
        MultipartEncodeFilter</filter-class>
    <init-param>
        <param-name>UploadFolder</param-name>
        <param-value>/home/files</param-value>
    </init-param>
</filter>
.
.
.
<filter-mapping>
    <filter-name>StandardEncodeFilter</filter-name>
    <url-pattern>/EncodeTestServlet</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>MultipartEncodeFilter</filter-name>
    <url-pattern>/EncodeTestServlet</url-pattern>
</filter-mapping>
```

View Helper



View Helper Pattern



The system creates presentation content, which requires processing of dynamic business data

View Helper – why we need it?

- ⌚ Presentation tier changes occur often and are difficult to develop and maintain when business data access logic and presentation formatting logic are interwoven
- ⌚ This makes the system less flexible, less reusable

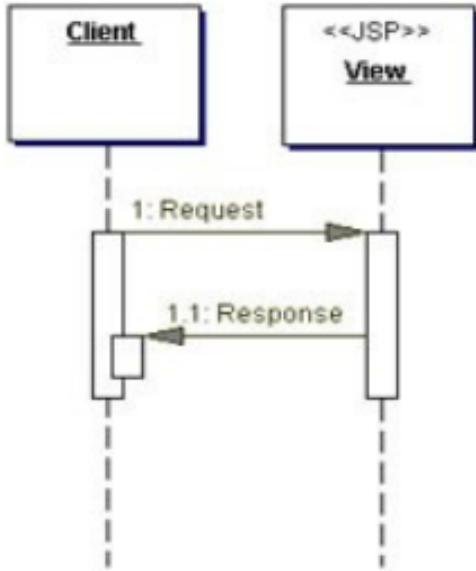
Hence,

- ⌚ Encapsulating business logic in a helper instead of a view makes our application more modular and facilitates component reuse

(Examples: Java Beans, JSP Tags, etc)

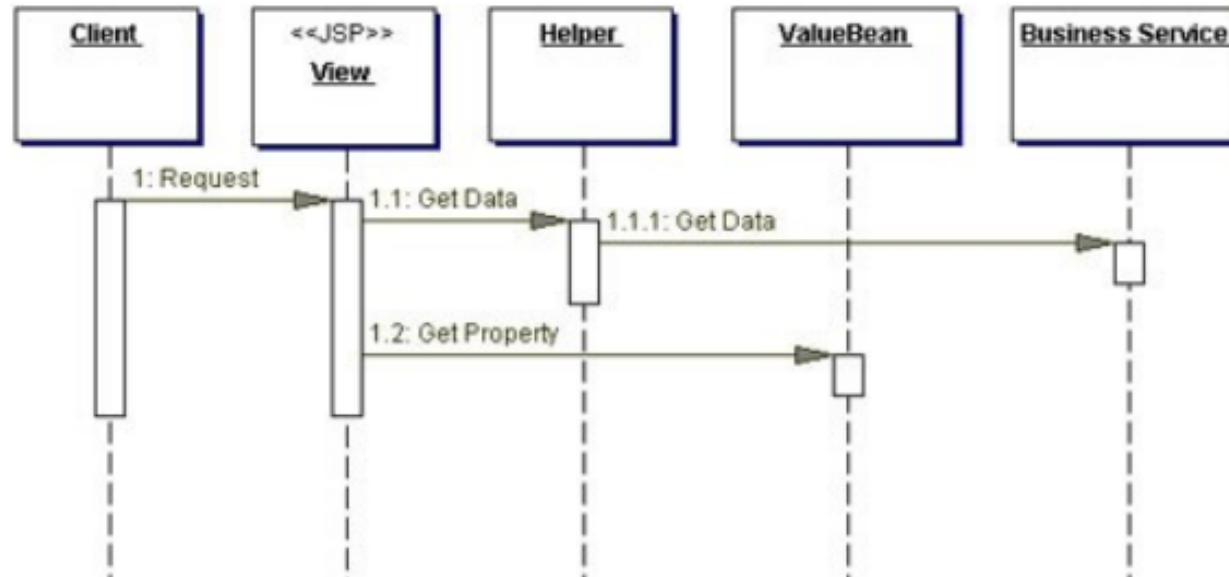
View Helper

The Simple Sequence Diagram



There are **no Helpers** here.

The page may be entirely static or with a little bit of scriptlets



View Helper in J2EE

The Servlet View Strategy

Considers the Servlet as the View

```
public class Controller extends HttpServlet {  
    public void init(ServletConfig config) throws  
        ServletException {  
        super.init(config);  
    }  
  
    public void destroy() { }  
  
    /** Processes requests for both HTTP  
     * <code>GET</code> and <code>POST</code> methods.  
     * @param request servlet request  
     * @param response servlet response  
     */  
    protected void processRequest(HttpServletRequest  
        request, HttpServletResponse response)  
        throws ServletException, java.io.IOException {  
        String title = "Servlet View Strategy";  
        try {  
            response.setContentType("text/html");  
            java.io.PrintWriter out = response.getWriter();  
            out.println("<html><title>" + title + "</title>");  
            out.println("<body>");  
            out.println("<h2><center>Employees List</h2>");  
            EmployeeDelegate delegate =  
                new EmployeeDelegate();  
  
            /** ApplicationResources provides a simple API  
             * for retrieving constants and other  
             * preconfigured values**/  
            Iterator employees = delegate.getEmployees(  
                ApplicationResources.getInstance().  
                ...);  
            ...  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

Not a Good Strategy

View Helper in J2EE

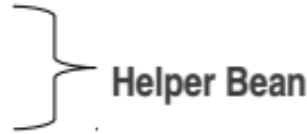
The JSP/ Java Bean Helper View Strategy

Considers the Java Beans as the View Helpers to separate the business process logic

```
<jsp:useBean id="welcomeHelper" scope="request"
    class="corepatterns.util.WelcomeHelper" />
}
<% if (welcomeHelper.nameExists())
{
%>
<center><H3> Welcome <b>
<jsp:getProperty name="welcomeHelper" property="name" />
</b><br><br> </H3></center>
<%
}
%>

<H4><center>Glad you are visiting our
site!</center></H4>

</BODY>
</HTML>
```



A Good Strategy

The Custom-Tag Helper Strategy

```
<%@ taglib uri="/web-INF/corepatternstaglibrary.tld"
   prefix="corepatterns" %>
<html>
<head><title>Employee List</title></head>
<body>

<div align="center">
<h3> List of employees in <corepatterns:department
   attribute="id"/> department - Using Custom Tag
   Helper Strategy. </h3>
<table border="1" >
  <tr>
    <th> First Name </th>
    <th> Last Name </th>
    <th> Designation </th>
    <th> Employee Id </th>
    <th> Tax Deductibles </th>
    <th> Performance Remarks </th>
    <th> Yearly Salary</th>
  </tr>
  <corepatterns:employeelist id="employeelist_key">
  <tr>
    <td><corepatterns:employee
      attribute="FirstName"/> </td>
    <td><corepatterns:employee
      attribute= "LastName"/></td>
    <td><corepatterns:employee
      attribute= "Designation"/> </td>
    <td><corepatterns:employee
      attribute= "Id"/></td>
    <td><corepatterns:employee
      attribute= "NoOfDeductibles"/></td>
    <td><corepatterns:employee
      attribute= "PerformanceRemarks"/></td>
    <td><corepatterns:employee
      attribute= "YearlySalary"/></td>
    <td>
    </tr>
  </corepatterns:employeelist>
</table>
</div>
</body>
</html>
```



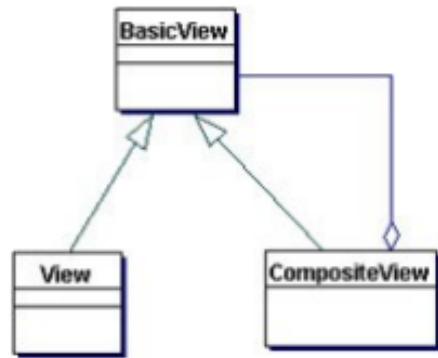
The Helper is implemented
as a Custom Tag

Requires more effort
to develop the Tag
Library

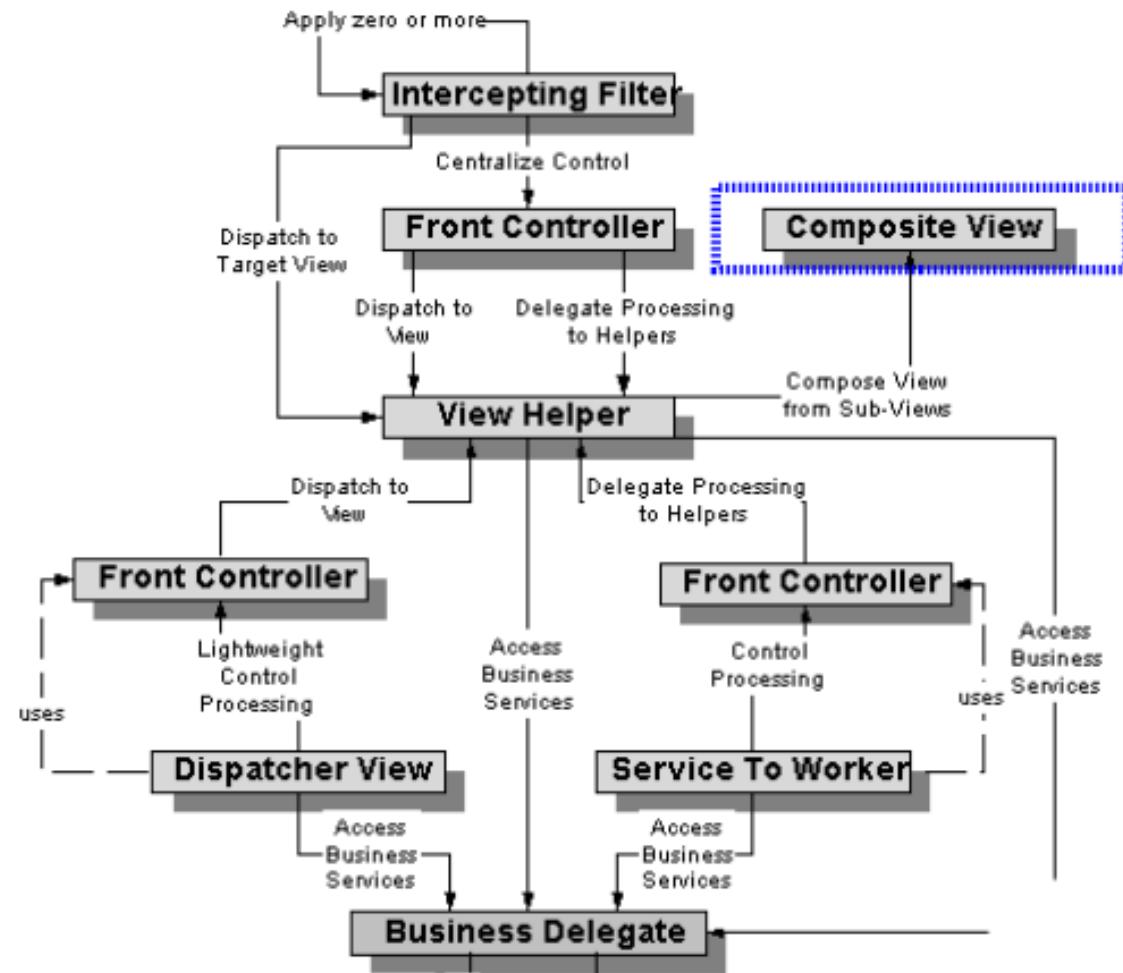
Good if you have more
common business logic

Composite View Pattern

Composite View

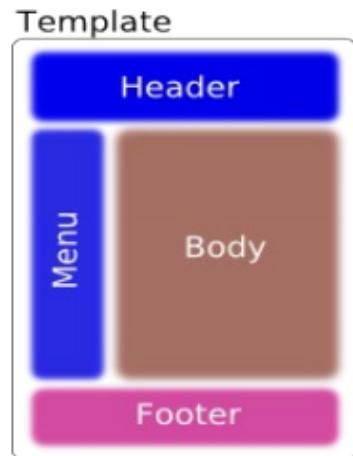


Use composite views that are composed of multiple atomic sub-views. Each component of the template may be included dynamically into the whole and the layout of the page may be managed independently of the content



Composite View Pattern

- This allows to create pages that have a similar structure, in which each section of the page vary in different situations

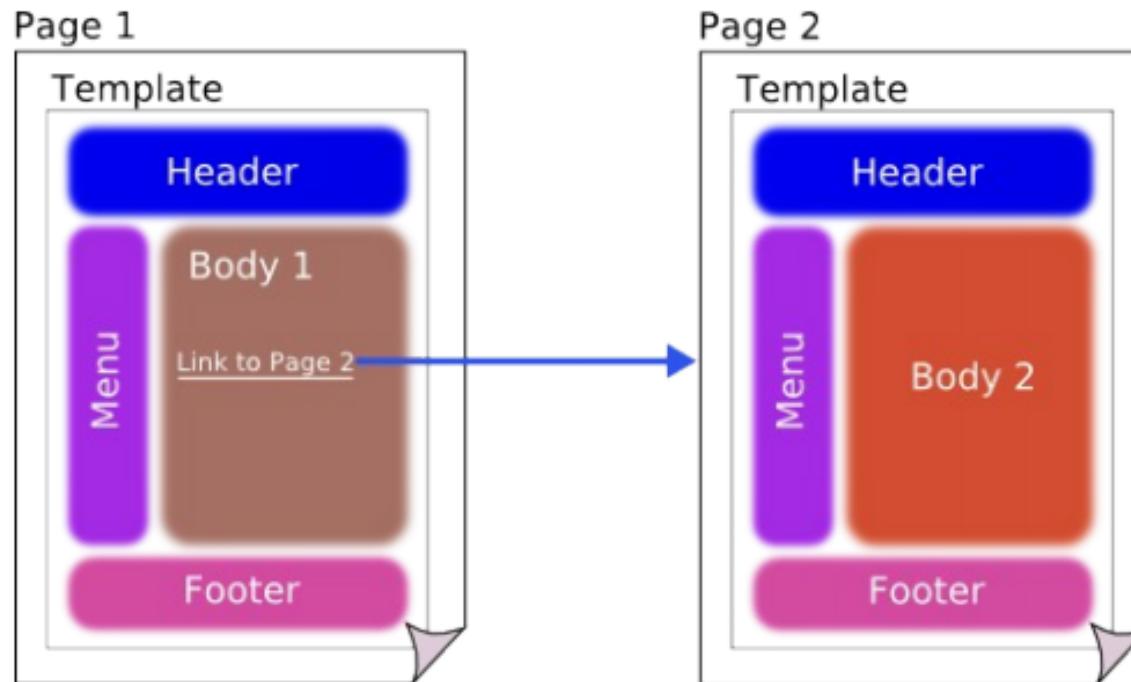


The Classic Page Layout

Composite View Pattern

Composite View is an Aggregation of multiple views

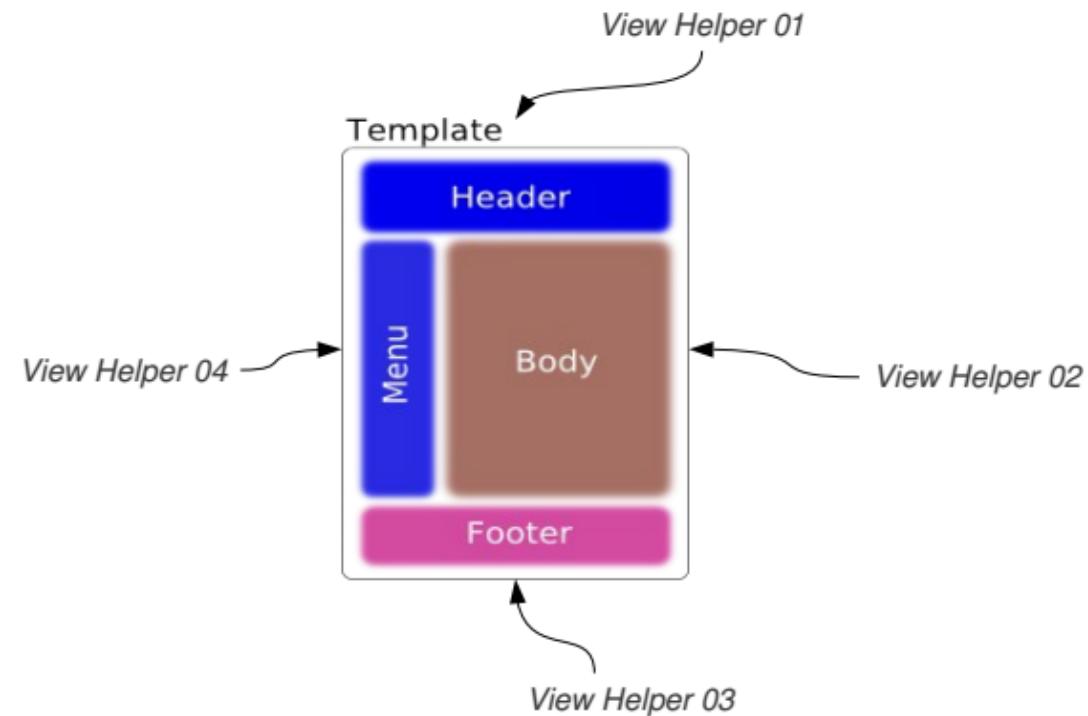
Composite View Pattern



Only the "Body" part is changed. Rest of the layout is preserved.
However the all pages related to the template are distinct.

Composite View Pattern

Each piece of the composed page can have a "view helper"

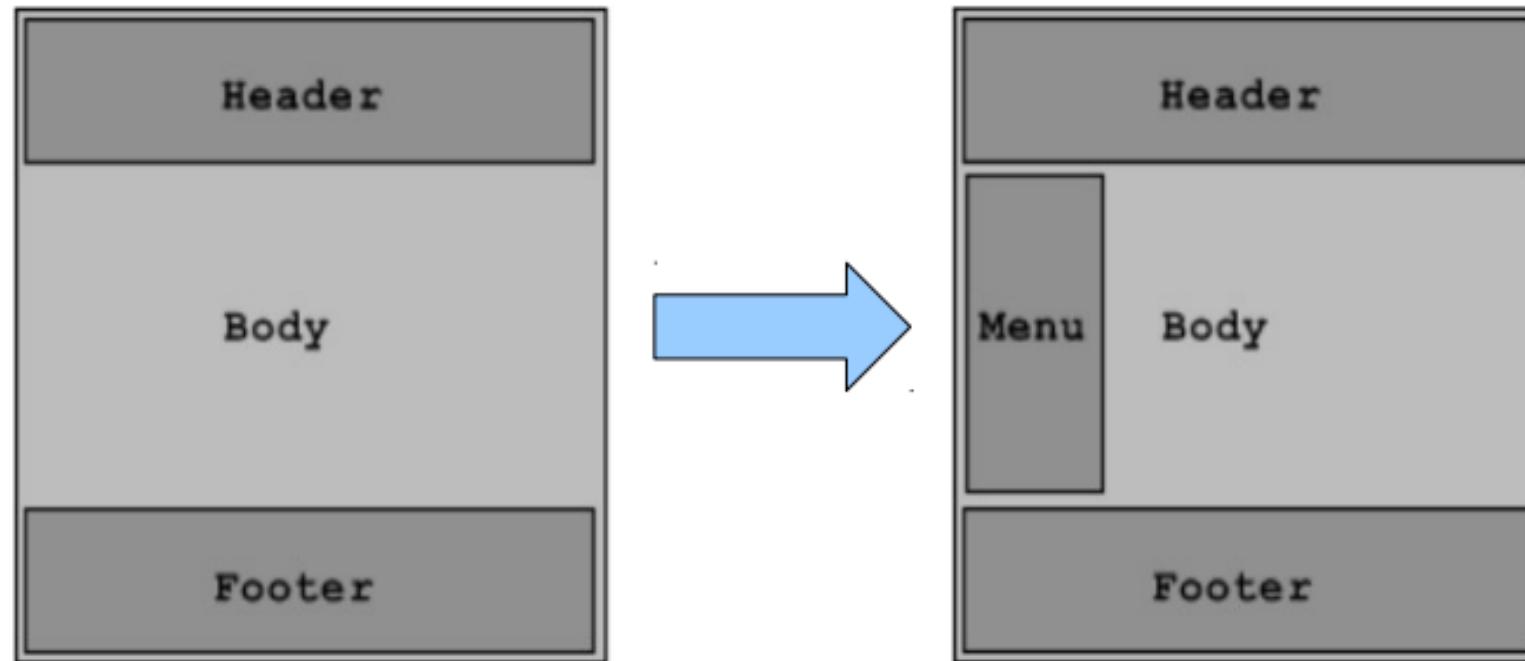


Composite View Pattern

Tiles Framework (<http://tiles.apache.org>)



What if you want to change the web application page layout like below?



Composite View Pattern

Tiles Framework (<http://tiles.apache.org>)



- Tiles uses a separate layout file
- When the layout of the application is changed, only this layout file and other tiles configuration files need to be changed

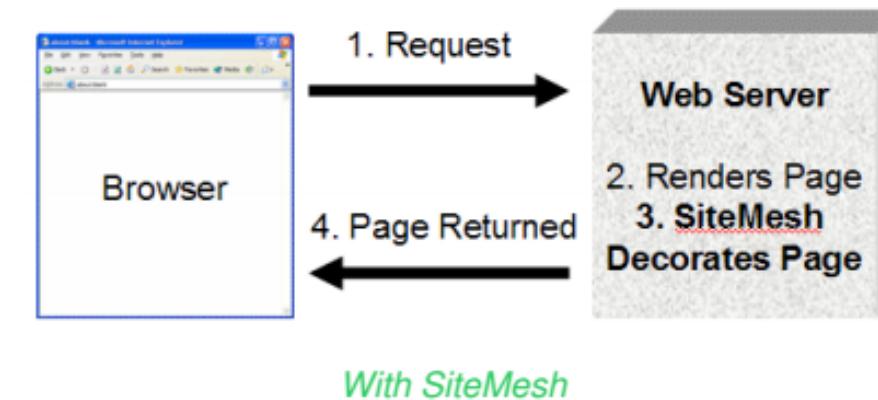
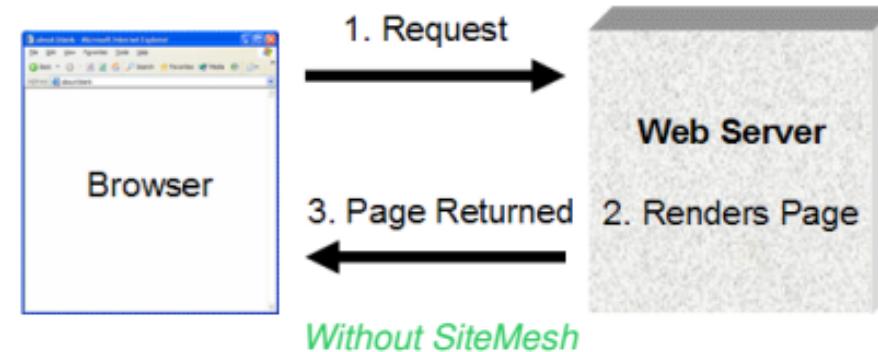
```
<%@ page language="java"%>

<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-tiles" prefix="tiles" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html locale="true">
  <head>
    <html:base />
    <title><tiles:getAsString name="title" /></title>
  </head>
  <body>
    <table border="1" width="600" cellspacing="5">
      <tbody>
        <tr><td colspan="2"><tiles:insert attribute="header" /></td></tr>
        <tr>
          <td width="200"><tiles:insert attribute="navigation" /></td>
          <td width="400"><tiles:insert attribute="body" /></td>
        </tr>
        <tr><td colspan="2"><tiles:insert attribute="footer" /></td></tr>
      </tbody>
    </table>
  </body>
</html:html>
```

SiteMesh Framework

<http://www.opensymphony.com/sitemesh/>

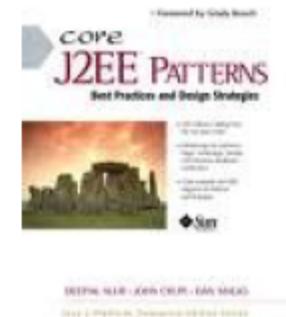


A Recap

- ⌚ **Requirement:** I need one place of control for handling all requests
 - ⌚ Pattern: [Front Controller Intercepting Filter](#)
- ⌚ **Requirement:** I need a generic command interface for delegating processing from a controller to various helper components
 - ⌚ Pattern: [Front Controller](#)
- ⌚ **Requirement:** I want to make sure data related to my presentation formatting logic is encapsulated correctly
 - ⌚ Pattern: [View Helper](#)
- ⌚ **Requirement:** I need to be able to create one View from a number of sub-Views
 - ⌚ Pattern: [Composite View](#)

Enterprise Application Blueprints

- ➊ These are well-defined design patterns geared towards a specific technology
 - ➋ Sun's J2EE Blueprint for Java
(Reference: Core J2EE Patterns Book)
 - ➋ .NET Blueprint for C# (.NET Pet Store example)



References

- ⌚ Patterns of Enterprise Application Architecture (PoEAA): *Martin Fowler* (<http://martinfowler.com/eaaCatalog/>)
- ⌚ J2EE Design Patterns: *William Crawford & Jonathan Kaplan*
- ⌚ Core J2EE Patterns: *Deepak Alur, John Crupi, Dan Malks*
- ⌚ <http://www.developer.com/design/article.php/3619786/Implementing-the-Intercepting-Filter-Pattern-in-Your-Enterprise-Java-Applications.htm>
- ⌚ <http://msdn.microsoft.com/en-us/library/ms998516.aspx>
- ⌚ http://struts.apache.org/1.x/userGuide/building_controller.html
- ⌚ <http://java.sys-con.com/node/36656>



Business Layer Patterns

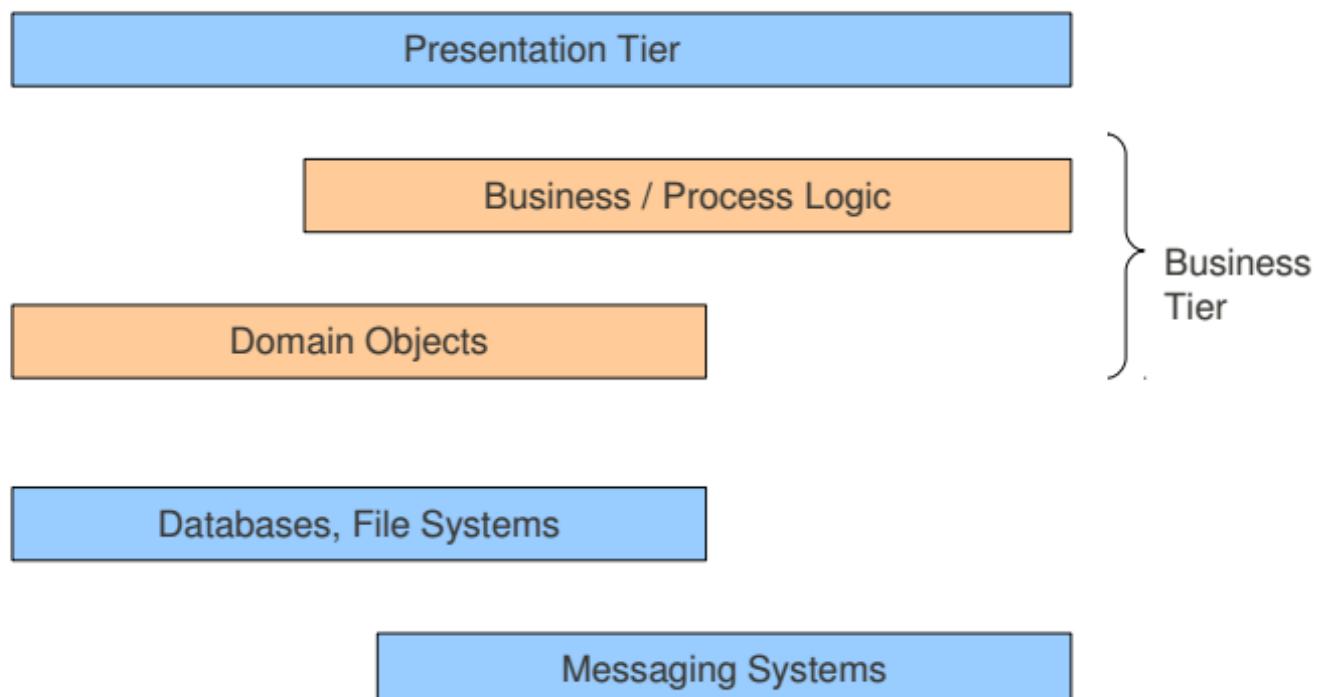
Lecture 04

by Udara Samaratunge

Three Tiers



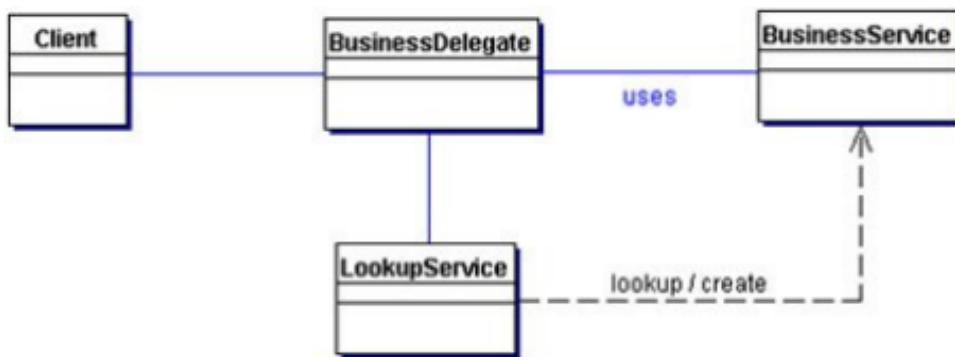
Business Tier



Domain / Business Layer Design Patterns

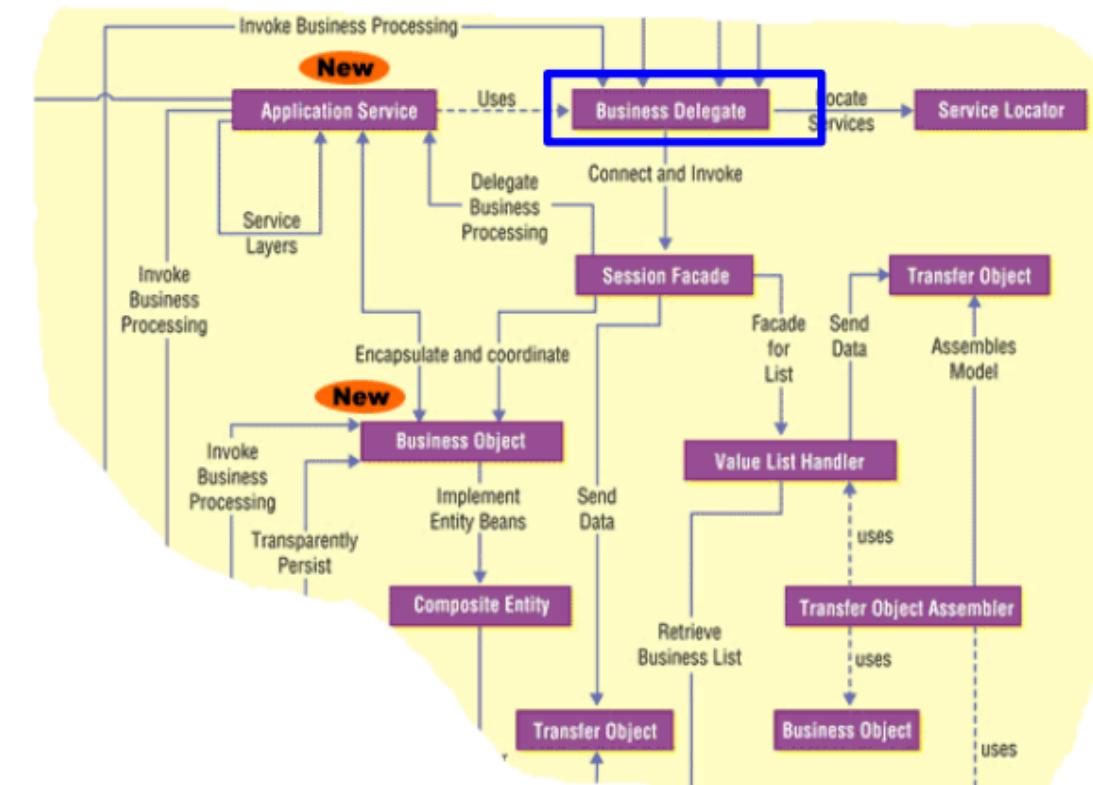
- ⌚ Business Delegate
- ⌚ Service Locator
- ⌚ Session Facade
- ⌚ Transfer Object
- ⌚ Value List Handler
- ⌚ Composite Entity
- ⌚ Transfer Object Assembler

Business Delegate Pattern



The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of business services

Business Delegate Pattern



Business Delegate Pattern

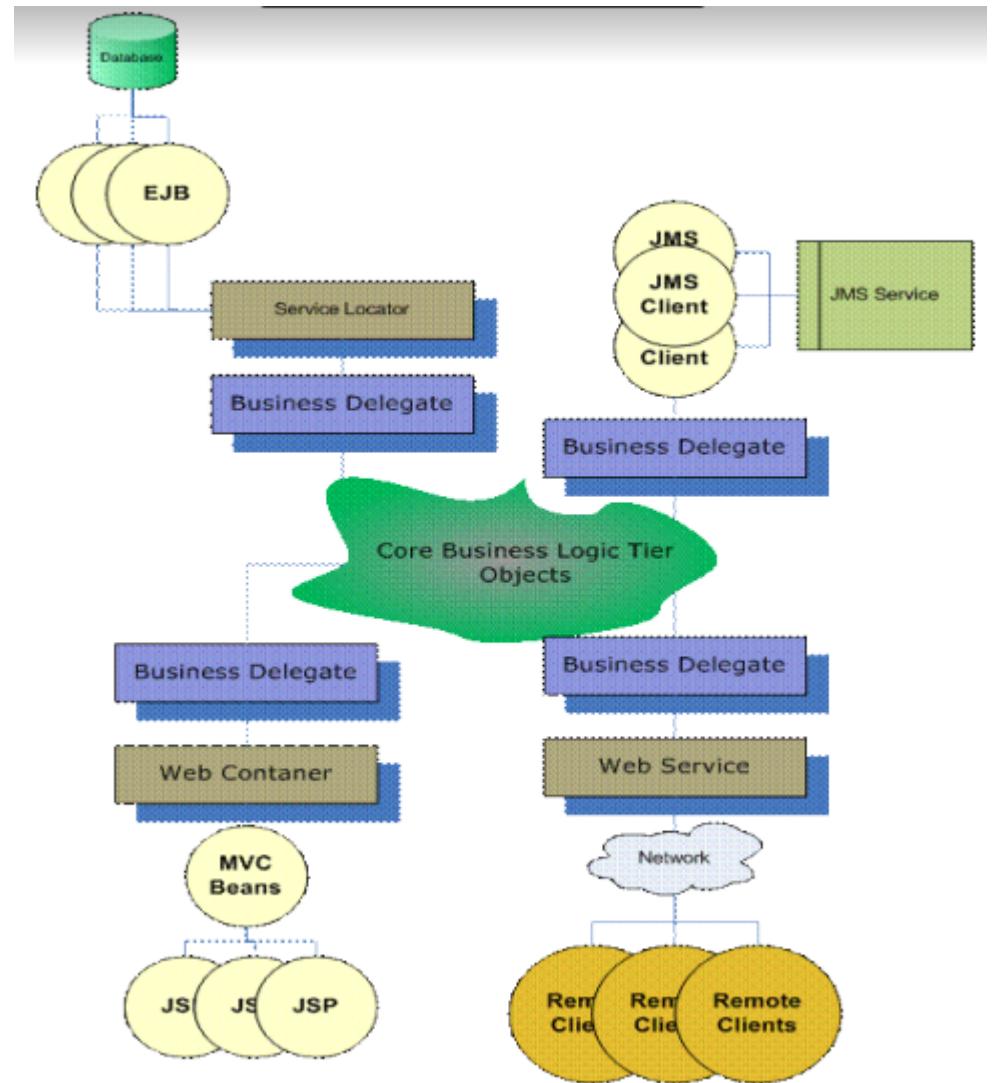
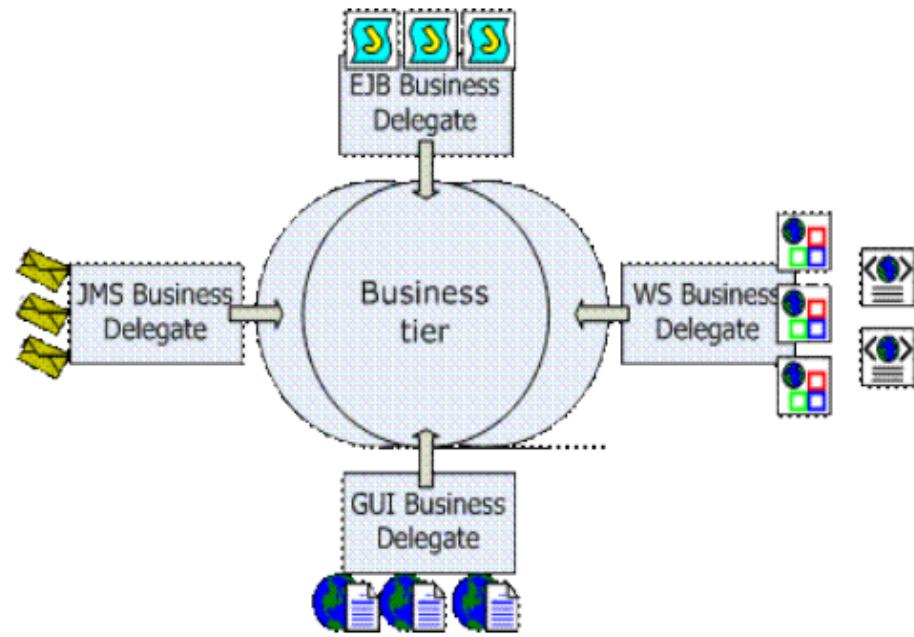
- ⦿ There can be a lot of invocations to Business Service APIs from the presentation layer. That can create a heavy network traffic



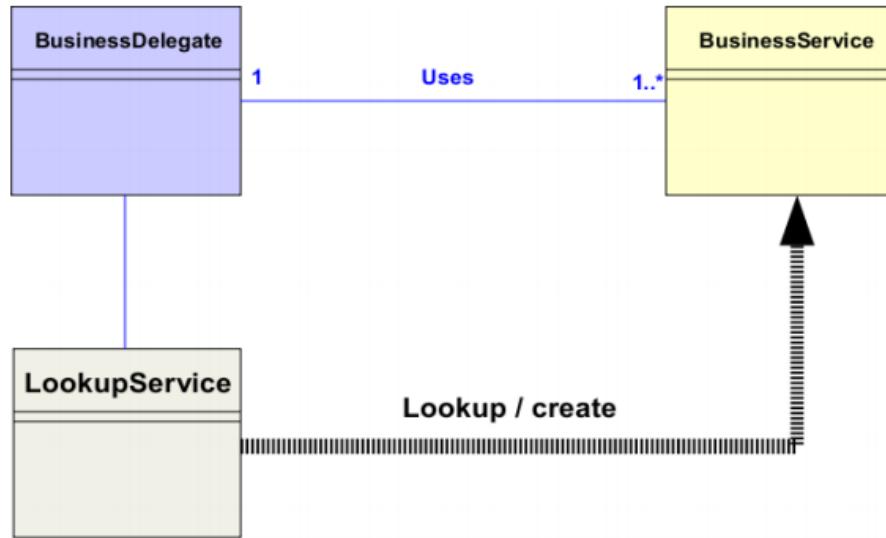
Without this pattern..

- ⦿ Presentation-tier components interact directly with business services
- ⦿ This direct interaction exposes the underlying implementation details of the business service application program interface (API) to the presentation tier
- ⦿ As a result, the presentation-tier components are vulnerable to changes in the implementation of the business services (**If the business services changed, the presentation tier must be changed too**)

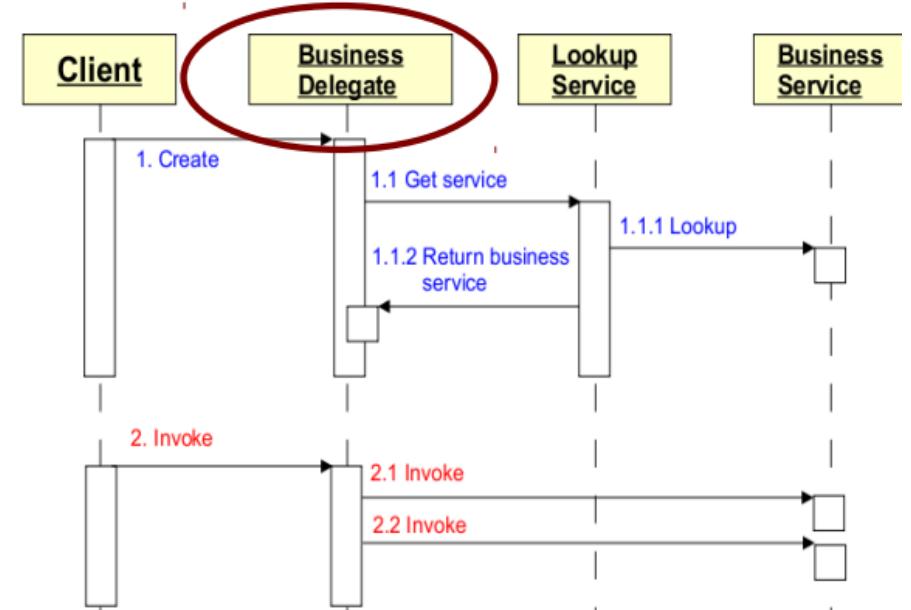
Business Delegate Pattern



Business Delegate Pattern



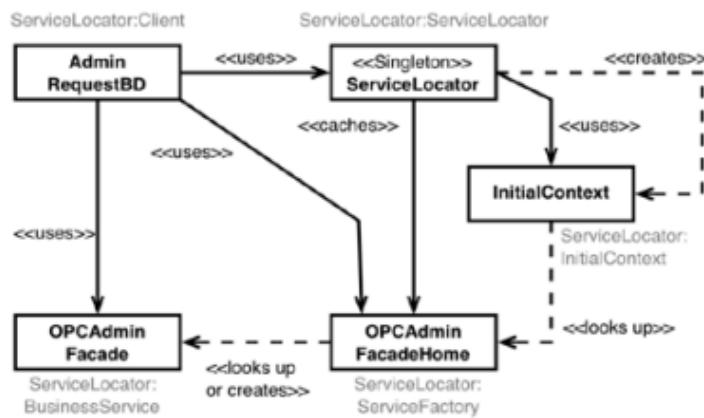
Business Delegate Pattern



Business Delegate Pattern

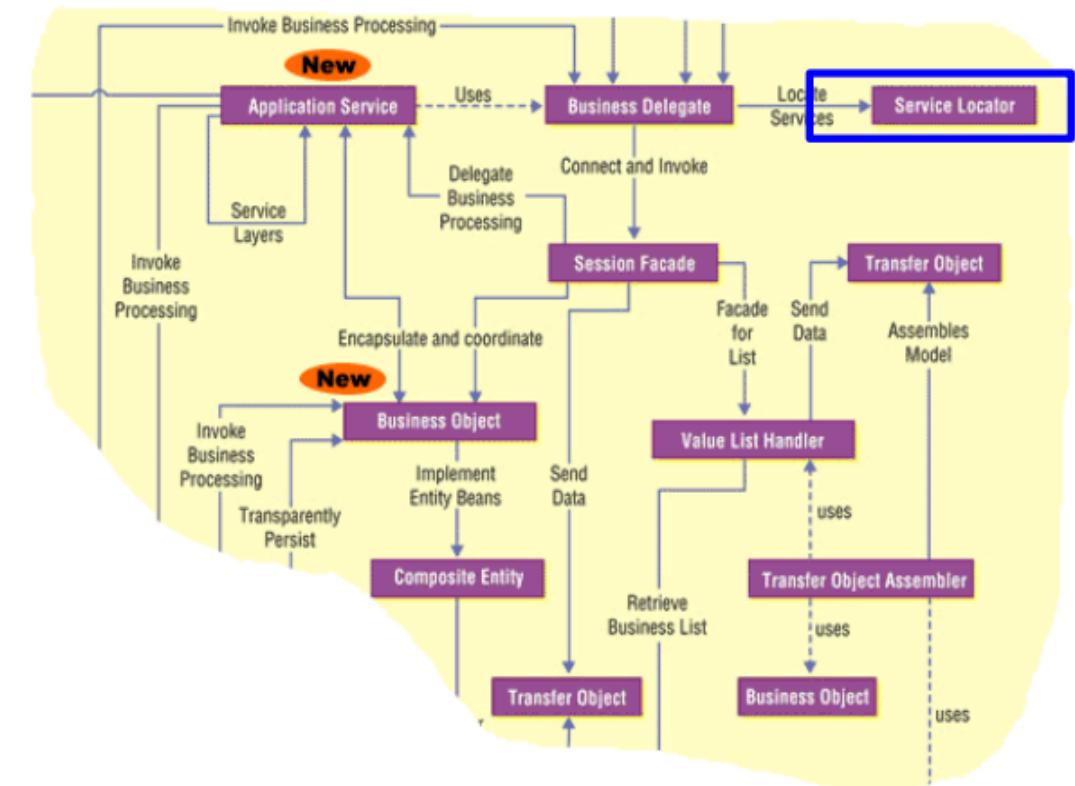
- ➊ The **Business Delegate** uses a **Lookup Service** for locating the business service
- ➋ The **Business Service** is used to invoke the business methods on behalf of the client
- ➌ The role of this pattern is to provide control and protection for the business service

Service Locator Pattern



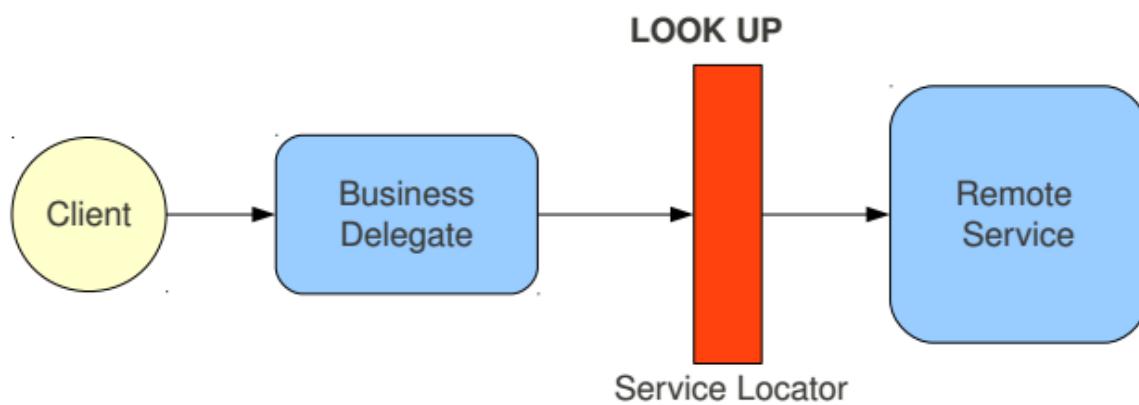
Use a Service Locator to implement and encapsulate service and component lookup. A Service Locator hides the implementation details of the lookup mechanism and encapsulates related dependencies

Service Locator Pattern



Service Locator Pattern

- Enterprise applications require a way to look up the service objects that provide access to distributed components

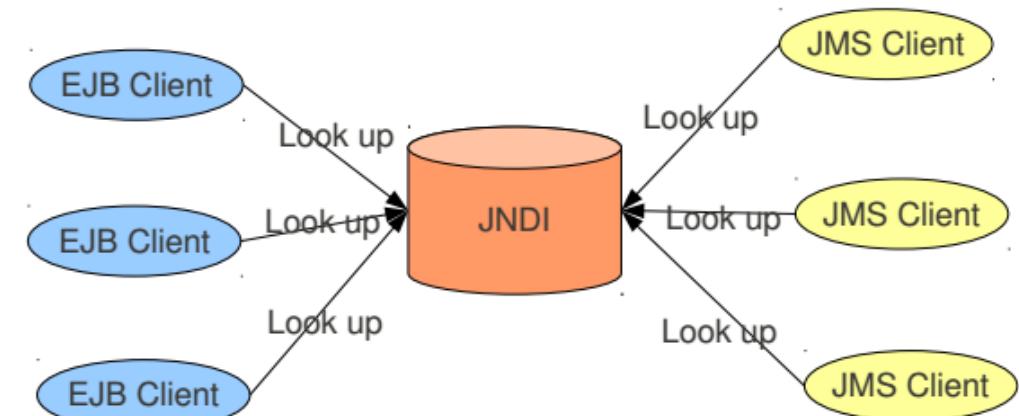


Service Locator Pattern

Problem

Without the Service Locator, (In J2EE)

- J2EE specification mandates the usage of JNDI (Java Naming and Directory Interface) to access different resources/services

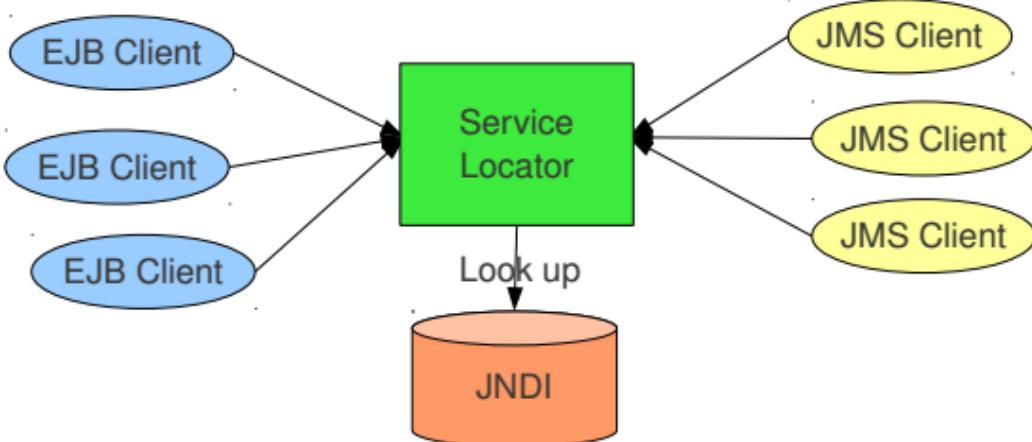


Service Locator Pattern

Solution

Using the Service Locator, (In J2EE)

- The solution is to cache those service objects when the client performs JNDI lookup first time and reuse that service object from the cache second time onwards for other clients.



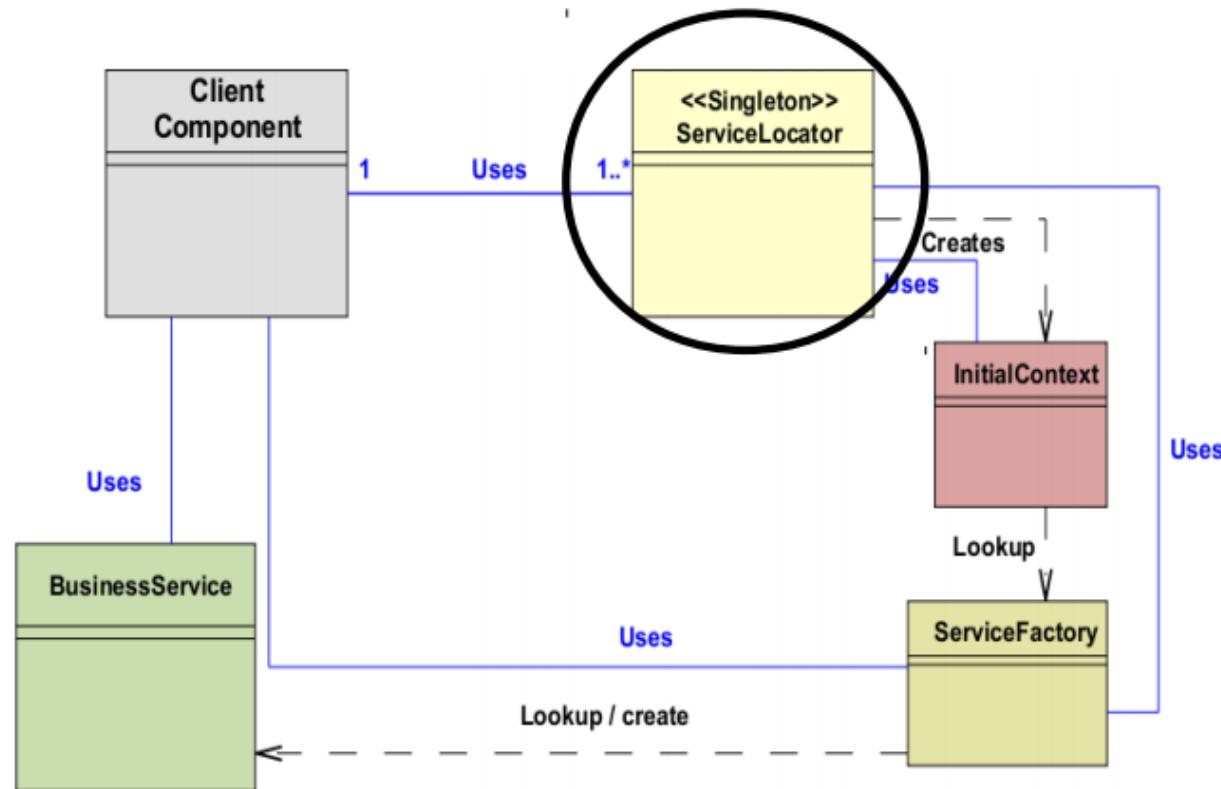
Service Locator Pattern

Solution

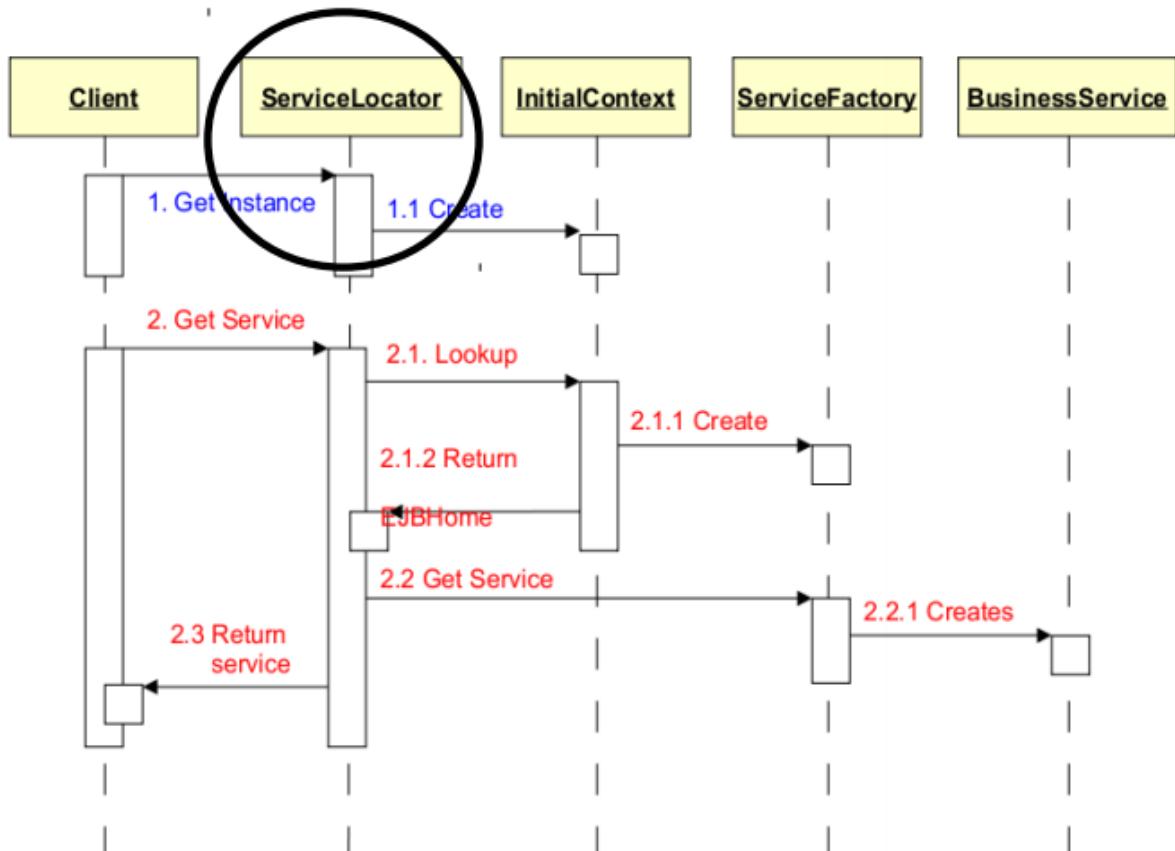
Using the Service Locator, (In J2EE) cont..

- Here the ServiceLocator class intercepting the client request and accessing JNDI once and only once for a service object.
- Here the clients call ServiceLocator class to get a service object rather than calling JNDI directly. ServiceLocator acts as interceptor between client and JNDI.

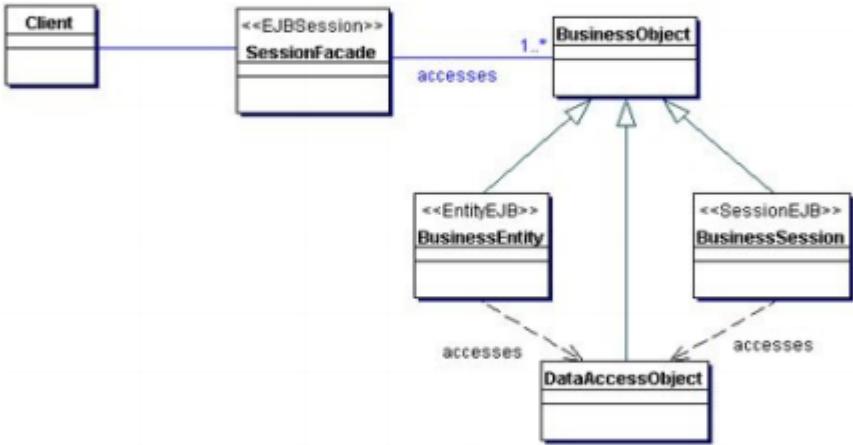
Service Locator Pattern



Service Locator Pattern

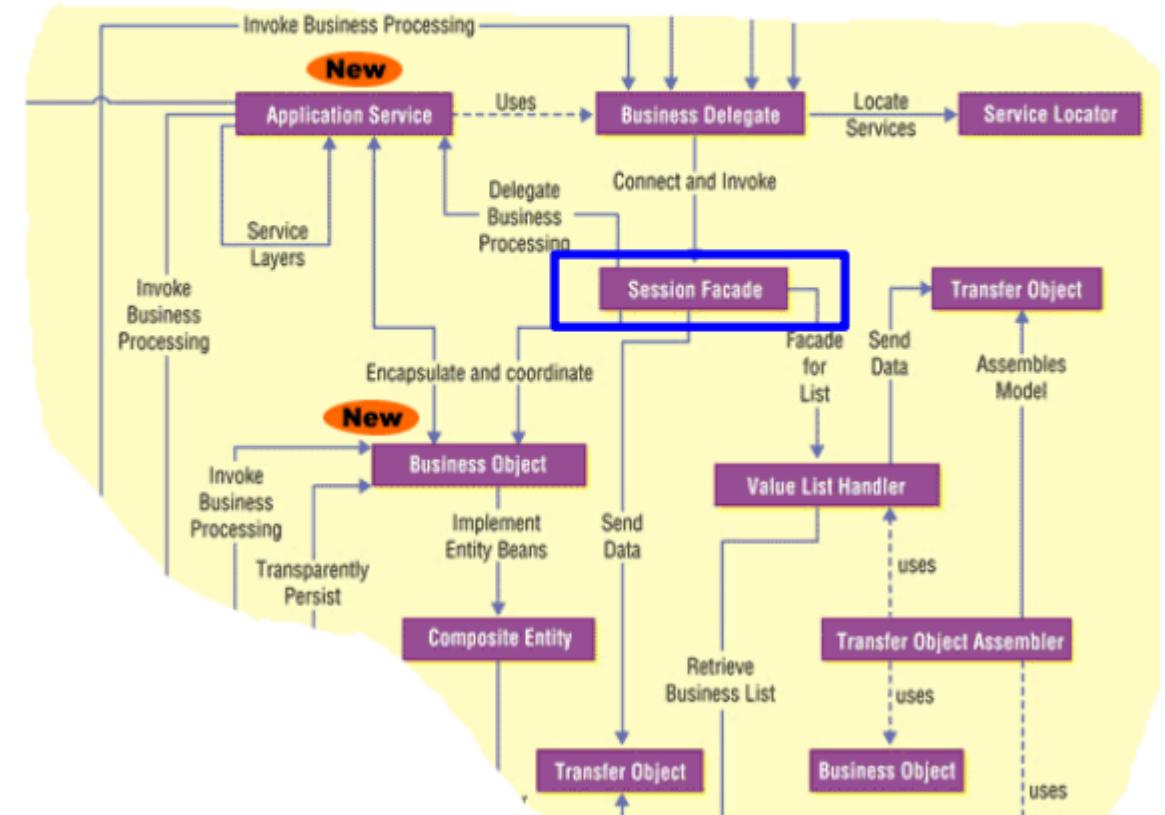


Session Facade Pattern

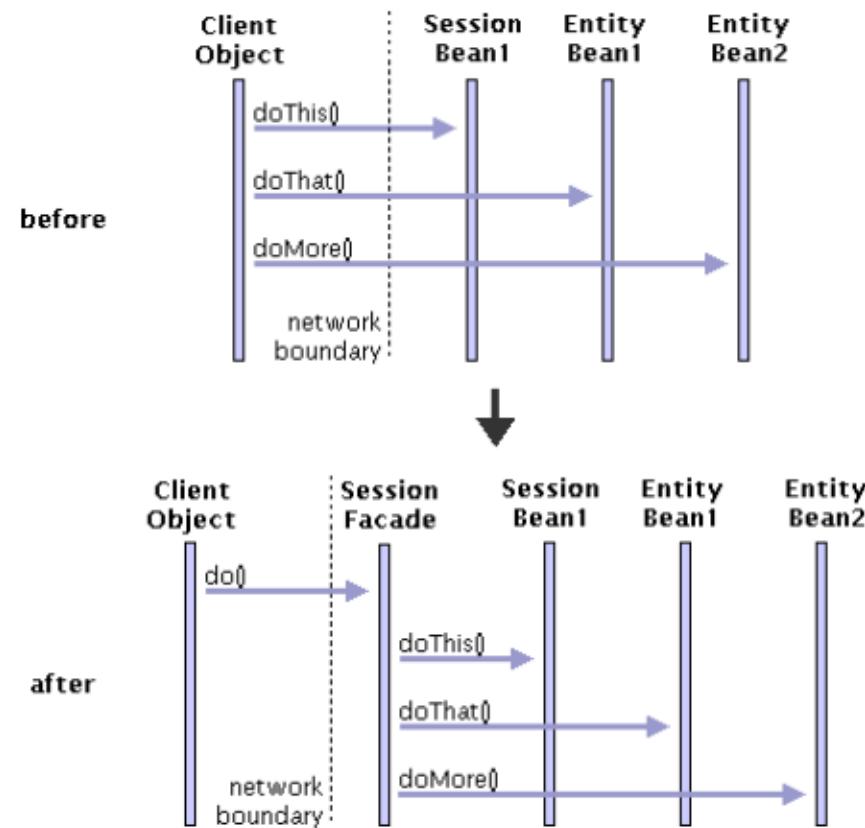


The Session Facade manages the business objects, and provides a uniform coarse-grained service access layer to clients

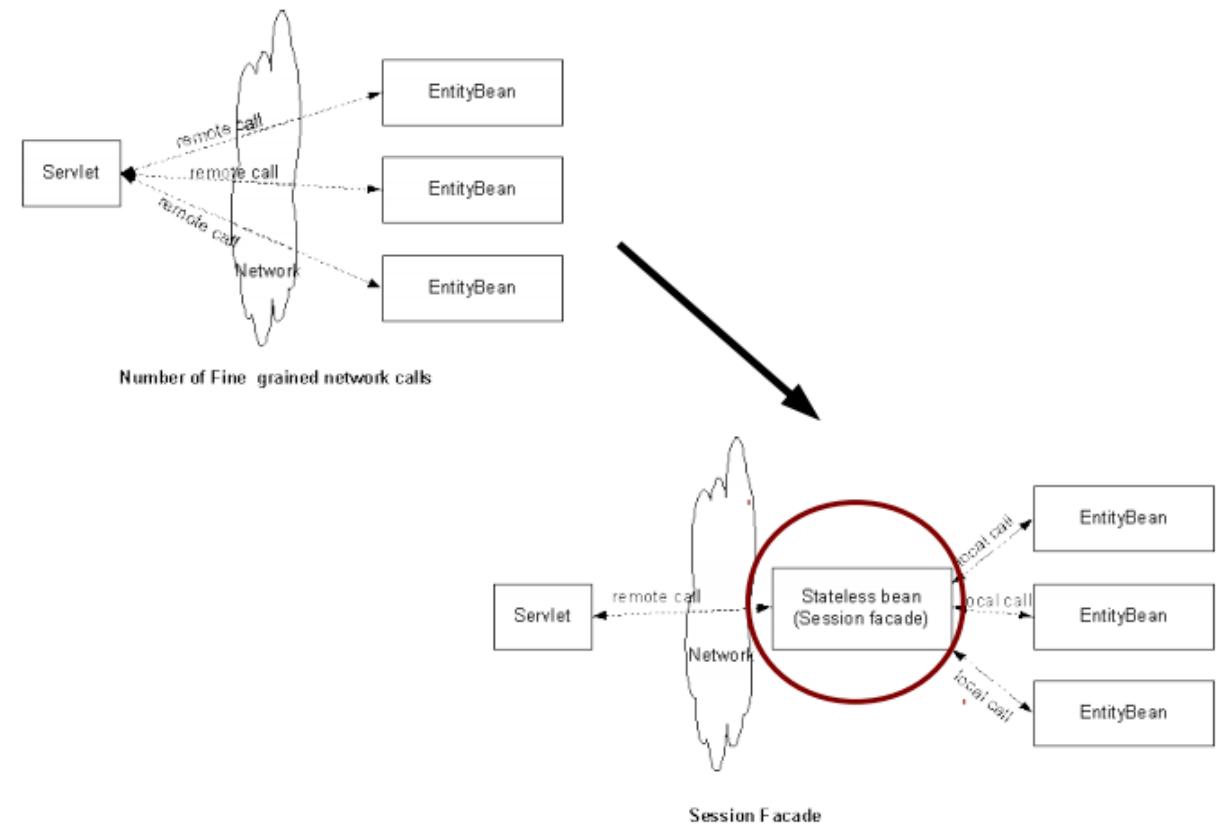
Session Facade Pattern



Session Facade Pattern



Session Facade Pattern



Session Facade Pattern

- ⌚ It is implemented as a higher level component (i.e.: Session EJB), and it contains all the interactions between low level components (i.e.: Entity EJB).
- ⌚ Provides a single interface for the functionality of an application or part of it
- ⌚ It decouples lower level components simplifying the design

Session Facade Pattern

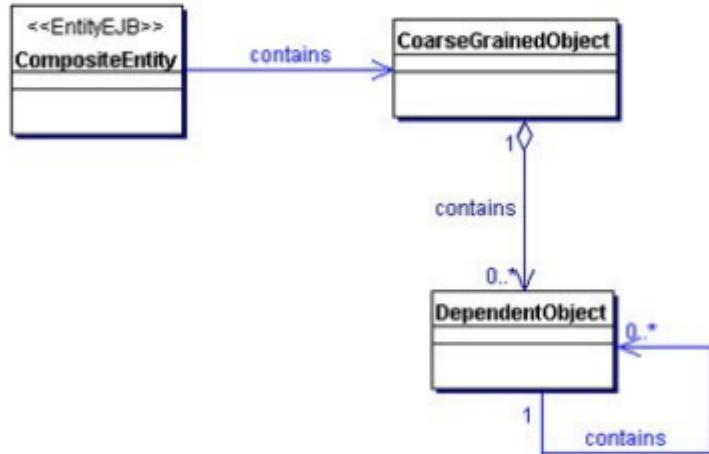
⌚ Benefits

- ⌚ Acts as a business-tier controller layer
- ⌚ Exposes a uniform interface
 - Reduces the complexity of underlying business components
- ⌚ Reduces coupling, introduce manageability
- ⌚ Improves performance, reduces fine-grained methods
- ⌚ Provides coarse grained access
 - Analyze the interaction between the client and the application services, using use cases and scenarios to determine the coarseness of the facade

Domain Object Model

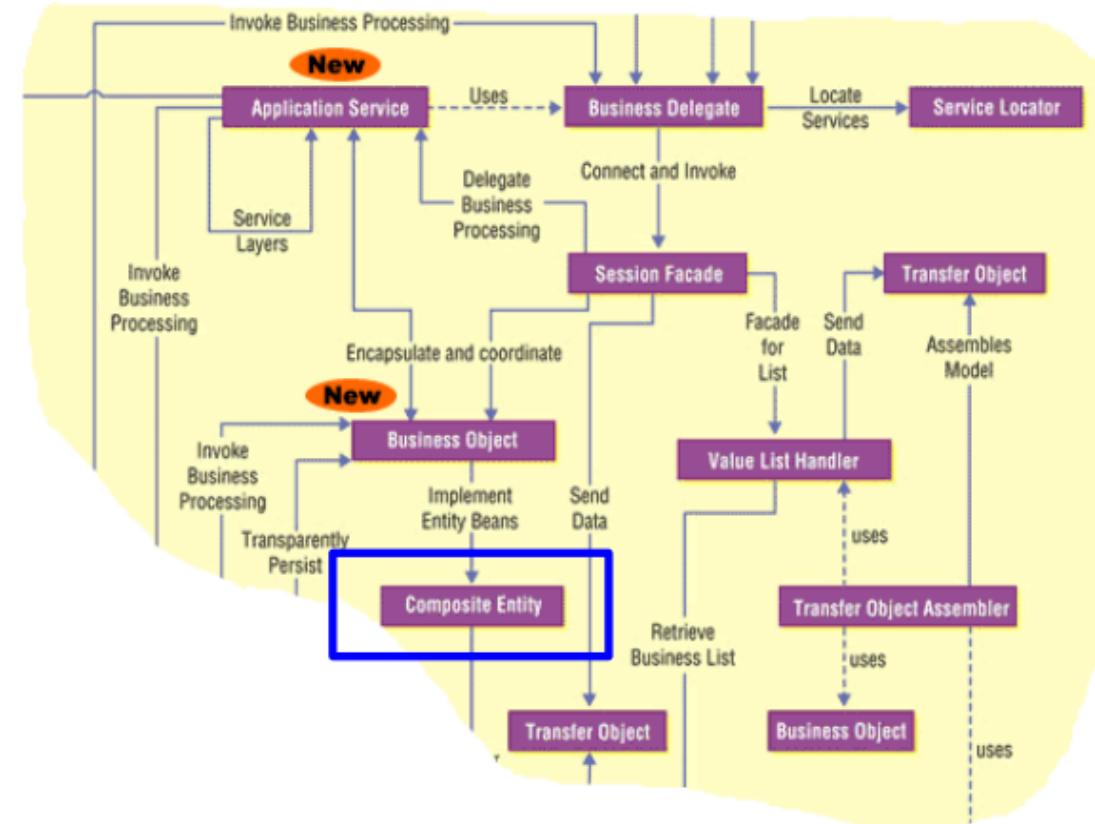
- ➊ The Domain Object Model defines an object-oriented representation of the underlying data of an application.
 - ➋ It is a portion of the business tier that defines the actual data and processes managed and implemented by the system
-
- ➌ **Purpose:**
 - ➍ The business tier becomes more simpler and are decoupled from the implementation of the business tier.
 - ➎ Allows changes to the resource layer level implementation without affecting the UI components.
 - ➏ The presentation tier can access those plain objects instead of manipulating the database directly
 - ➐ In Java,
 - Two types of domain object models.
 - Enterprise Java Beans (EJBs)
 - POJOs (Plain Old Java Objects)
 - EJBs often make sense when you need a coarse-grained entity model, remote access, and integral transaction support

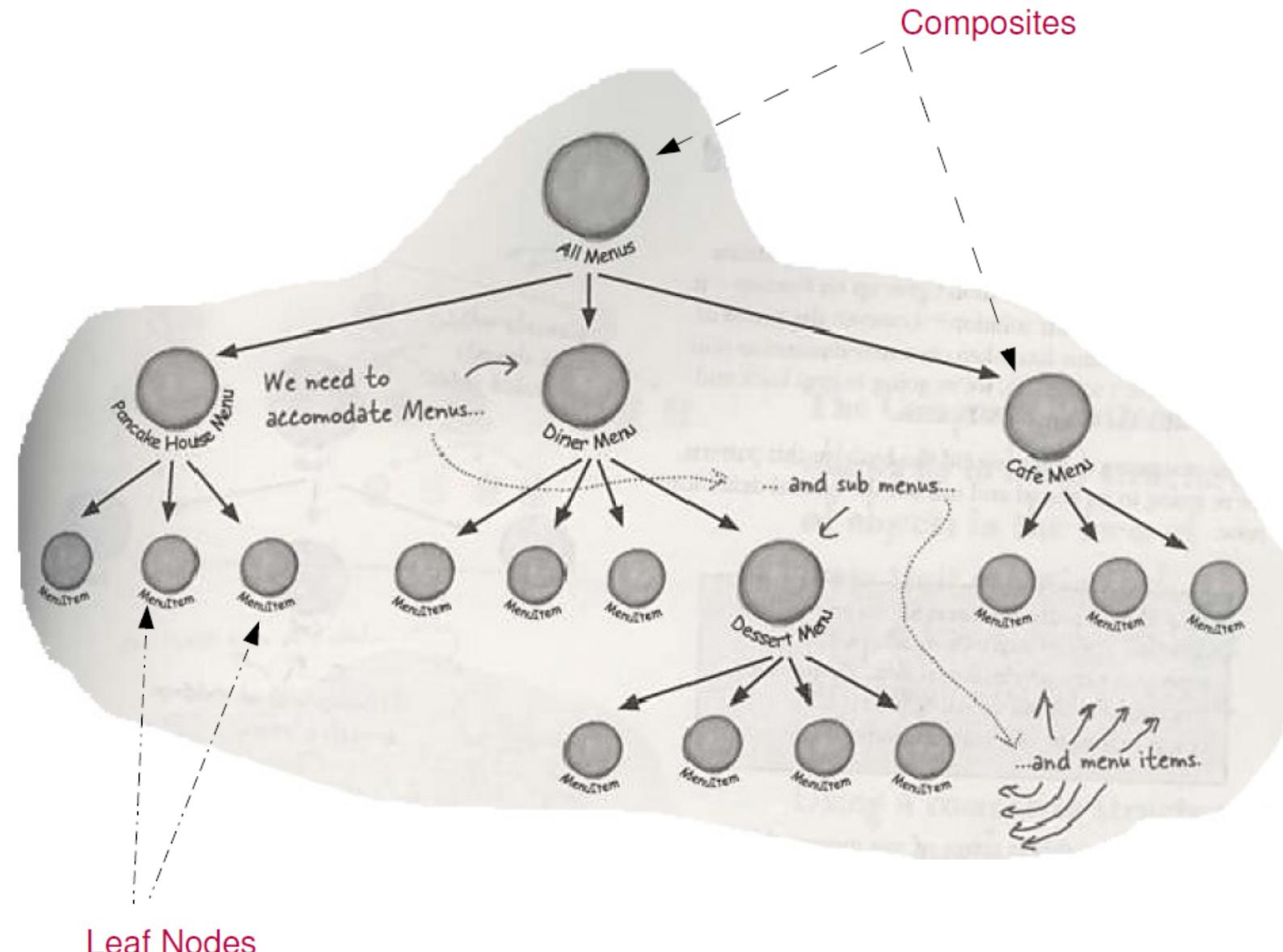
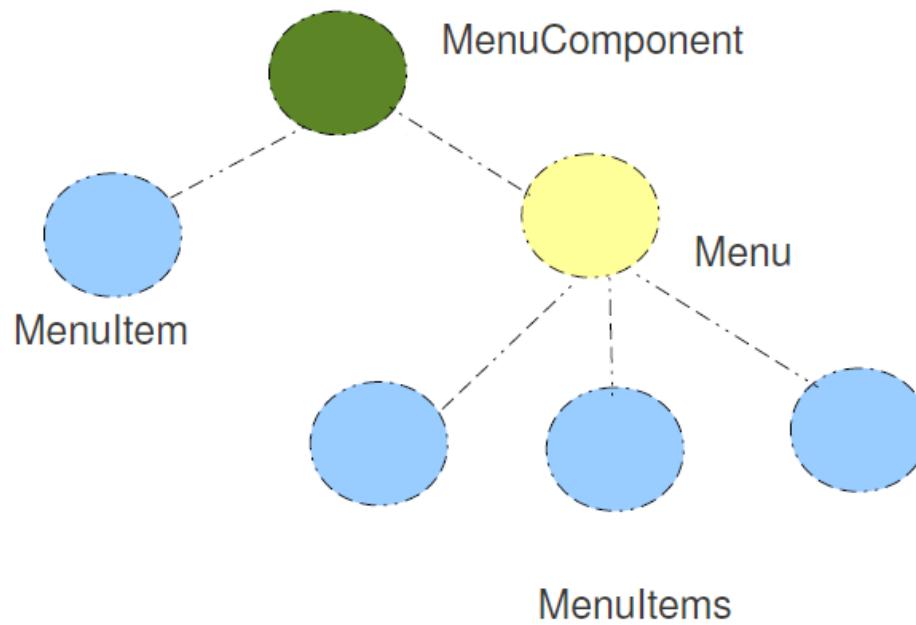
Composite Entity Pattern



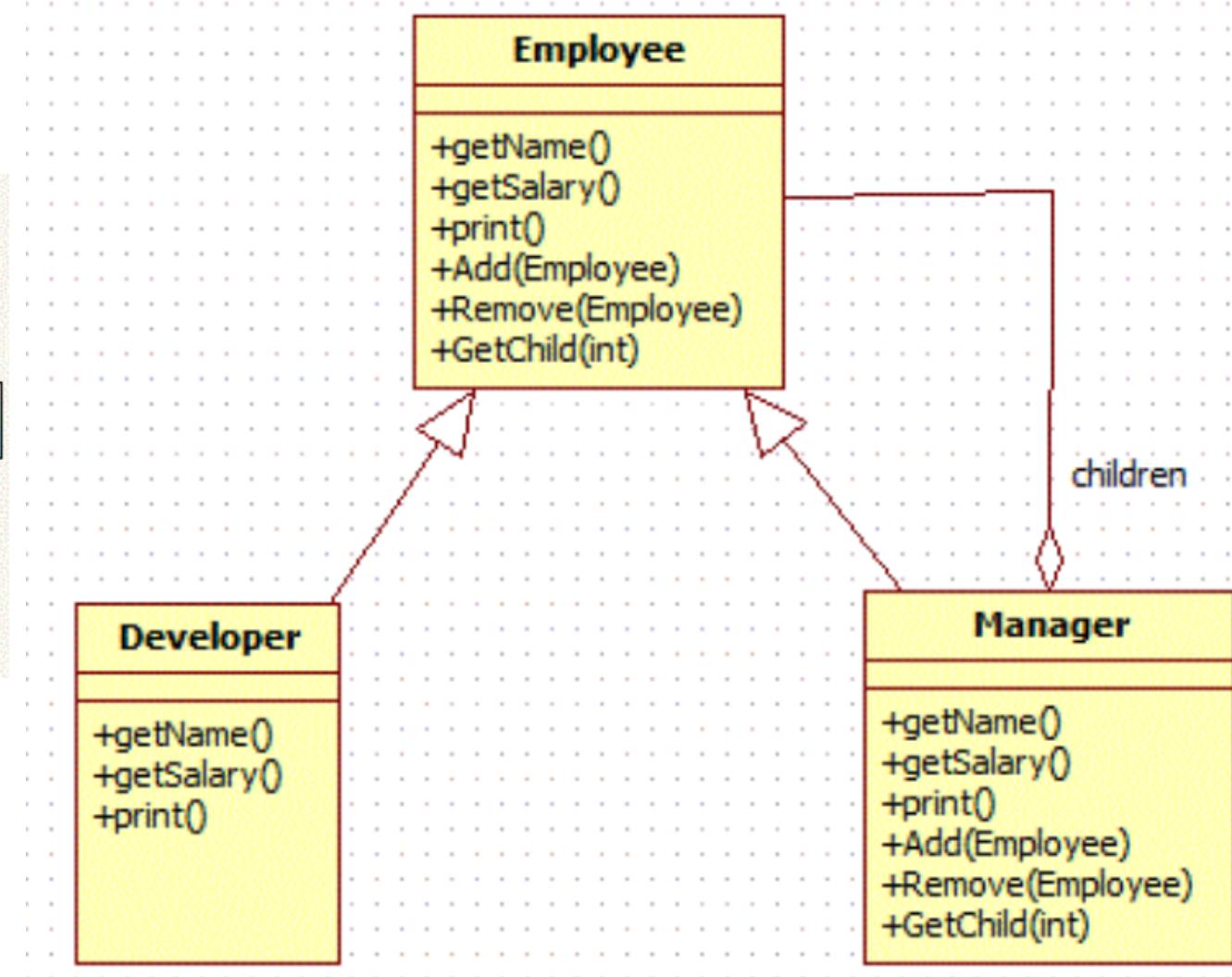
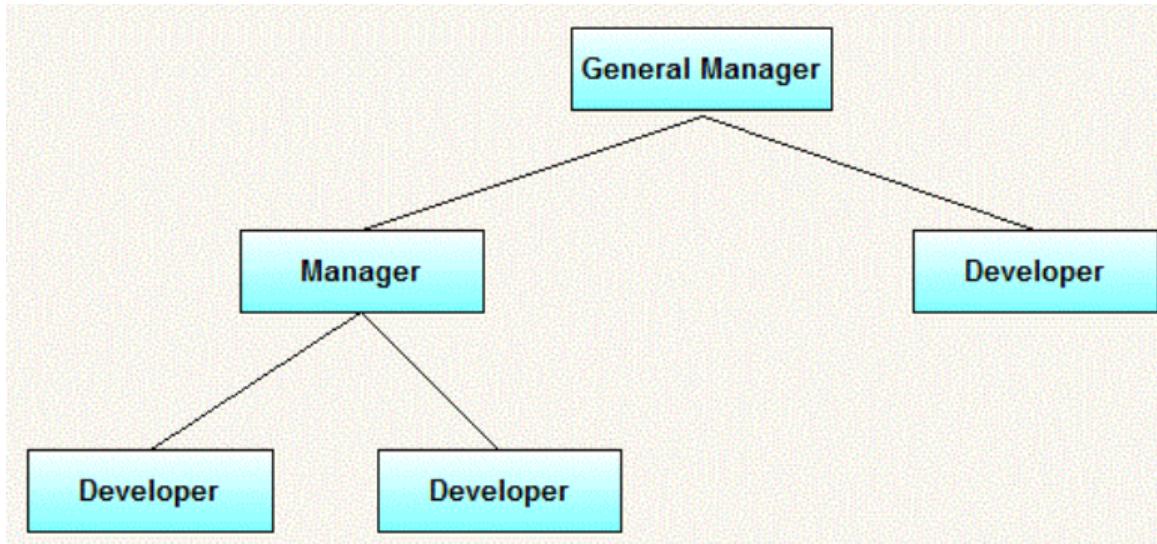
Use Composite Entity to model, represent, and manage a set of interrelated persistent objects rather than representing them as individual fine-grained objects.

Composite Entity Pattern





In-class Activity for Composite Entity Patterns



Test class for the Output

The screenshot shows an IDE interface with several tabs at the top: ymentController.java, Employee.java, CompositeDesignPatternMain.java (which is currently selected), Manager.java, Console, and Problems.

The Java code in the CompositeDesignPatternMain.java tab is as follows:

```
public class CompositeDesignPatternMain {  
    public static void main(String[] args) {  
        Employee emp1 = new Developer("John", 10000);  
        Employee emp2 = new Developer("David", 15000);  
        Employee manager1 = new Manager("Daniel", 25000);  
        manager1.add(emp1);  
        manager1.add(emp2);  
        Employee emp3 = new Developer("Michael", 20000);  
        Manager generalManager = new Manager("Mark", 50000);  
        generalManager.add(emp3);  
        generalManager.add(manager1);  
        generalManager.print();  
    }  
}
```

The output in the Console tab is:

```
<terminated> CompositeDesignPatternMain  
-----  
Name =Mark  
Salary =50000.0  
-----  
Name =Michael  
Salary =20000.0  
-----  
Name =Daniel  
Salary =25000.0  
-----  
Name =John  
Salary =10000.0  
-----  
Name =David  
Salary =15000.0  
-----
```

Composite Entity Pattern

⌚ Why this pattern?

⌚ One common mistake is mapping each table in the underlying database to a class, and each row in that table to an object instance

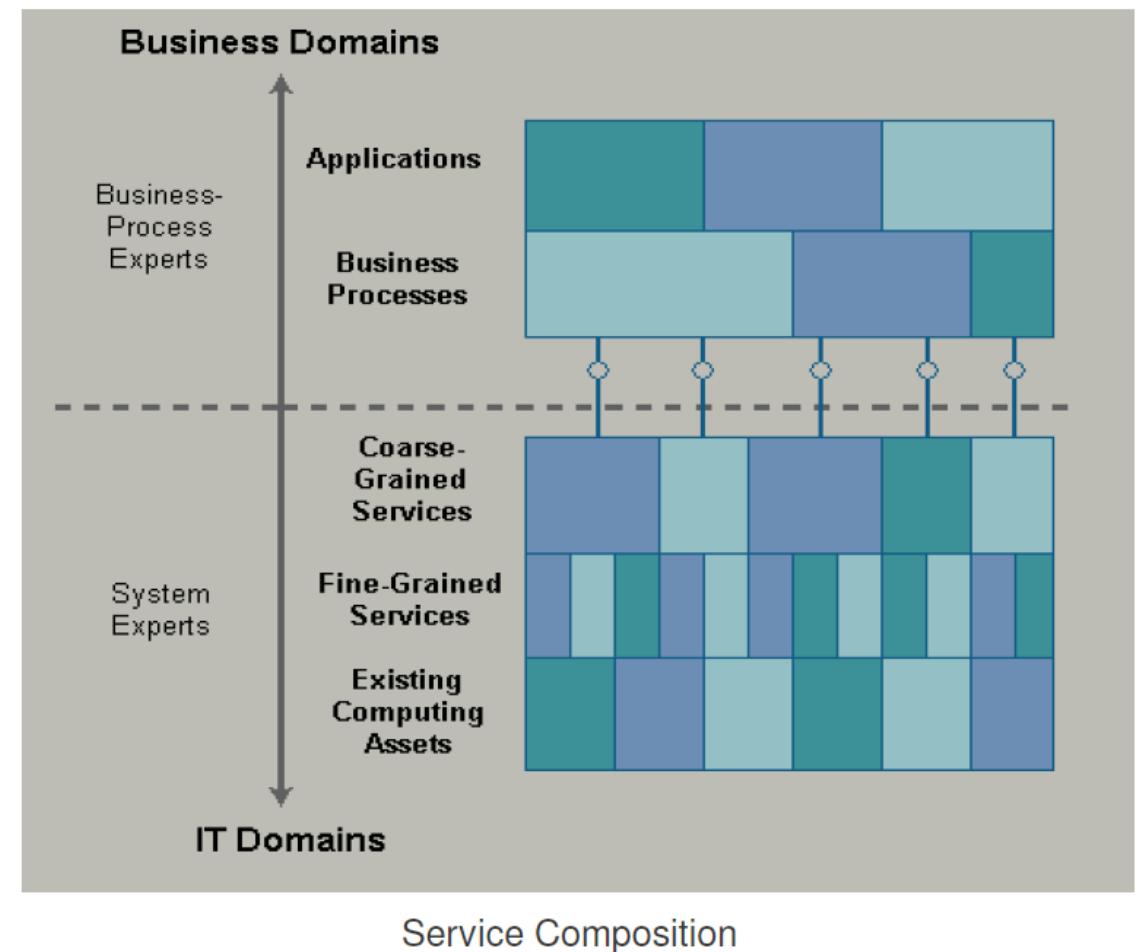
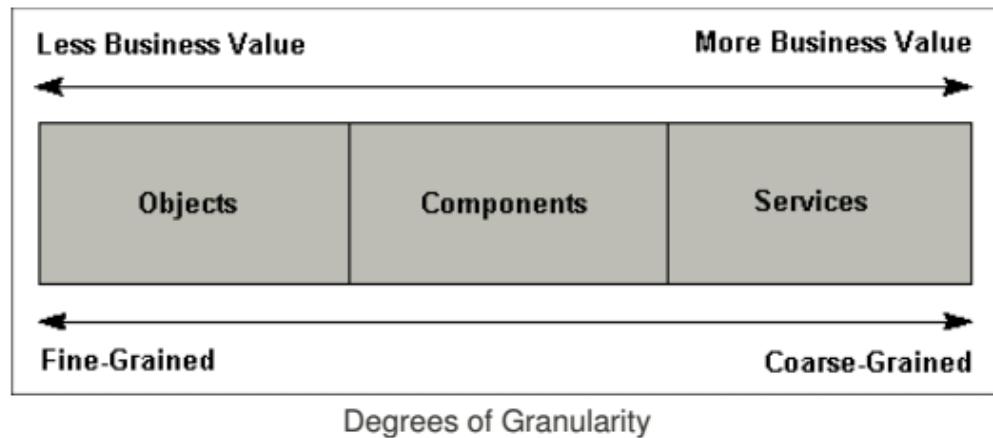
- Examples: CMP, other O/R Mapping Tools
- Issues:
 - Servers must do more to maintain the integrity of each bean
 - Communication overhead causes performance degradation

Persistence of Objects

- ⌚ A **persistent object** is an object that is stored in some type of a data store.
- ⌚ Multiple clients usually share persistent objects.
- ⌚ Persistent objects can be classified into two types:
 - ⌚ Coarse-grained objects
 - ⌚ Dependent objects

Service Composition

Degrees of Granularity (*Fine-Grained/ Coarse-Grained*)



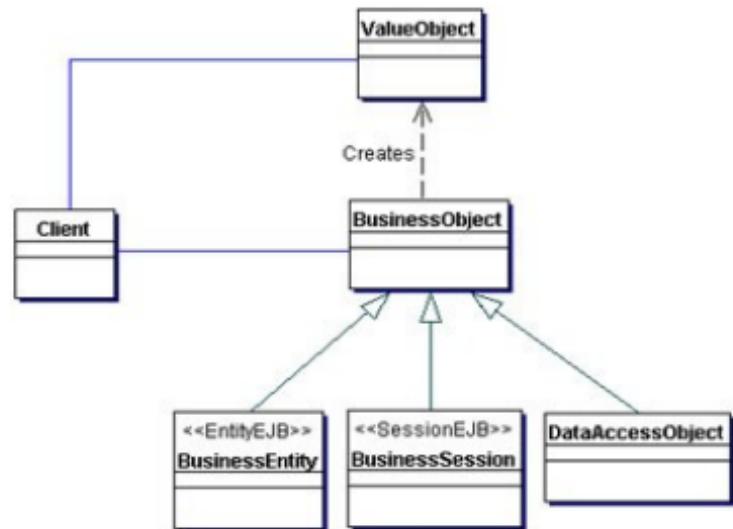
Lazy Loading

- ⌚ What is it?
- ⌚ Lazy loading is a characteristic of an application when the actual loading and instantiation of a class is delayed until the point just before the instance is actually used.
- ⌚ The goal is to only dedicate memory resources when necessary by only loading and instantiating an object at the point when it is absolutely needed .

Lazy Loading

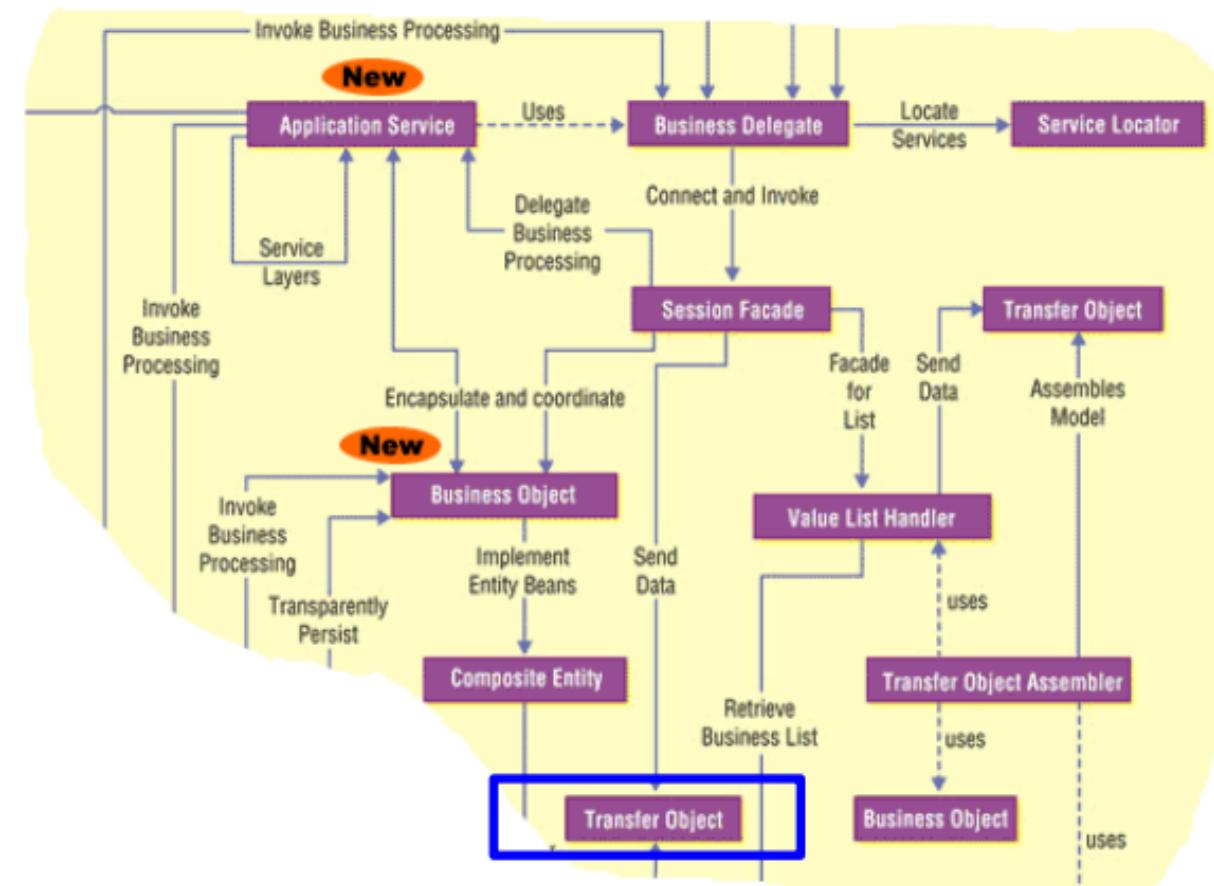
- ⌚ Loading all the dependent objects of the Composite Entity may take considerable time and resources (i.e. ejbLoad() in EJB spec)
- ⌚ One way to optimize this is by using a lazy loading strategy for loading the dependent objects
- ⌚ When at first only load those dependent objects that are most crucial to the Composite Entity clients.
- ⌚ Subsequently, when the clients access a dependent object that has not yet been loaded from the database, the Composite Entity can perform a load on demand

Transfer Object Pattern



Use a Transfer Object to encapsulate the business data. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client

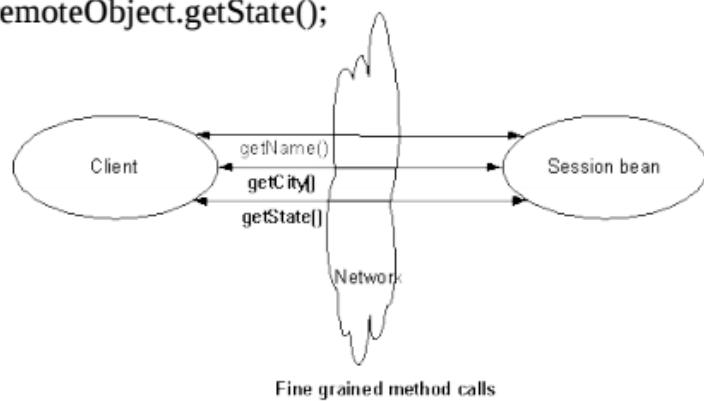
Transfer Object Pattern



Transfer Object Pattern

Problem

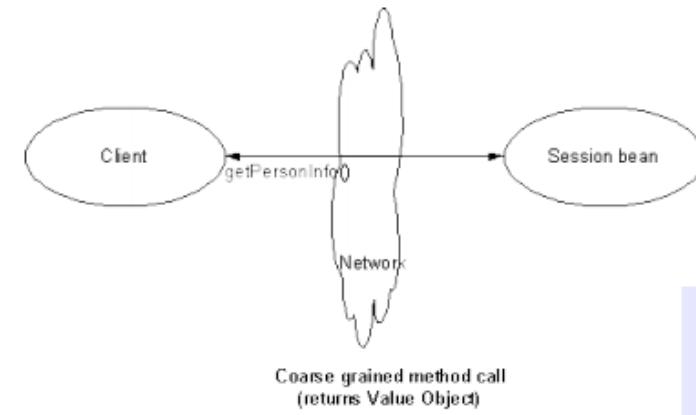
- ⌚ Need to avoid fine-grained method calls. This imposes a overhead on the network due to the number of calls
 - `remoteObject.getName(); remoteObject.getCity();
remoteObject.getState();`



Transfer Object Pattern

Solution

- ⌚ Use the coarse-grained approach



```
PersonInfo person = new PersonInfo();
person.setName("Amal");
person.setCity("Colombo");
person.setState("WP");
person.zipCode("11600");
```

```
// send Value Object through network
remoteObject.getPersonInfo(person);
```

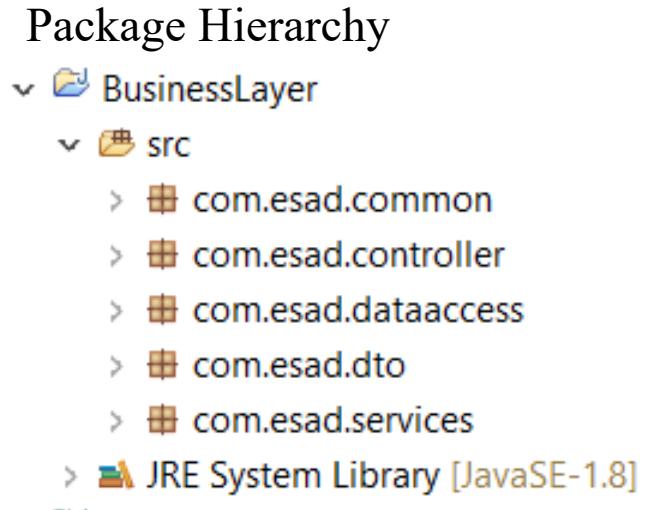
In-class Activity: Exercise 02 for Business Layer Patterns Implementation

- Implement the **Business Layer** assuming the request comes from **Presentation Layer (Controller class)** and invoke the **business delegator (ServiceDelegator)** class that fetches required **business Services (PaymentService, ReservationService)**. You can implement common Service Interface that uses in all business services. Then the required request should be forwarded to the **DataAccess Layer** to execute necessary SQL query. You can apply **Transfer Object Pattern** to send the request among layers and in **DataAccess Layer** you can get those values to map to the Database Entities. Create suitable package hierarchy and Controller implementation should be as following piece of code.

Controller Implementation

```
PaymentDTO paymentDTO = new PaymentDTO();
paymentDTO.setAmount(120000.00);
paymentDTO.setDescription("Electricity Bill");

((IPaymentService)(ServiceDelegate.getService("payment"))).addPayment(paymentDTO);
```



Solution

```
public interface IService {  
}  
  
public interface IPaymentService extends IService{  
    public void addPayment(PaymentDTO paymentDTO);  
}
```

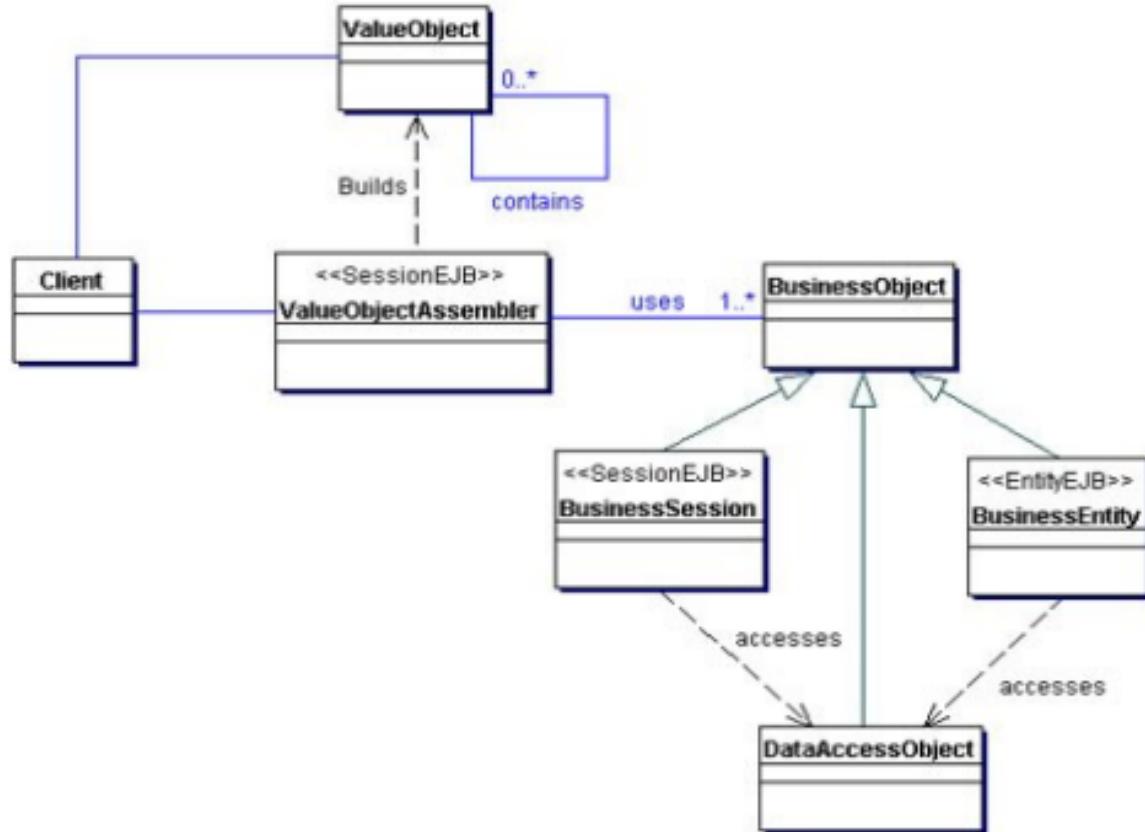
```
public class PaymentServiceImpl implements IPaymentService {  
  
    @Override  
    public void addPayment(PaymentDTO paymentDTO) {  
        double amount = paymentDTO.getAmount();  
        String description = paymentDTO.getDescription();  
        System.out.println("Insert for Payment Table " +  
            description + " = " + amount);  
    }  
}
```

```
public class ServiceDelegate {  
  
    private static final String PAYMENT = "payment";  
  
    private static final String RESERVATION = "reservation";  
  
    public static IService getService(String serviceType){  
  
        if(serviceType.equals(PAYMENT)){  
            return new PaymentServiceImpl();  
        }else if(serviceType.equals(RESERVATION)){  
            return new ReservationServiceImpl();  
        }else{  
            return null;  
        }  
    }  
}
```

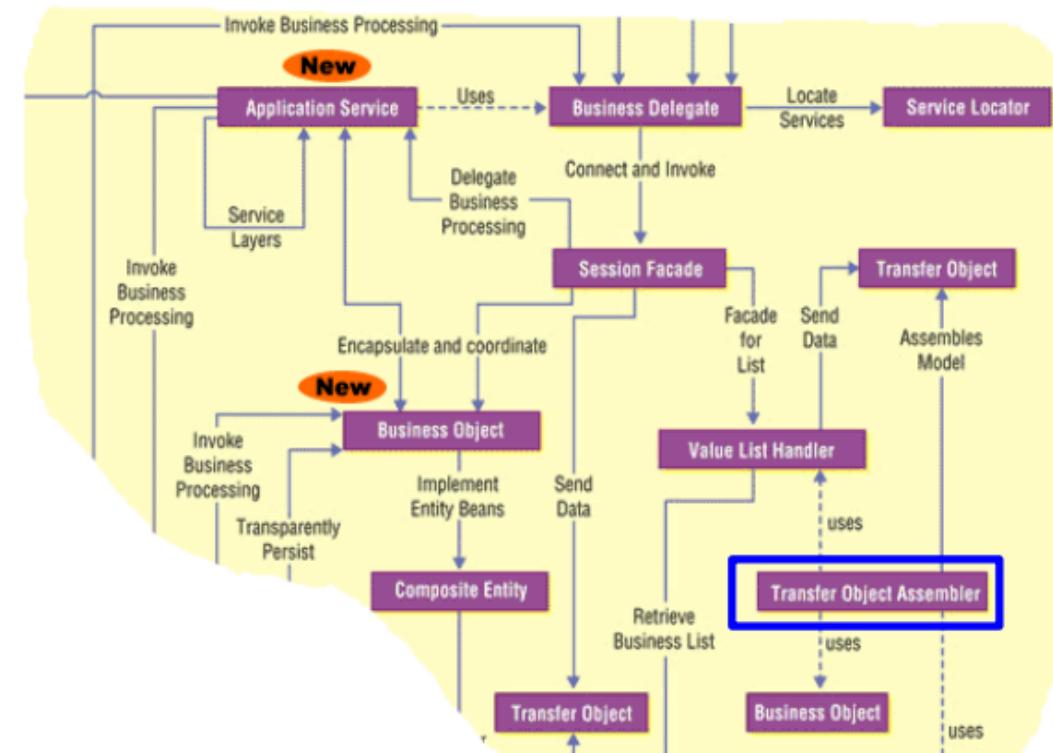
```
public class PaymentDTO {  
  
    private String ID;  
    private String description;  
    private double amount;  
  
    public String getID() {  
        return ID;  
    }  
  
    public void setID(String iD) {  
        ID = iD;  
    }  
  
    public String getDescription() {  
        return description;  
    }  
  
    public void setDescription(String description) {  
        this.description = description;  
    }  
  
    public double getAmount() {  
        return amount;  
    }  
  
    public void setAmount(double amount) {  
        this.amount = amount;  
    }  
}
```

```
public interface IPaymentDao {  
    public boolean addPayment(Object obj);  
}  
  
public class PaymentDAOImpl implements IPaymentDao{  
    @Override  
    public boolean addPayment(Object obj) {  
        return false;  
    }  
}  
  
src  
└ com.esad.common  
   └ ServiceDelegate.java  
└ com.esad.controller  
   └ PaymentController.java  
└ com.esad.dataaccess  
   └ IPaymentDao.java  
   └ PaymentDAOImpl.java  
└ com.esad.dto  
   └ PaymentDTO.java  
└ com.esad.services  
   └ IPaymentService.java  
   └ IReservationService.java  
   └ IService.java  
   └ PaymentServiceImpl.java  
   └ ReservationServiceImpl.java
```

Transfer Object Assembler Pattern

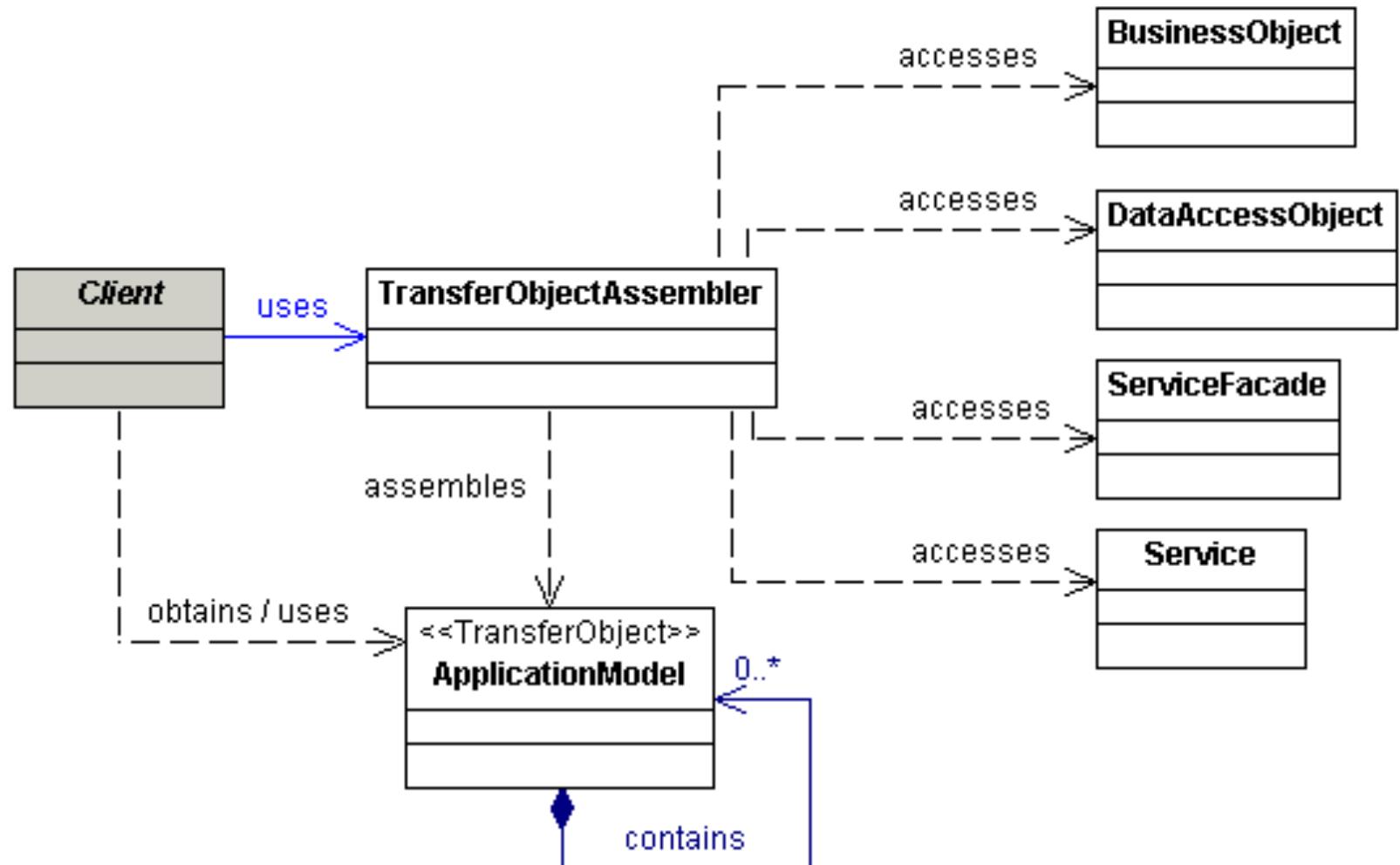


Transfer Object Assembler Pattern

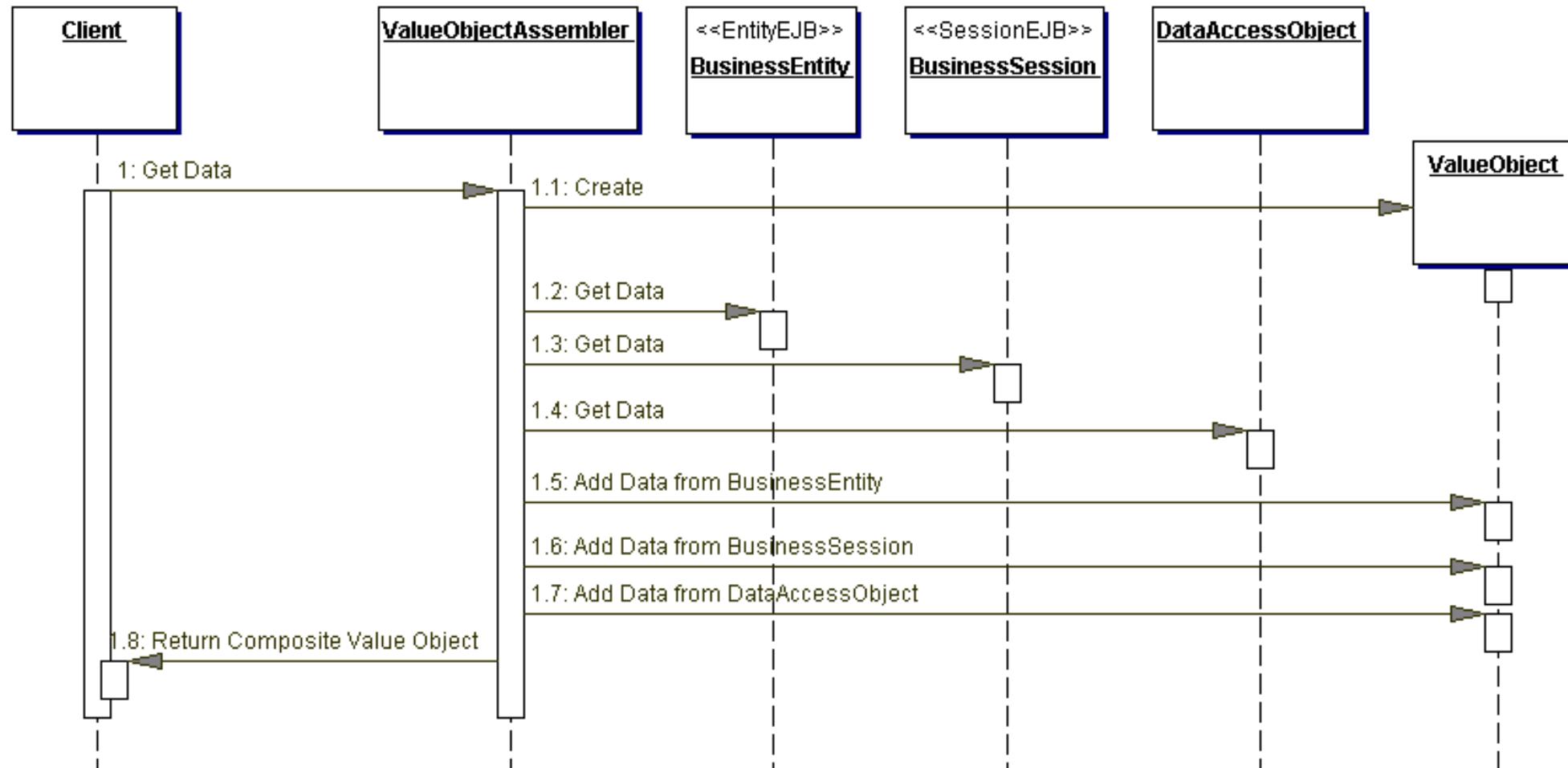


Use a Transfer Object Assembler to build the required model or sub-model. This uses Transfer Objects to retrieve data from various business objects and other objects that define the model or part of the model.

- Use a Transfer Object Assembler to build an application model as a composite Transfer Object.
- The **Transfer Object Assembler aggregates multiple Transfer Objects** from various business components and services and returns it to the client.



Transfer Object Assembler Pattern



In-class Activity: Exercise 04 for Transfer Object Assembler

Refer the below TO classes (ResourceTO, TaskTO, ProjectTO, ProjectManagerTO, and TaskResourceTO)

```
public class ResourceTO {  
    private String resourceId;  
    private String resourceName;  
    private String resourceEmail;  
    public String getResourceId() {  
        return resourceId;  
    }  
    public void setResourceId(String resourceId) {  
        this.resourceId = resourceId;  
    }  
    public String getResourceName() {  
        return resourceName;  
    }  
    public void setResourceName(String resourceName) {  
        this.resourceName = resourceName;  
    }  
    public String getResourceEmail() {  
        return resourceEmail;  
    }  
    public void setResourceEmail(String resourceEmail) {  
        this.resourceEmail = resourceEmail;  
    }  
}  
  
public class TaskTO {  
    private String projectId;  
    private String taskId;  
    private String name;  
    private String description;  
    private Date startDate;  
    private Date endDate;  
    private String assignedResourceId;  
    public String getProjectId() {  
        return projectId;  
    }  
    public void setProjectId(String projectId) {  
        this.projectId = projectId;  
    }  
    public String getTaskId() {  
        return taskId;  
    }  
    public void setTaskId(String taskId) {  
        this.taskId = taskId;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}  
  
public class TaskResourceTO {  
    private String projectId;  
    private String taskId;  
    private String name;  
    private String description;  
    private Date startDate;  
    private Date endDate;  
    private TaskResourceTO assignedResource;  
    public String getProjectId() {  
        return projectId;  
    }  
    public void setProjectId(String projectId) {  
        this.projectId = projectId;  
    }  
    public String getTaskId() {  
        return taskId;  
    }  
    public void setTaskId(String taskId) {  
        this.taskId = taskId;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
public class ProjectManagerTO {  
  
    private String name;  
    private String address;  
    private String projects;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getAddress() {  
        return address;  
    }  
    public String getAddress() {  
        return address;  
    }  
    public void setAddress(String address) {  
        this.address = address;  
    }  
    public String getProjects() {  
        return projects;  
    }  
    public void setProjects(String projects)  
        this.projects = projects;  
    }  
}
```

```
public class ProjectTO {  
  
    private String projectId;  
    private String projectName;  
    private String projectDesc;  
    public String getProjectId() {  
        return projectId;  
    }  
    public void setProjectId(String projectId) {  
        this.projectId = projectId;  
    }  
    public String getProjectName() {  
        return projectName;  
    }  
    public void setProjectName(String projectName) {  
        this.projectName = projectName;  
    }  
    public String getProjectDesc() {  
        return projectDesc;  
    }  
    public void setProjectDesc(String projectDesc) {  
        this.projectDesc = projectDesc;  
    }  
}
```

A Transfer Object Assembler pattern can be implemented to assemble this composite transfer object.

```
public class ProjectDetailsData {  
    private ProjectTO projectData;  
    private ProjectManagerTO projectManagerData;  
    private Collection < TaskResourceTO > listOfTasks;  
  
    public ProjectDetailsData(ProjectTO projectData, ProjectManagerTO projectManagerData,  
        Collection < TaskResourceTO > listOfTasks) {  
        super();  
        this.projectData = projectData;  
        this.projectManagerData = projectManagerData;  
        this.listOfTasks = listOfTasks;  
    }  
}
```

In-class Activity: Exercise 04

- Create Transfer Object Assembler class to aggregate Project data, Project Manager Data, and List of tasks using classes (ProjectTO, ProjectManagerTO, and collection of ResourcesTO) Refer the provided transfer Object classes and implement the **Project Details Assembler** class

Solution

```
public class ProjectDetailsAssembler {  
  
    public ProjectDetailsData getData(String projectId) {  
  
        // Construct the composite transfer object  
        // get project related information from database and set to ProjectDetailsData class object.  
        ProjectTO pData = new ProjectTO();  
  
        // get ProjectManager info and add to ProjectDetailsData  
        ProjectManagerTO pManagerData = new ProjectManagerTO();  
  
        // construct a new TaskResourceTO using Task and Resource data.  
        //get the Resource details from database.  
        // construct a list of TaskResourceTOS  
        Collection < TaskResourceTO > listTasks = new ArrayList < > ();  
        // Add Project Info to ProjectDetailsData  
        // Add ProjectManager info to ProjectDetailsData  
        ProjectDetailsData pData = new ProjectDetailsData(pData, pManagerData, listTasks)  
        return pData;  
    }  
}
```

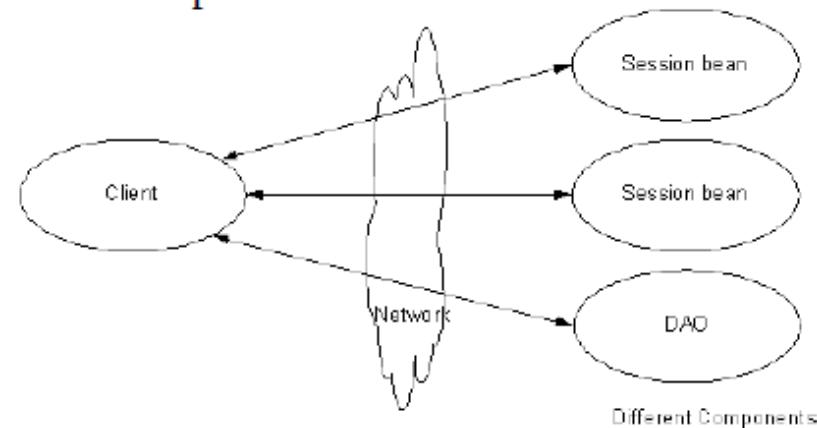
Transfer Object Assembler Pattern

Problem

- ➊ The application model is an *abstraction* of the business data and business logic implemented on the server side as business components.
- ➋ In a J2EE application, the application model is a distributed collection of objects such as *session beans*, *entity beans*, or *DAOs* and *other objects*
- ➌ For a client to obtain the data for the model, such as to display to the user or to perform some processing, it must access individually each distributed object that defines the model

Problem

- ➊ The client must reconstruct the model after obtaining the model's parts from the distributed components. The client therefore needs to have the necessary business logic to construct the model. If the model construction is complex and numerous objects are involved in its definition, then there may be an additional performance overhead on the client due to the construction process



Transfer Object Assembler Pattern

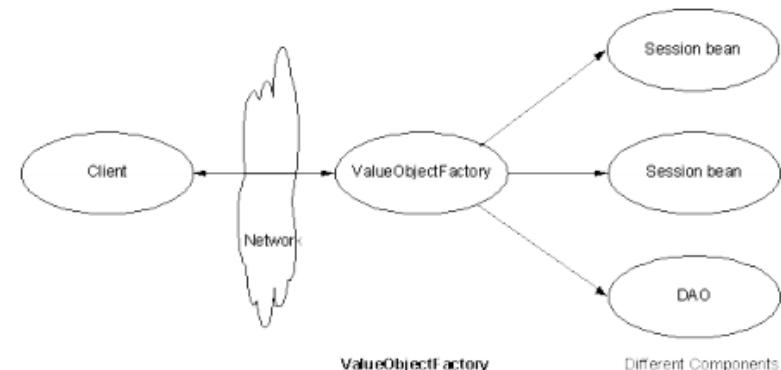
Problem

- When the client constructs the application model, the construction happens on the client side. Complex model construction can result in a significant performance overhead on the client side for clients with limited resources
- Because the client is tightly coupled to the model, changes to the model require changes to the client

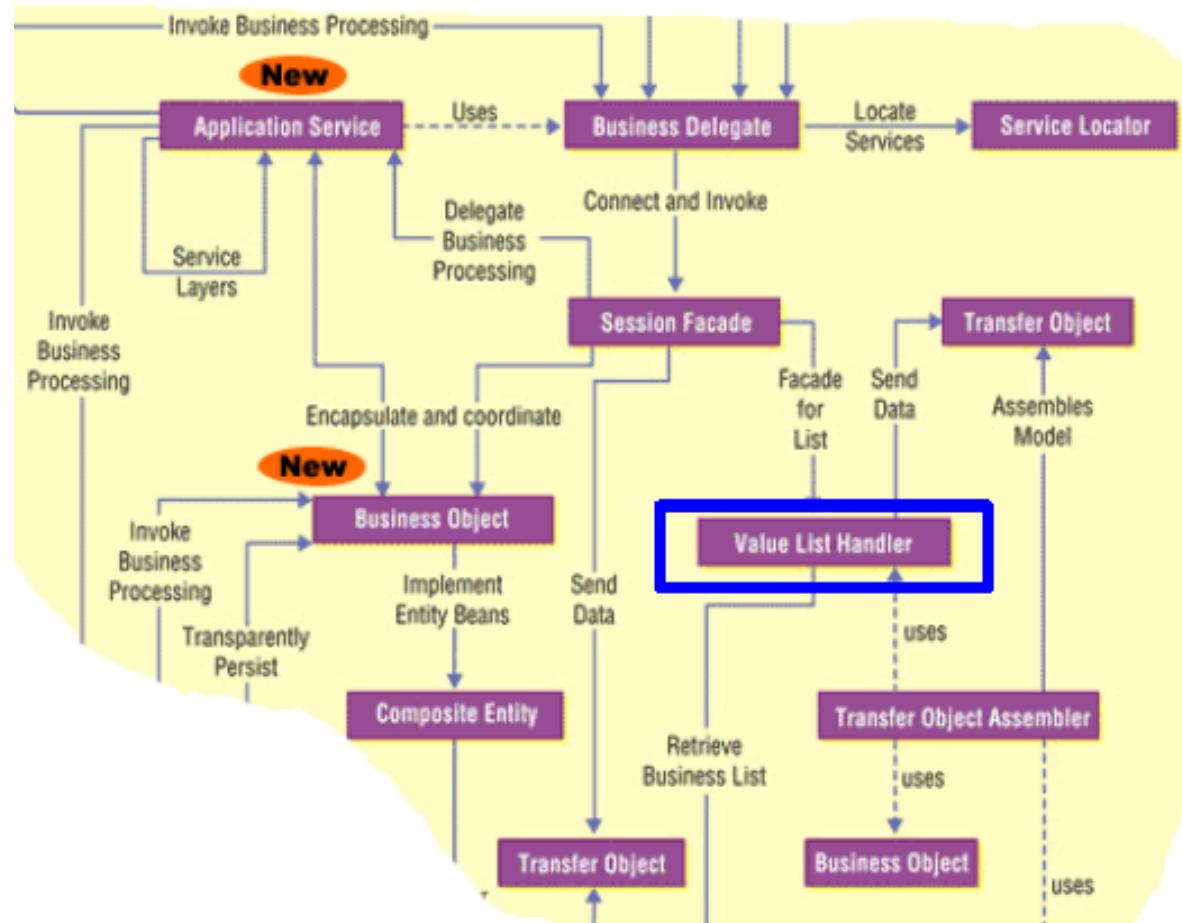
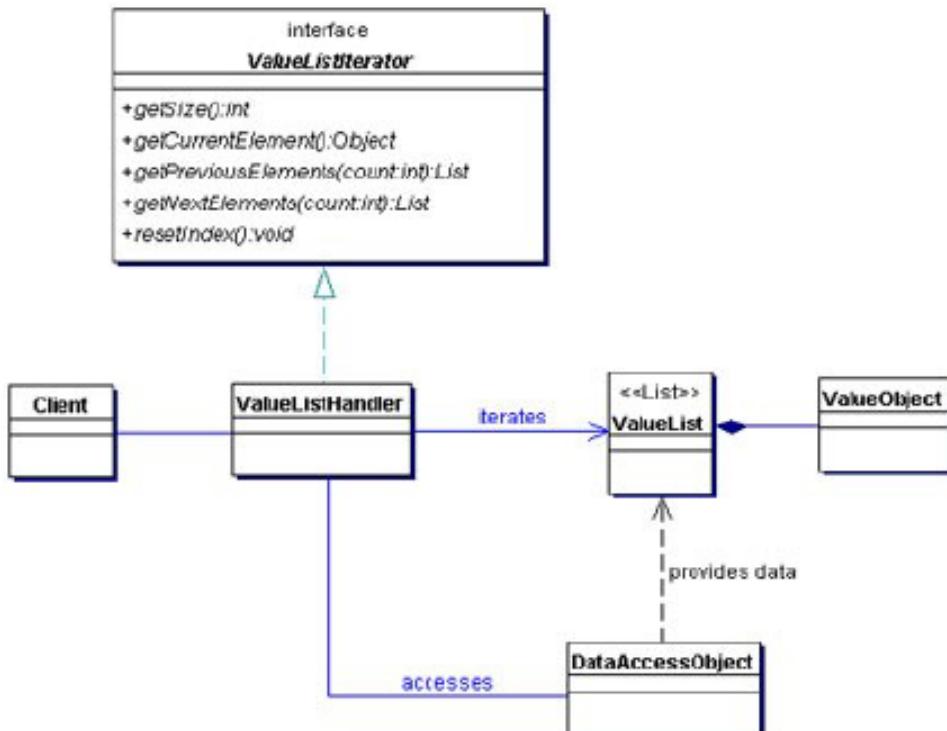
Transfer Object Assembler Pattern Solution

Using the Transfer Object Assembler

- To reduce the network traffic due to accessing multiple components by a client for a single request, this holds different ValueObjects as place holders and respond with a single ValueObject for a client request



Value List Handler Pattern

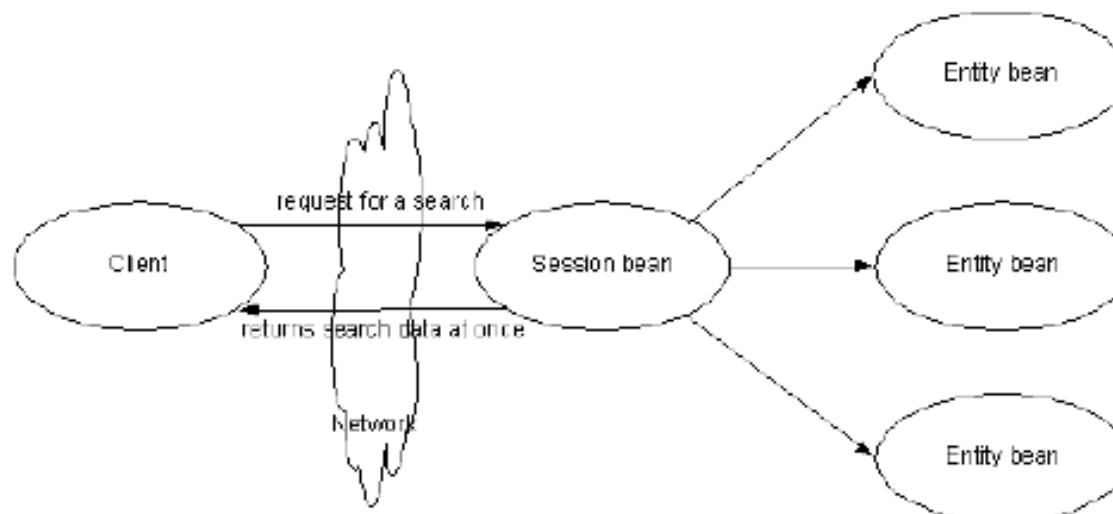


Use a Value List Handler to control the search, cache the results, and provide the results to the client in a result set whose size and traversal meets the client's requirements

Value List Handler Pattern

Problem

- ⌚ Enterprise applications generally have the **search facility** and have to search huge data and retrieve results
- ⌚ If an application returns huge queried data to the client, the client takes long time to retrieve that large data

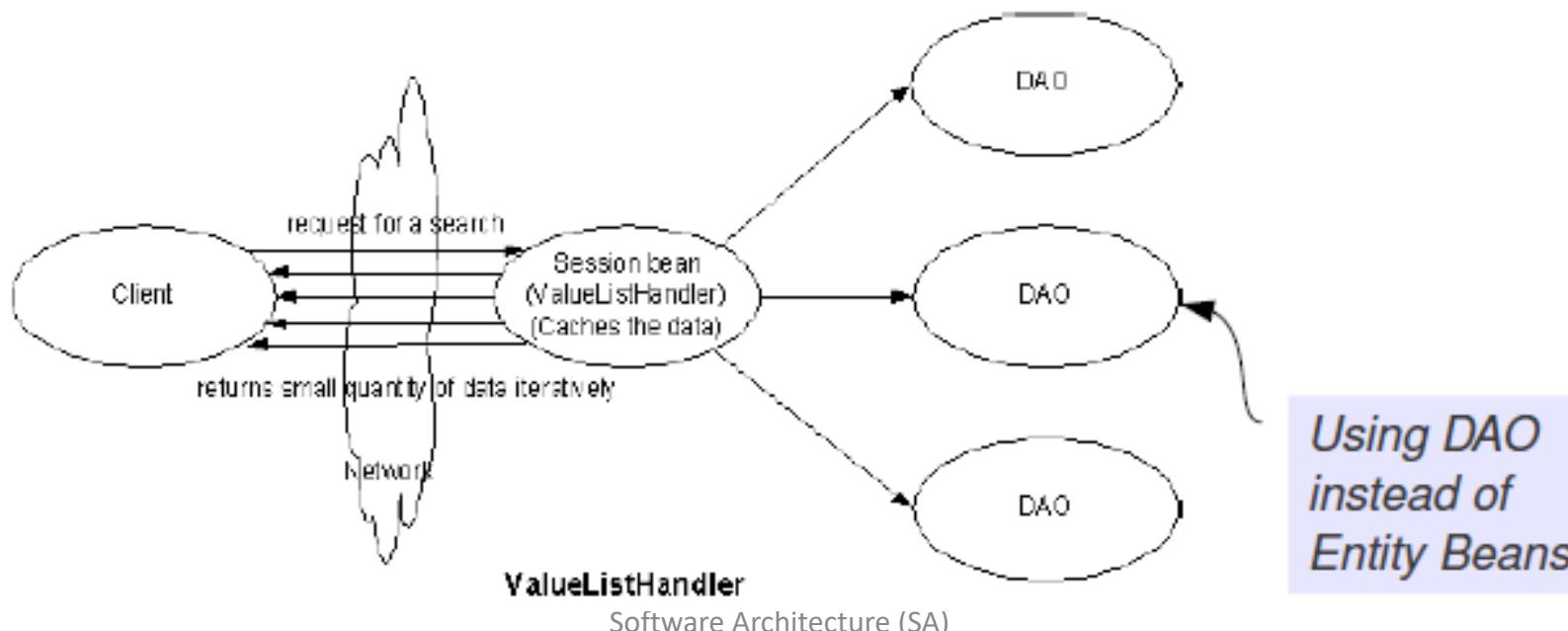


Value List Handler Pattern

Solution

Using the Value List Handler

- Return **small quantity of data** *multiple times iteratively* rather than returning large amount of data at once to the client



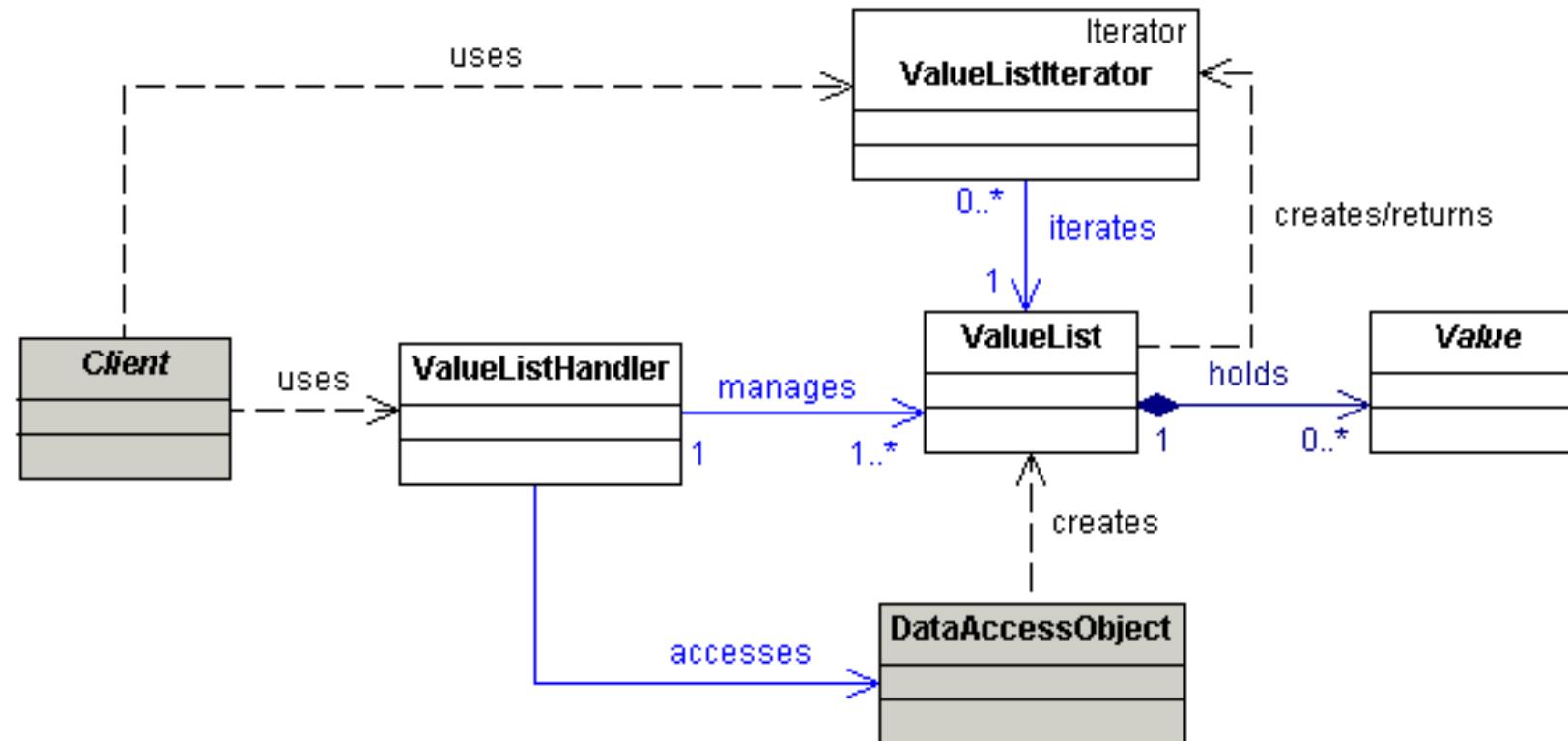
Value List Handler Pattern

Solution

- ➊ This pattern creates a ValueListHandler to control query execution functionality and results caching.
- ➋ This directly accesses a **DAO** that can execute the required query.
- ➌ Then it stores the results obtained from the DAO as a collection of Transfer Objects.
- ➍ The client requests the ValueListHandler to provide the query results as needed.
- ➎ This implements an Iterator pattern [GoF] to provide the solution

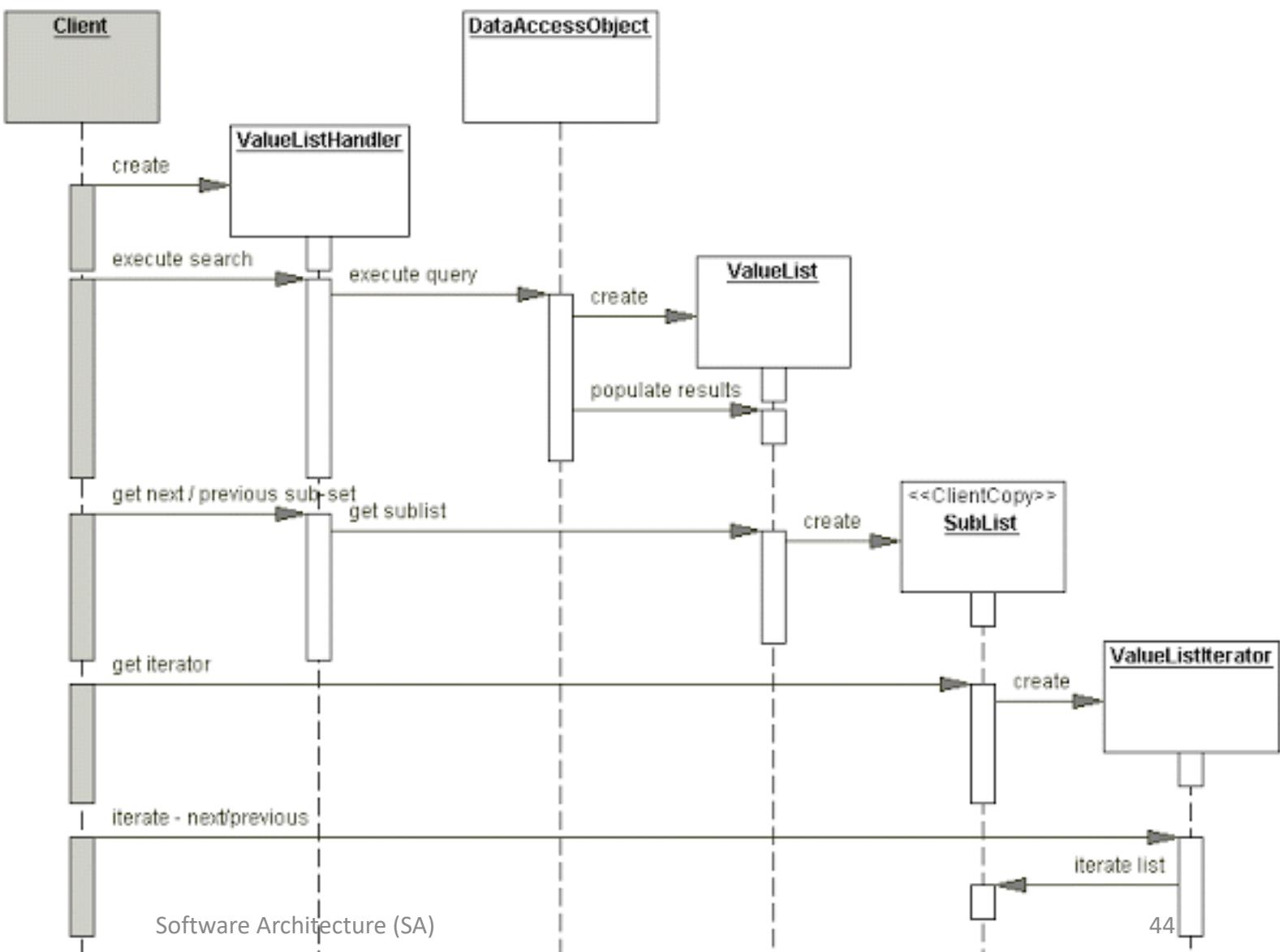
Value List Handler Pattern

Class Diagram



Value List Handler Pattern

Sequence Diagram



Value List Handler Pattern

```
public class ProjectTO {  
    private String projectId;  
    private String projectName;  
    private String managerId;  
    private Date startDate;  
    private Date endDate;  
    private boolean started;  
    private boolean completed;  
    private boolean accepted;  
    private Date acceptedDate;  
    private String customerId;  
    private String projectDescription;  
    private String projectStatus;  
  
    public String getProjectId() {  
        return projectId;  
    }  
  
    public void setProjectId(String projectId) {  
        this.projectId = projectId;  
    }  
}
```

```
public interface ValueListIterator {  
    public int getSize() throws IteratorException;  
  
    public Object getCurrentElement() throws IteratorException;  
  
    public List getPreviousElements(int count) throws IteratorException;  
  
    public List getNextElements(int count) throws IteratorException;  
  
    public void resetIndex() throws IteratorException;  
  
    // other common methods as required  
    // ...  
}
```

```

public class ValueListHandler implements ValueListIterator {

    protected List list;
    protected ListIterator listIterator;
    public ValueListHandler() {}

    protected void setList(List list) throws IteratorException {
        this.list = list;
        if (list != null)
            listIterator = list.listIterator();
        else
            throw new IteratorException("List empty");
    }

    public Collection getList() {
        return list;
    }

    public int getSize() throws IteratorException {
        int size = 0;
        if (list != null)
            size = list.size();
        else
            throw new IteratorException("No data found"); //No Data
        return size;
    }

    public Object getCurrentElement() throws IteratorException {

        Object obj = null;
        // Will not advance iterator
        if (list != null) {
            int currIndex = listIterator.nextInt();
            obj = list.get(currIndex);
        } else
            throw new IteratorException("");
        return obj;
    }
}

public List getPreviousElements(int count) throws IteratorException {
    int i = 0;
    Object object = null;
    LinkedList list = new LinkedList();
    if (listIterator != null) {
        while (listIterator.hasPrevious() && (i < count)) {
            object = listIterator.previous();
            list.add(object);
            i++;
        }
    } else
        throw new IteratorException("No data found");
    return list;
}

public List getNextElements(int count) throws IteratorException {
    int i = 0;
    Object object = null;
    LinkedList list = new LinkedList();
    if (listIterator != null) {
        while (listIterator.hasNext() && (i < count)) {
            object = listIterator.next();
            list.add(object);
            i++;
        }
    } else
        throw new IteratorException("No data found");
    return list;
}

public void resetIndex() throws IteratorException {
    if (listIterator != null) {
        listIterator = list.listIterator();
    } else
        throw new IteratorException("No data found");
}

```

References

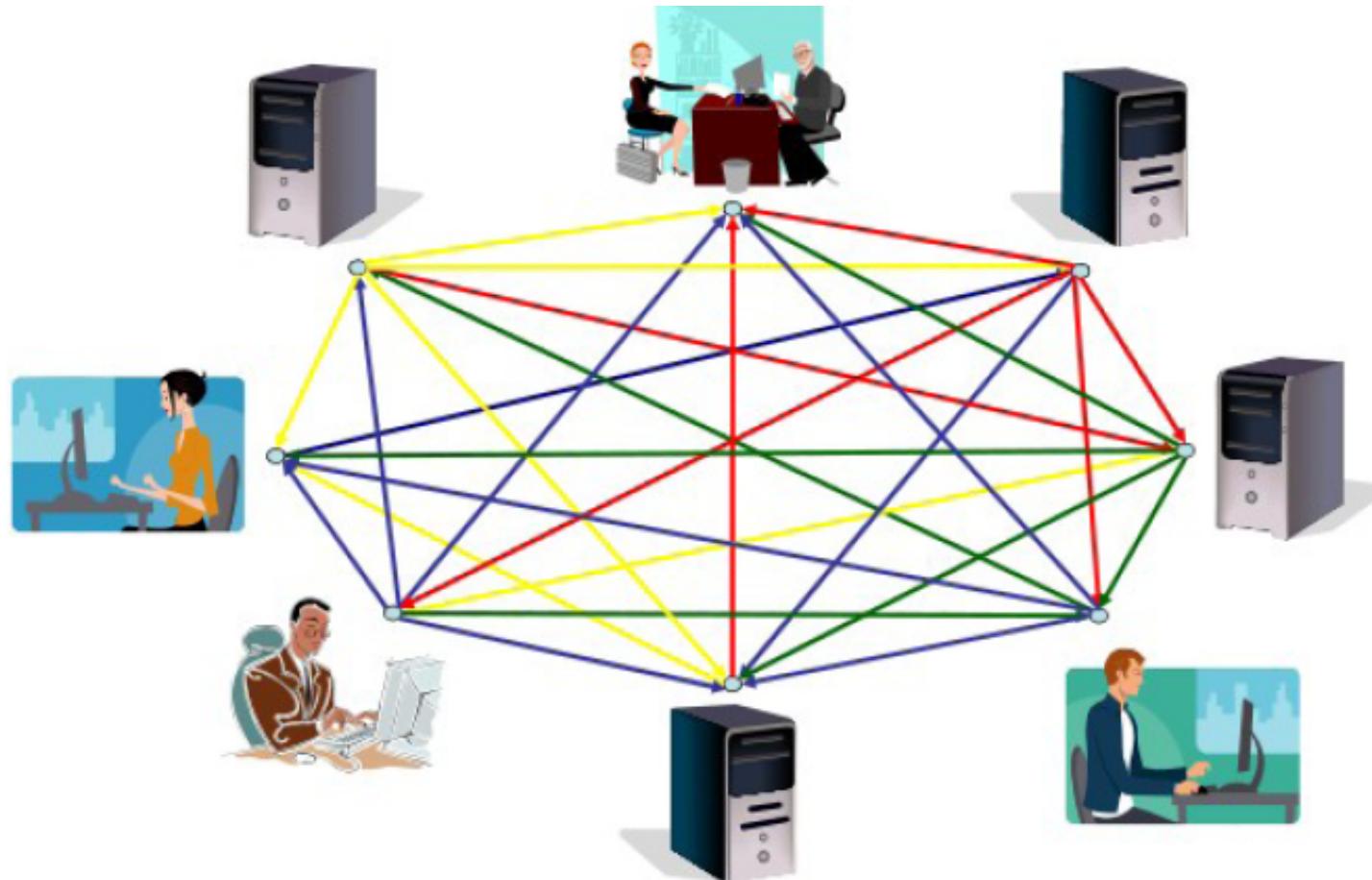
- ⌚ Patterns of Enterprise Application Architecture (PoEAA): *Martin Fowler*
- ⌚ J2EE Design Patterns: *William Crawford & Jonathan Kaplan*
- ⌚ Core J2EE Patterns: *Deepak Alur, John Crupi, Dan Malks*
- ⌚ http://articles.techrepublic.com.com/5100-10878_11-5064520.html
- ⌚ <http://www.precisejava.com/javaperf/j2ee/Patterns.htm#Patterns103>



Enterprise Application Integration

**Software Architecture
3rd Year– Semester 1
By Udara Samaratunge**

Service Oriented Architecture - SOA



SOA Evolution

- Main Frames – Used Tapes to transfer files
- Later lower level socket based communication was used
- Then came, Network File System (NFS) and File Transfer Protocols (FTP)
- Remote Procedure Calls (RPCs) got matured along with the improved of server hardware
- CORBA came in but the advent of Java resulted the demise of CORBA
- DCOM came in but its proprietary nature resulted its demise
- SOAP relies on XML as the payload, which has got much higher degree of interoperability between programming languages.

Problems related to RPC

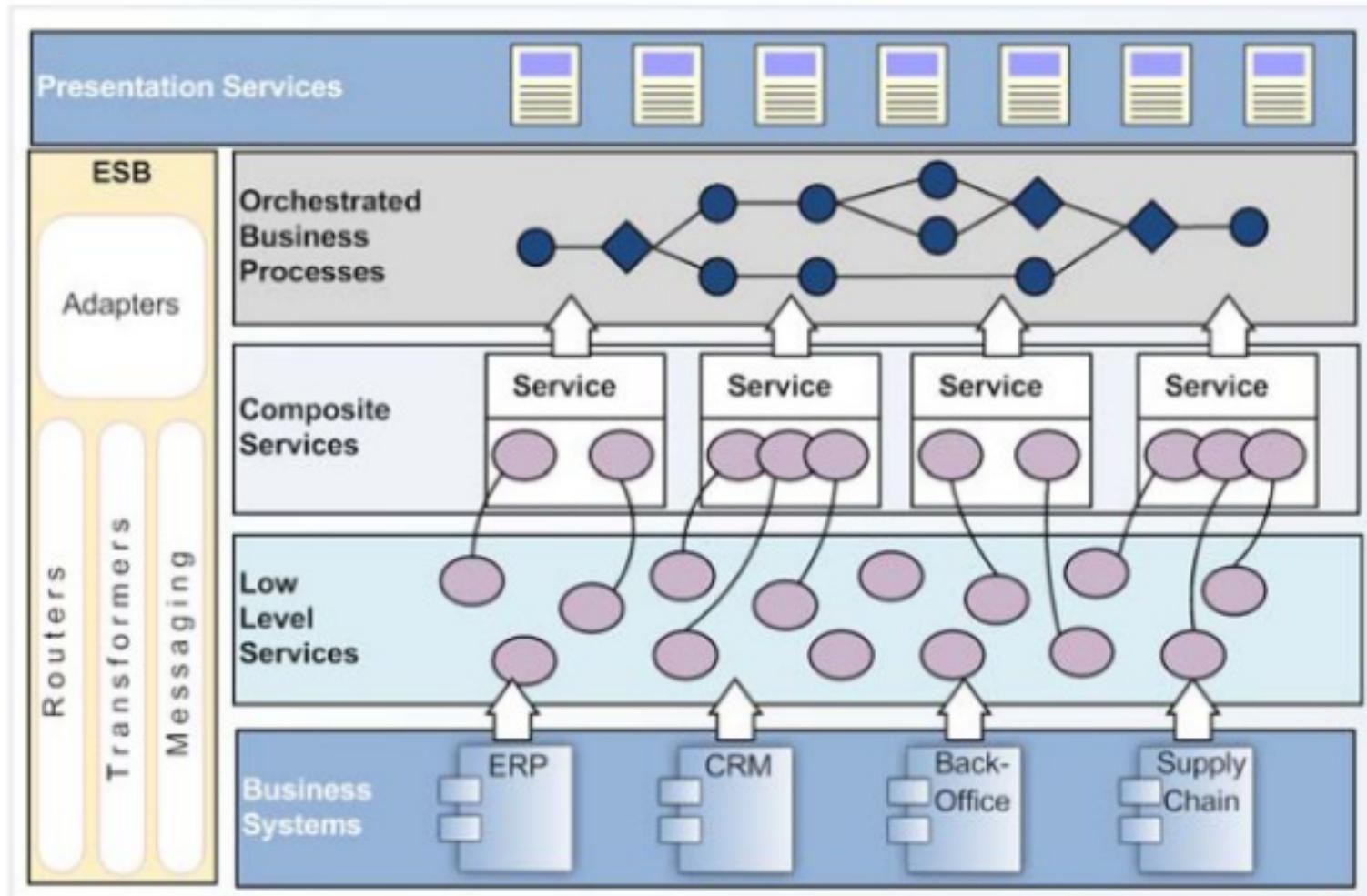
- **Tight coupling** between local and remote systems requires significant bandwidth demands
- **Interoperability issues** – Mainly due to incompatible data types in different languages

SOAP's message style can overcome above issues. SOA was developed keeping “loose coupling” and the “interoperability” in mind

Why SOA?

- CORBA, EJB, DCOM introduced a **highly coupled RPC**. [*Unlike SOA*]
- EJB and DCOM were **tied to specific platforms** and not at all inter-operable. [*Unlike SOA*]
- EJB, DCOM and CORBA were more relied on **commercial oriented products**. [*Unlike SOA*]
- **SOA** can be implemented using a **completed “Open Source” Stack**.
- **SOA** relies on **XML** as the *underlying data representation*, unlike the others, which used *proprietary binary-based objects*
- Unlike CORBA, EJB or DCOM, **SOA** is **more than a RPC technology**, It is a,
 - Governance
 - SLAs (Service Level Agreements)
 - Meta-data Definitions/ Registries

The SOA Environment

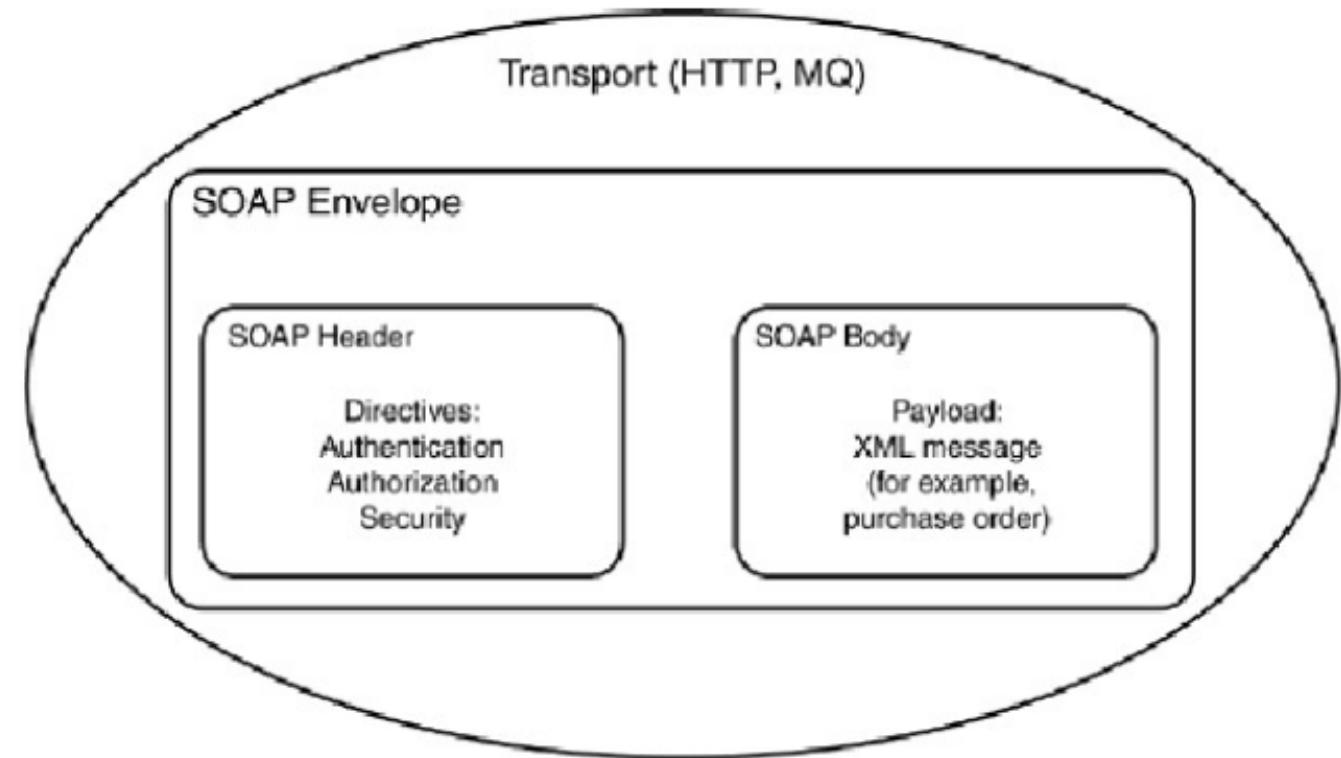


The characteristics of SOA

- The service **Interface/Contract**
- A service must have well defined interface contract.
- This contract should identify,
 - The “**operations**” that are available **through the service**
 - “**Data Requirements**” for any “**Exchanged Information**”
- **WSDL (Web Service Description Language)** is a good example for a **service contract**
- Web services – Related technologies = **XML / XML Schema**
- **Web service** use **SOAP** as **communication Protocol**. [**Simple Object Access Protocol**]
- **SOAP** runs on **HTTP protocol** & uses **default port 80**. **Message** structured as **XML**.

XML Messaging SOAP

- The SOAP envelop is just a container to hold XML data.
- **SOAP envelope**
 - **SOAP Header** – Contains information related to the message and its security
 - **SOAP Body** – Contains the **message payload**
- Requests are encoded in XML and sent via HTTP POST
- Most **firewalls allow** HTTP traffic. This allows **XML-RPC** or **SOAP** messages to be used as **HTTP messages**.



WSDL

- Three Sections

- **What Section** – Input and Output messages (<wsdl:types>, <wsdl:message>)
- **How Section** – How messages should be packaged (bind) to different protocols in the SOAP envelop and how to transfer it (<wsdl:binding>)
- **Where Section** – The endpoint details (<wsdl:service>)

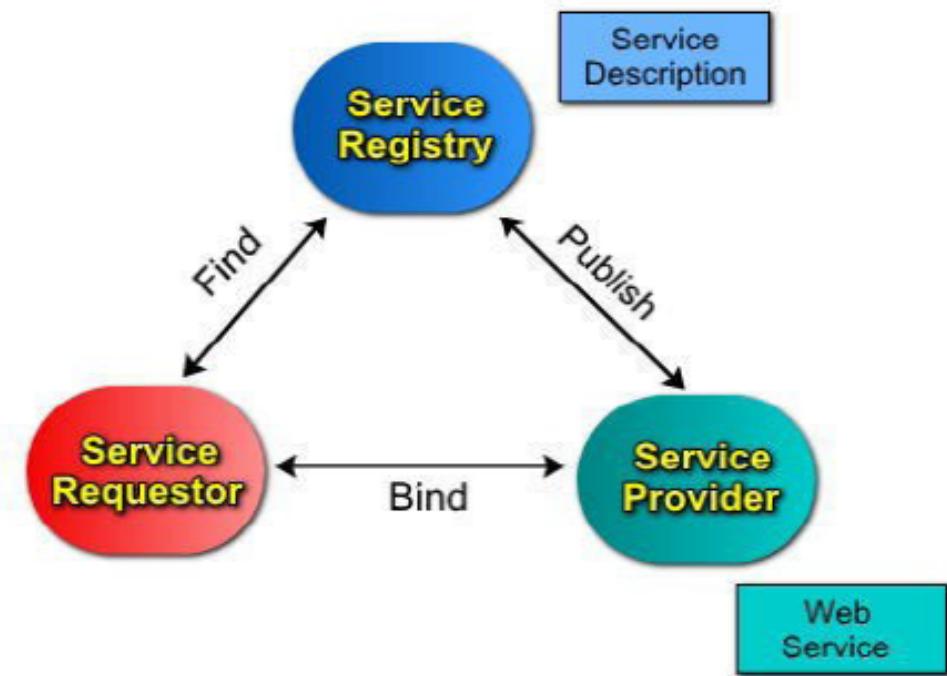
```
-<wsdl:definitions targetNamespace="http://ead">
  <wsdl:documentation> Please Type your service description here </wsdl:documentation>
+<wsdl:types></wsdl:types>
+<wsdl:message name="farenhit2celciusRequest"></wsdl:message>
+<wsdl:message name="farenhit2celciusResponse"></wsdl:message>
+<wsdl:message name="celcius2farenhitRequest"></wsdl:message>
+<wsdl:message name="celcius2farenhitResponse"></wsdl:message>
-<wsdl:portType name="TempWSPortType">
  -<wsdl:operation name="farenhit2celcius">
    <wsdl:input message="ns:farenhit2celciusRequest" wsaw:Action="urn:farenhit2celcius"/>
    <wsdl:output message="ns:farenhit2celciusResponse" wsaw:Action="urn:farenhit2celciusResponse"/>
  </wsdl:operation>
  -<wsdl:operation name="celcius2farenhit">
    <wsdl:input message="ns:celcius2farenhitRequest" wsaw:Action="urn:celcius2farenhit"/>
    <wsdl:output message="ns:celcius2farenhitResponse" wsaw:Action="urn:celcius2farenhitResponse"/>
  </wsdl:operation>
</wsdl:portType>
+<wsdl:binding name="TempWSSoap11Binding" type="ns:TempWSPortType"></wsdl:binding>
+<wsdl:binding name="TempWSSoap12Binding" type="ns:TempWSPortType"></wsdl:binding>
+<wsdl:binding name="TempWSHttpBinding" type="ns:TempWSPortType"></wsdl:binding>
-<wsdl:service name="TempWS">
  -<wsdl:port name="TempWSHttpSoap11Endpoint" binding="ns:TempWSSoap11Binding">
    <soap:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpSoap11Endpoint"/>
  </wsdl:port>
  -<wsdl:port name="TempWSHttpSoap12Endpoint" binding="ns:TempWSSoap12Binding">
    <soap12:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpSoap12Endpoint"/>
  </wsdl:port>
  -<wsdl:port name="TempWSHttpEndpoint" binding="ns:TempWSHttpBinding">
    <http:address location="http://192.168.2.2:8080/axis2/services/TempWS.TempWSHttpEndpoint"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

The diagram illustrates the three sections of WSDL by grouping the code with curly braces:

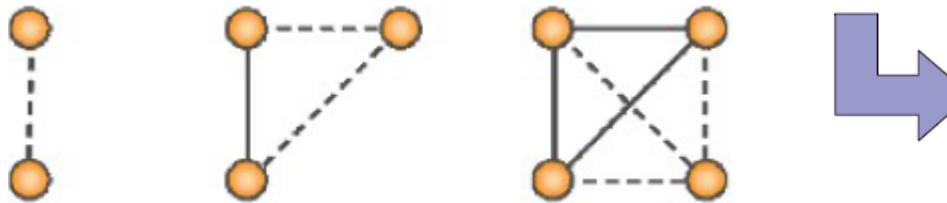
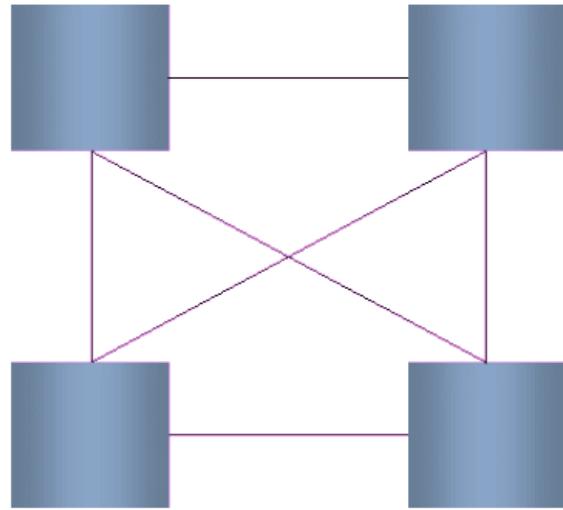
- What:** Groups the first four lines of the WSDL code, which define the service's target namespace and documentation.
- How:** Groups the port type definition, which includes two operations (farenhit2celcius and celcius2farenhit) and their corresponding input and output messages.
- Where:** Groups the service definition, which contains three ports (TempWSHttpSoap11Endpoint, TempWSHttpSoap12Endpoint, and TempWSHttpEndpoint) each bound to a specific WSDL binding.

Web Services Model

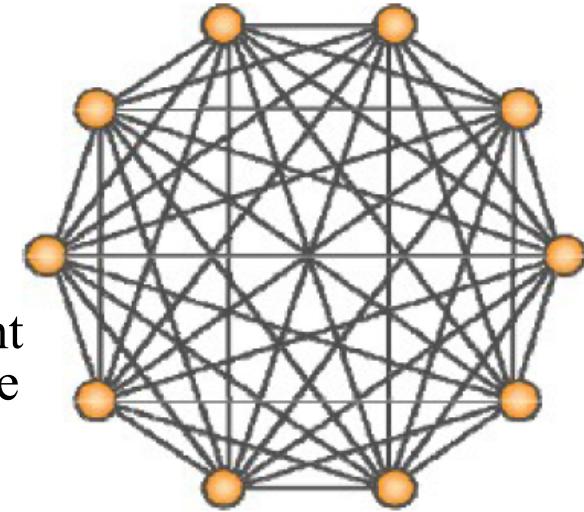
- UDDI is an XMLbased standard for describing, publishing, and finding Web services
- UDDI = (Universal Description, Discovery and Integration)
- UDDI uses WSDL to describe interfaces to web services
- **Approaches of writing web service**
 - **Bottom-Up/Code First Approach** [Implement web service method first]
 - **Top-Down/Contract First Approach** [Write WSDL first then generate Stub classes & Skeleton classes]
- If you generate web services for different plat-forms (Java or .NET) which method is most suitable?
- **Service Provider:** The provider of web service
- **Service Requester:** The web service consumer
- **Service Registry:** The central directory of services



Point-to-Point Integration

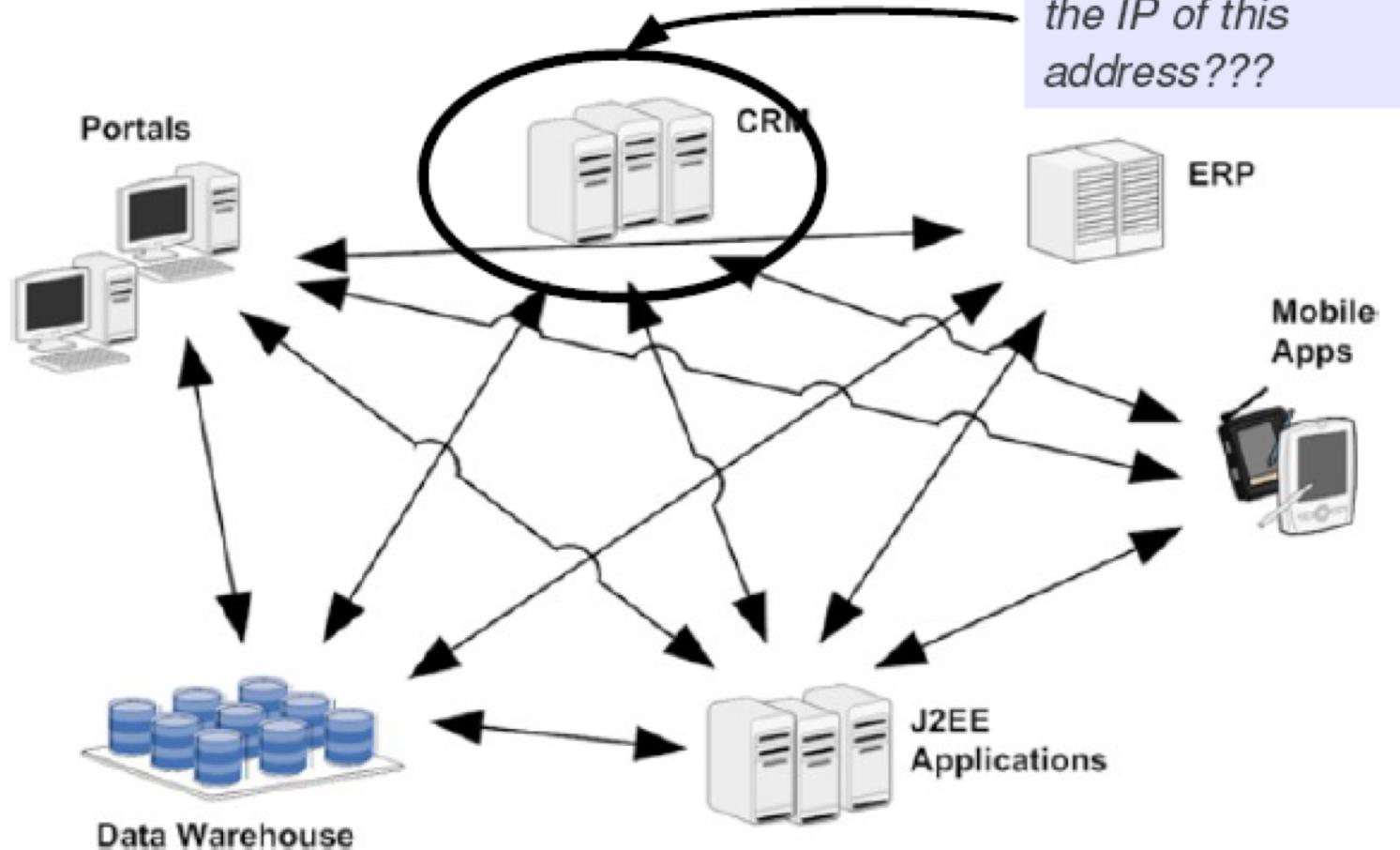


- Benefits:
 - Provides a way to connect each other
- Drawbacks:
 - Isolated without insufficient
 - Extremely “Spaghetti” like architecture, create headaches
- Specifically, linking every component to every other component will require **N(N-1)/2 physical connections**
- N = Total Number of Components in the Network
- E.g: If there are 10 components in the network,
- Total number of physical connections = $10 (10-1)/2 = 45$



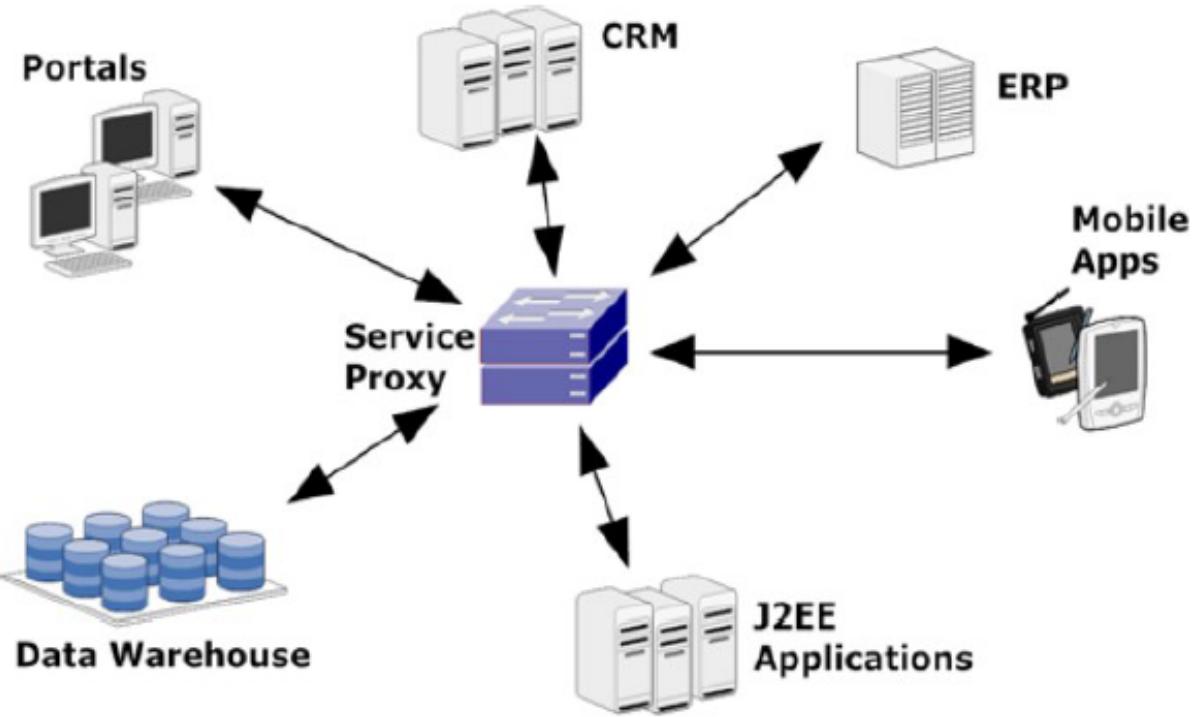
Why ESB?

The Service Transparency



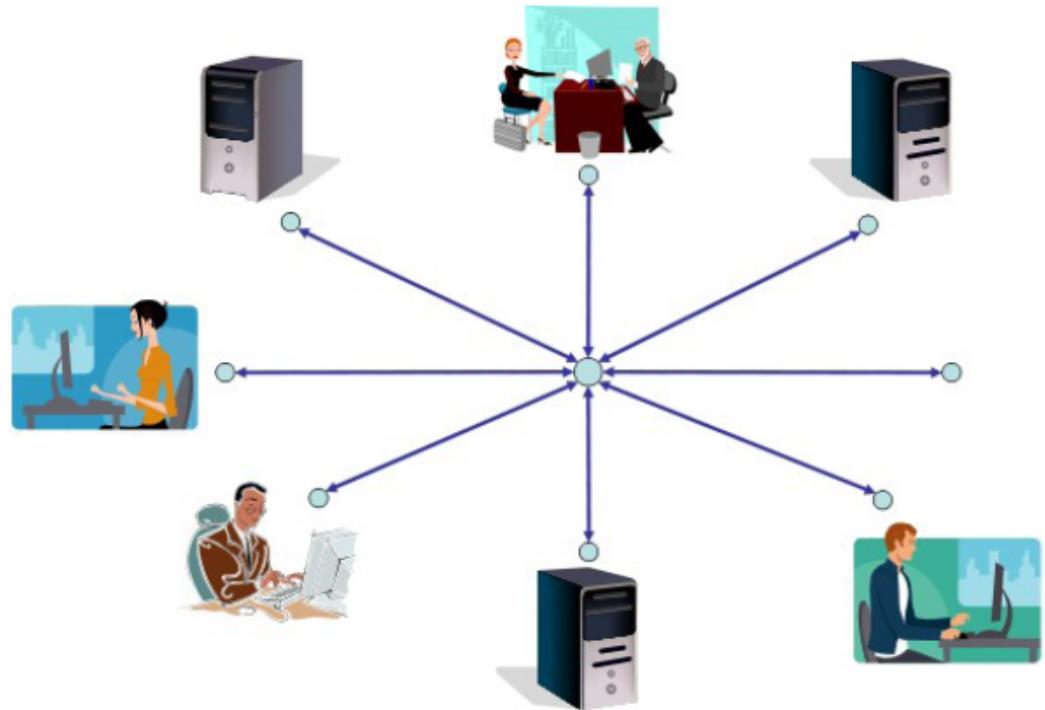
Solution for the Issue

The Service Transparency



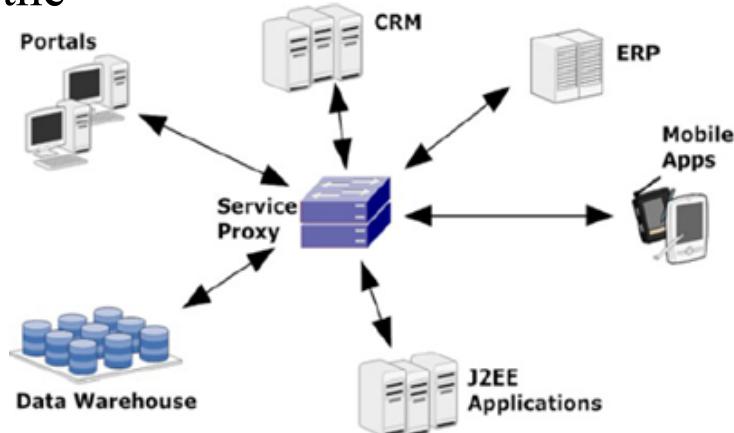
Hub-Spoke Model

A more centralized approach to the previous point-to-point approach



Solution from ESB

- An **ESB** or a **Service Proxy** can be the solution to the mess created by the point-to-point approach. **ESB** = Enterprise Service Bus
- All service calls are directed to the **proxy** or **gateway**, which in turn, forwards the message to the appropriate endpoint destination.
- If an **endpoint is changed**, only the **proxy configuration** will be required to be **changed**.
- Each **component** communicates with **Proxy**. **Proxy** should know how to send the **message for destination**.
- Now **component free** from maintaining IP addresses of destination. That **responsibility delegated to 3rd party module called ESB**.
- **ESB mediate the message for exact endpoint based on proxy details.**



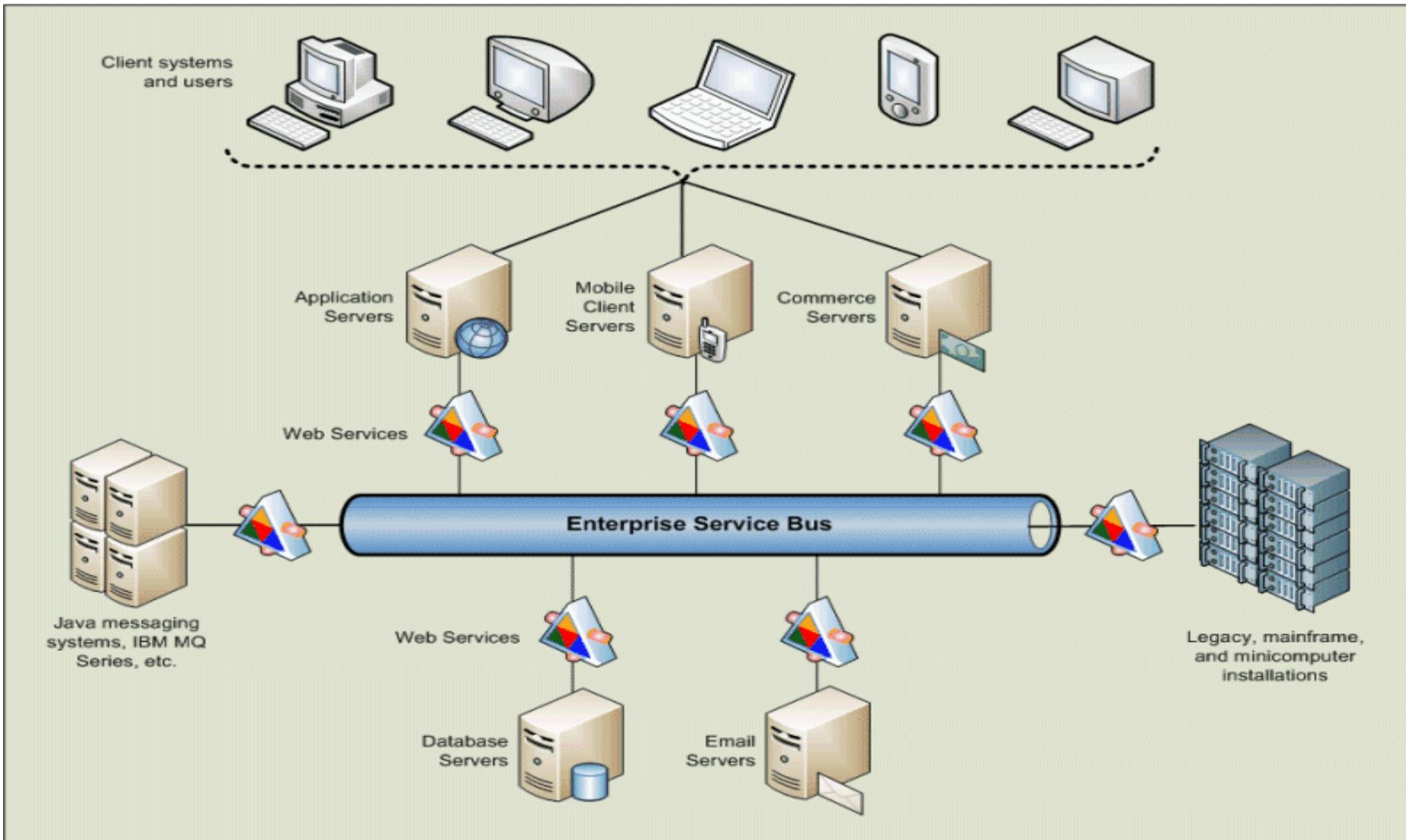
What is an ESB?



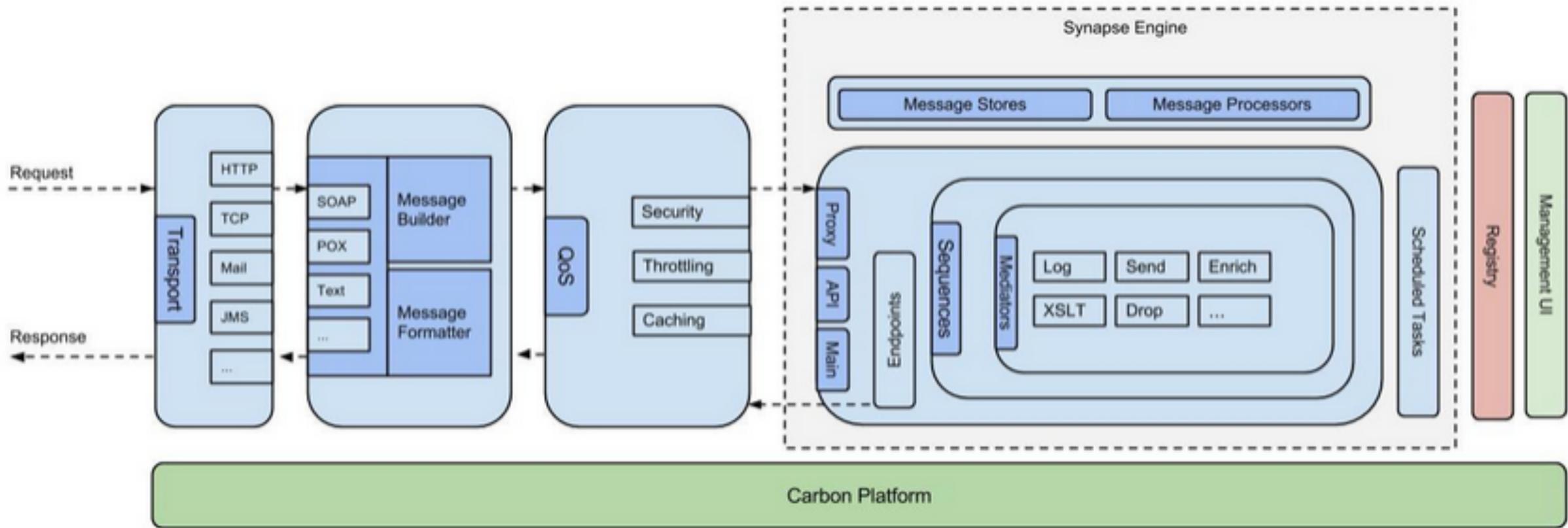
ESB “is a software architecture model used for designing and implementing the interaction and communication between **mutually interacting** software applications in **Service Oriented Architecture**”

- Promotes **asynchronous message mediation**
- Message **identification and routing** between applications and services.
- Allows messages to flow across **different transport protocols**
- **Transforming** of messages
- Allows **secure, reliable** communications
- Extensible architecture (based on pluggable components)

What is an ESB ? Cont....

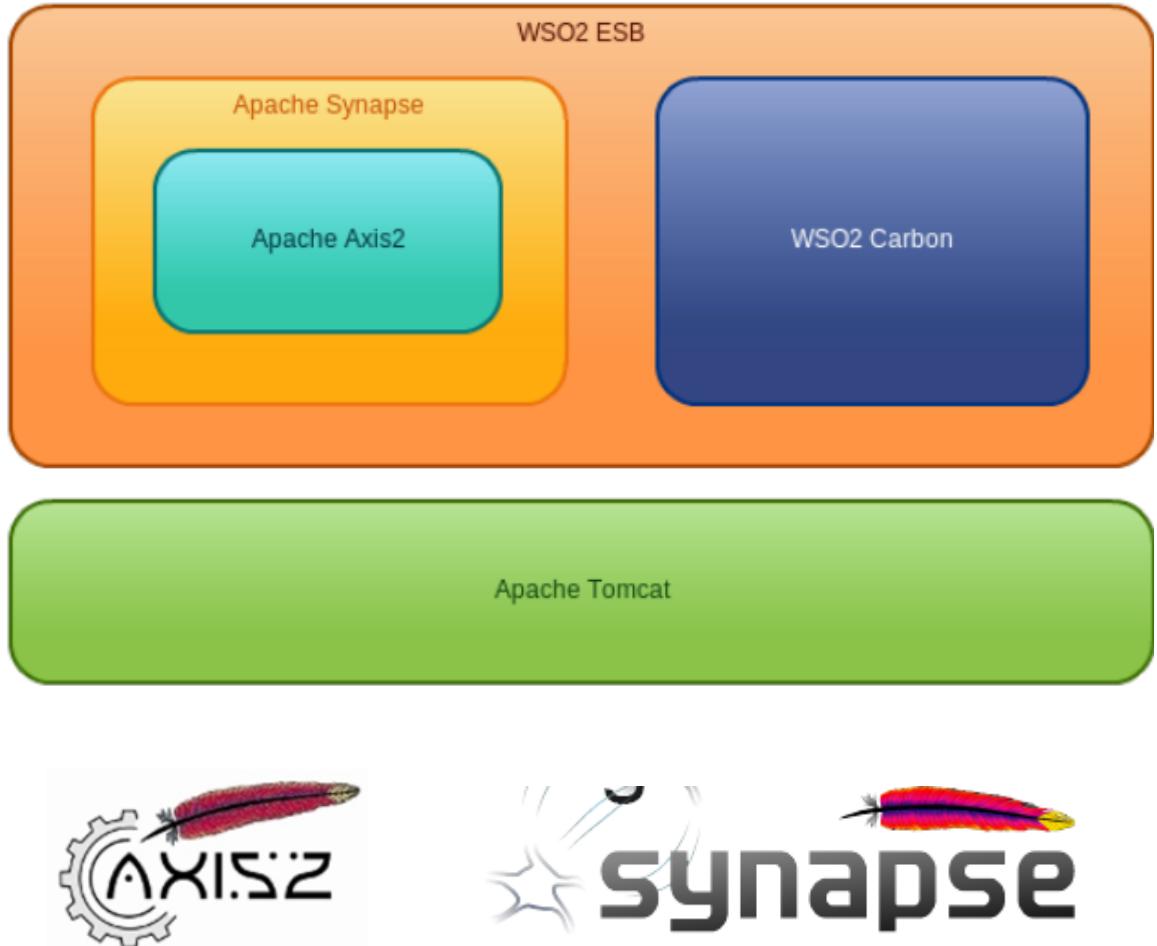


WSO2 ESB Architecture



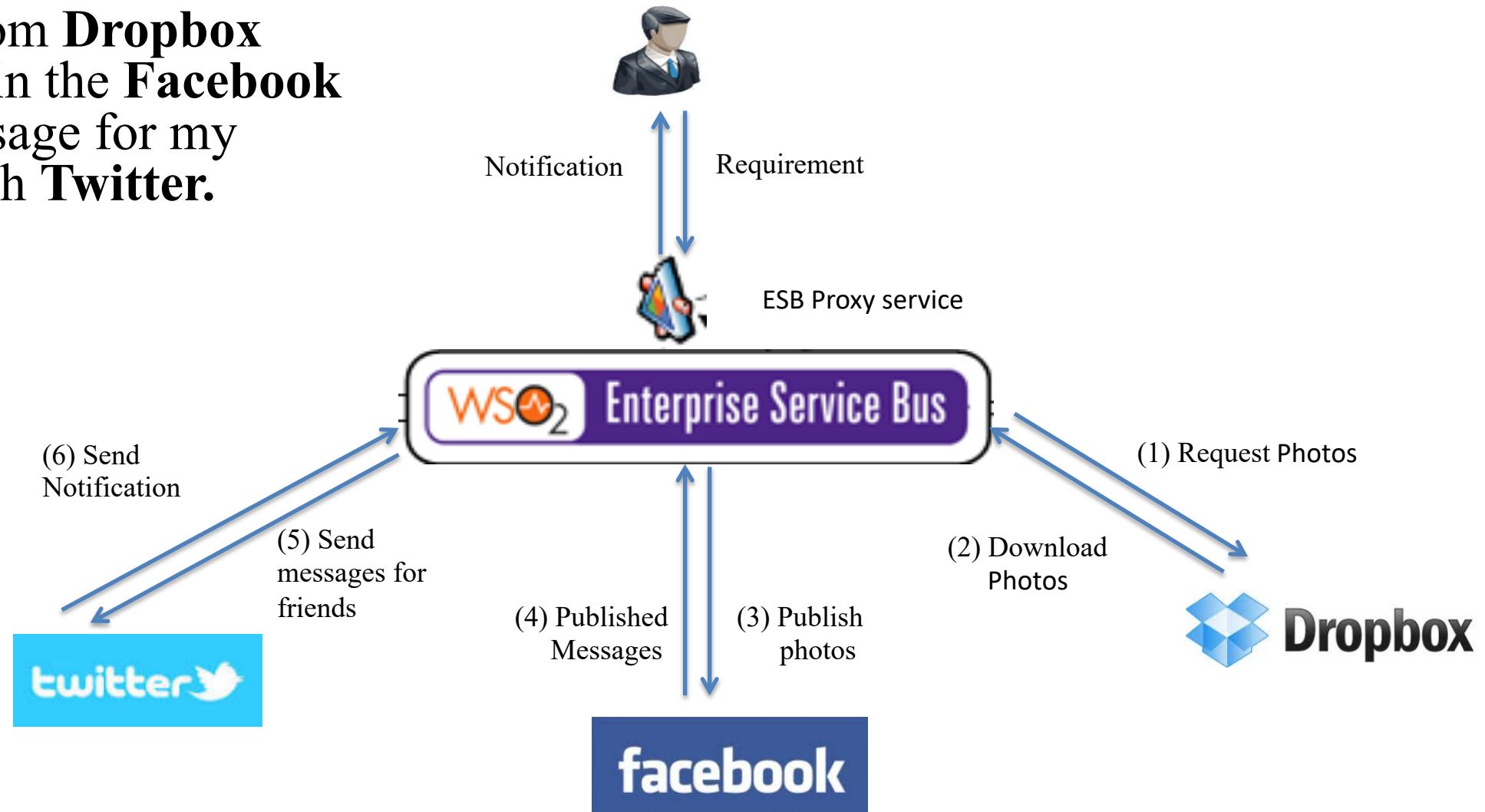
Technology Stack

- WSO2's core product, Middleware platform.
- A dynamic component model built for Java
- Components can be **started, stopped, installed, uninstalled** etc without reboot
- Built on **OSGi concepts**
- Powers SOA capabilities
- EVERYTHING that **WSO2 builds is based on Carbon**
- **Apache Synapse:**
 - Based on Axis2/Java
 - Acts a mediation library for different protocols
 - Supports different protocols through SOAP based Proxy Services



ESB Business Scenario.

- Get photos from **Dropbox** publish them in the **Facebook** and send message for my friends through **Twitter**.

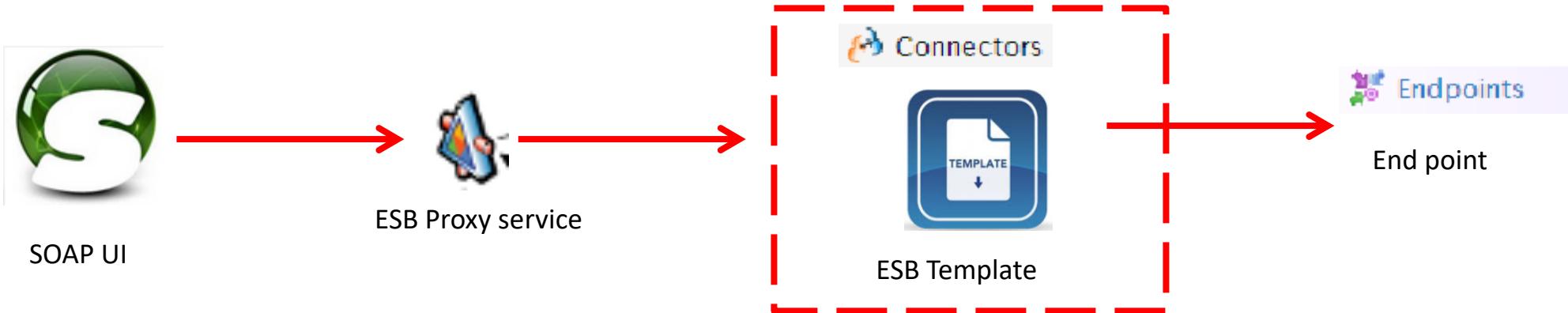


Apache Synapse

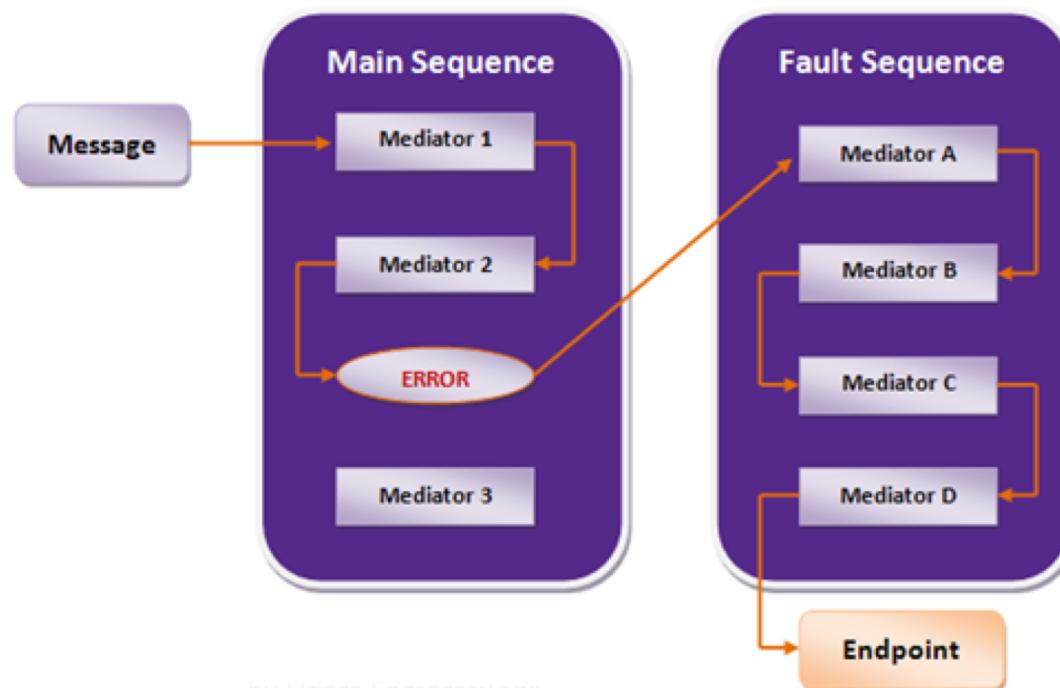
Apache Synapse “*is a lightweight and high-performance ESB with a powerful mediation engine.*”

- Default transport – **HTTP-NIO** (configurable pool of non-blocking worker threads)
- Support for any request types **XML**, **SOAP**, plain text, binary, **JSON** and etc.
- Supports any protocol **HTTP**, **HTTPS**, Mail (POP3, IMAP, SMTP), **JMS**, **TCP**, **UDP**, **VFS**, **SMS**, **XMPP** and **FIX**
- Non-blocking **HTTP/HTTPS** transports
- Support for **WS-*** standards (**WS-Addressing**, **WS-Security** and **WS-Reliable Messaging**)

Flow of request



In case an error occurs in the main sequence while processing, the message goes to the fault sequence.



Sample Request types

SOAP Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:urn="urn:wso2.connector.googledrive.getfile">  
    <soapenv:Body>  
        <root>  
            <urn:fileId>1QL5LZOm9m4-h1lehXgU6gN4Kjovf8o3sqPKvw2RN40k</urn:fileId>  
            <urn:updateViewedDate>false</urn:updateViewedDate>  
            <urn:useServiceAccount>false</urn:useServiceAccount>  
            <urn:clientId>521684679704.apps.googleusercontent.com</urn:clientId>  
            <urn:clientSecret>AOZtcdakFwcnx1BuIavjtMEX</urn:clientSecret>  
            <urn:accessToken></urn:accessToken>  
            <urn:refreshToken>1/khM2ZQlpe_1PSp8WI</urn:refreshToken>  
            <urn:serviceAccountEmail>  
                757865184057@developer.gserviceaccount.com</urn:serviceAccountEmail>  
            <urn:fields>alternateLink,labels</urn:fields>  
        </root>  
    </soapenv:Body>  
</soapenv:Envelope>
```

JSON Request

```
{  
    "accessToken": "AQV068KoSLBPT8U",  
    "apiUrl": "https://api.linkedin.com",  
    "publicUrl": "http://www.linkedin.com/pub/wso2connector-abdera/87/998/935",  
    "memberId": "12323"  
}
```

POX Request

```
<createExpense>  
    <arbitraryPassword></arbitraryPassword>  
    <apiUrl>https://sansu.freshbooks.com</apiUrl>  
    <authenticationToken>  
        c361a63c7456519412fa8051ea605a6d</authenticationToken>  
    <staffId>1</staffId>  
    <status>1</status>  
    <vendor>Sun Tzu Auto</vendor>  
    <categoryId>5</categoryId>  
    <projectId>19410</projectId>  
    <date>2014-05-22</date>  
    <clientId>99962</clientId>  
    <compoundTax></compoundTax>  
    <amount>2000</amount>  
    <tax1Name>VAT</tax1Name>  
    <tax1Amount>200</tax1Amount>  
    <tax1Percent>10</tax1Percent>  
    <tax2Name>GST</tax2Name>  
    <tax2Amount>200</tax2Amount>  
    <tax2Percent>10</tax2Percent>  
    <notes>Expense Test</notes>  
    <compoundTax>true</compoundTax>  
</createExpense>
```

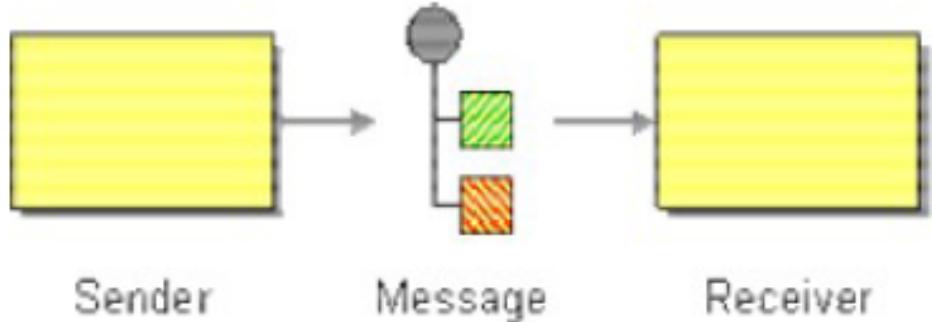
Message Mediation

- A **mediator** is **the basic, full-powered message processing unit** in the ESB.
- A mediator can **take a message, carry out some predefined actions** on it, and output the modified message.
- Usually, a **mediator** is configured using **XML**.
- Different **mediators** have their **own XML configurations**.
- At the **run-time**, a **message is injected** in to the mediator with the **ESB run-time information**.
- Then this mediator **can do virtually anything** with the message.
- A user can write a mediator and put it into the ESB.

Messaging

- **Messaging** is a form of *loosely coupled* distributed communication.
- ‘Communication’ can be understood as an exchange of messages between software components.
- **Message-oriented Middleware (MOM)** attempt to relax tightly coupled communication (such as TCP network sockets, CORBA or RMI) by the introduction of an “intermediary component”.
- **MOMs** allow software components to communicate ‘indirectly’ with each other.
- Benefits of **Messaging** include message senders **not needing** to have precise **knowledge of their receivers**.
- Thus any data that is to be transmitted via a messaging system must be converted into one or more **messages**

Message Structure

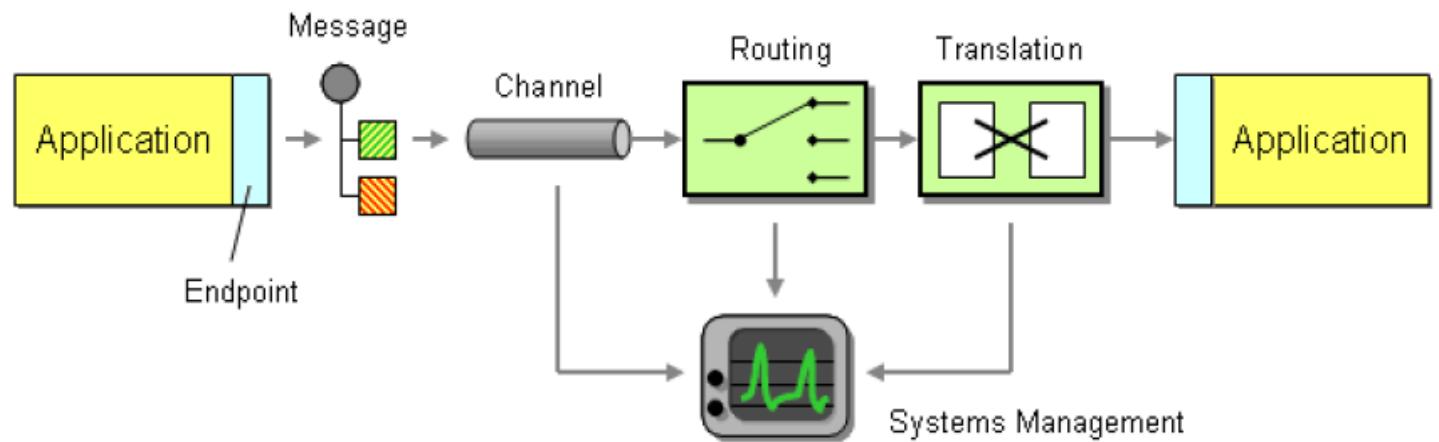


- Message consist of two basic parts.
- **Header**
 - Describes the data being transmitted, its origin, its destination, MessageID, Timestamp, Priority, Delivery mode, Message type and etc.
- **Body**
 - The data being transmitted; generally ignored by the messaging system and simply transmitted as-is.
- There are different kind of messages.
 - JMS Message
 - .NET Message
 - SOAP Message
- **Messaging play major role** in Enterprise Application Integration.

Enterprise Integration Patterns

There are 7 root patterns

- 1) Messaging
- 2) Message Channel
- 3) Message
- 4) Pipes and Filters
- 5) Message Router
- 6) Message Translator
- 7) Message Endpoint



Enterprise Integration Patterns - Messaging

- **Messaging**

- ✓ Is a technology that enables **high speed**, “**asynchronous**”, program-to-program communication with **reliable delivery**.
- ✓ Message itself is simply some sort of data structure such as a **string**, a byte array, a record, or an object



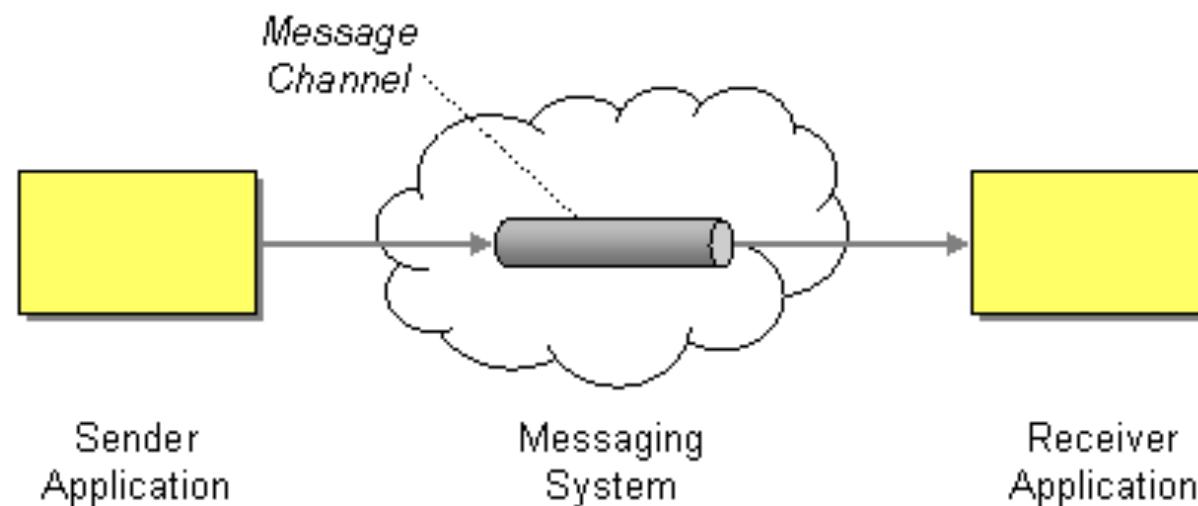
- **Messages**

- ✓ Programs communicate by sending *packets of data* called “**messages**” to each other.
- ✓ First process **marshals** the data into a byte stream and copy it from the first process to the second process.
- ✓ The second process **unmarshal** the data back to its original form.

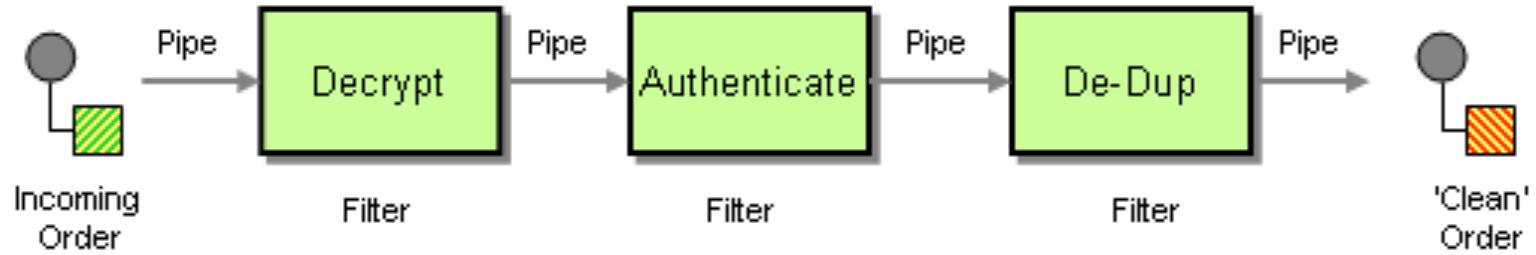
Enterprise Integration Patterns – Message channel

- **Message Channel**

- ✓ Messaging applications transmit data through a **Message Channel**, a virtual pipe that connects a sender to a receiver
- ✓ The media that can move data from one application to the other

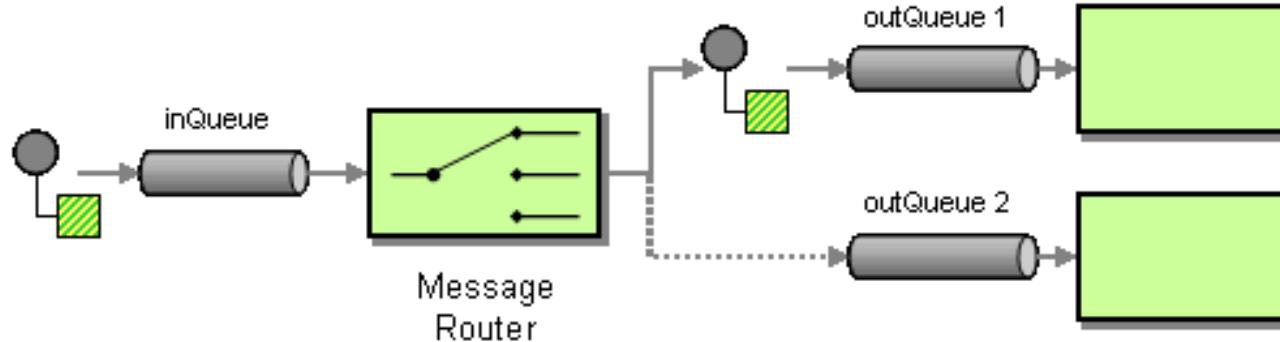


Enterprise Integration Patterns – Pipes and Filters



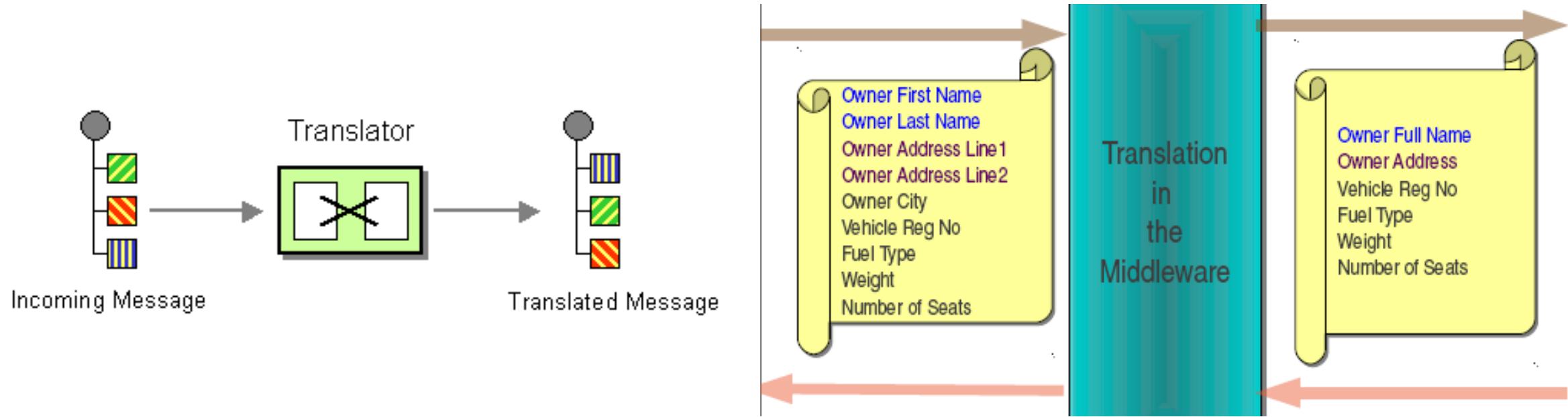
- Use the **Pipes and Filters** architectural style to **divide a larger processing task into a sequence of smaller, independent processing steps** (Filters)
- That are connected by channels (Pipes).
- Each filter exposes a very simple interface.
- The **connection between filter** and pipe is sometimes called **port**. In the basic form, each filter component has **one input port** and **one output port**.
- We can add new filters, omit existing ones or rearrange them into a new sequence.

Message Router Pattern



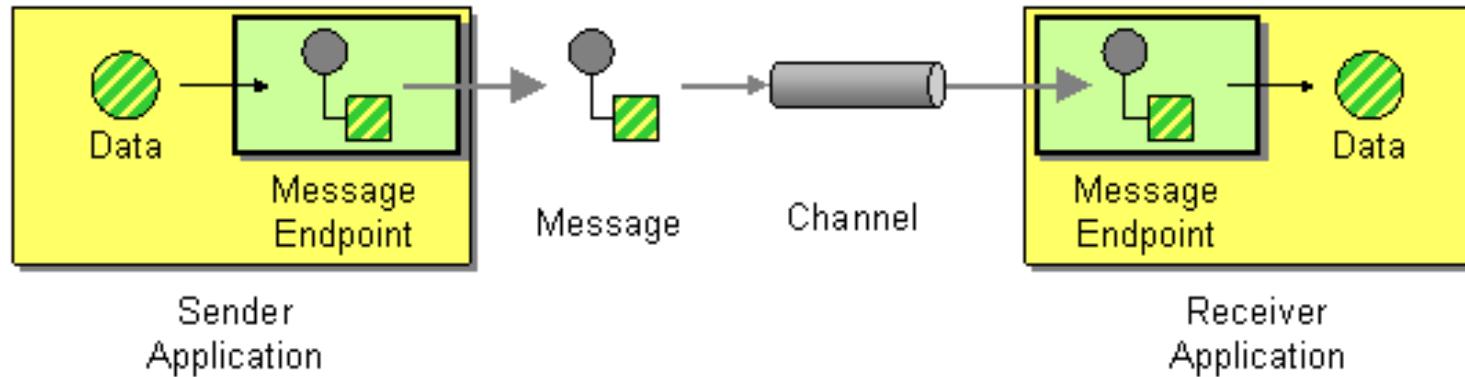
- Consumes a Message from **one Message Channel** and **republishes** it to a different Message Channel depending on a **set of conditions**.
- A key property of the **Message Router** is that **it does not modify the message contents**.
- If **new message types are defined**,
 - new processing components are added,
 - or the routing rules change,
 - we need to change only the Message Router logic and all other components remain unaffected

Message Translator Pattern



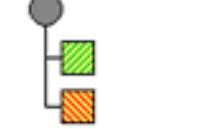
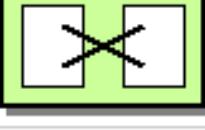
- The ***Message Translator*** is the messaging equivalent of the ***Adapter pattern***.
- An **adapter** converts the **interface** of a component into a **another interface** so it can be used in a different context.
- **XSLT** Mediators can be used for message transformation.

Message Endpoint Pattern



- Connect an application to a messaging channel using a **Message Endpoint**.
- A client of the messaging system that the application can then use to send or receive messages.
- It is the **endpoint** that receives a message, **extracts the contents, and gives them to the application** in a meaningful way.

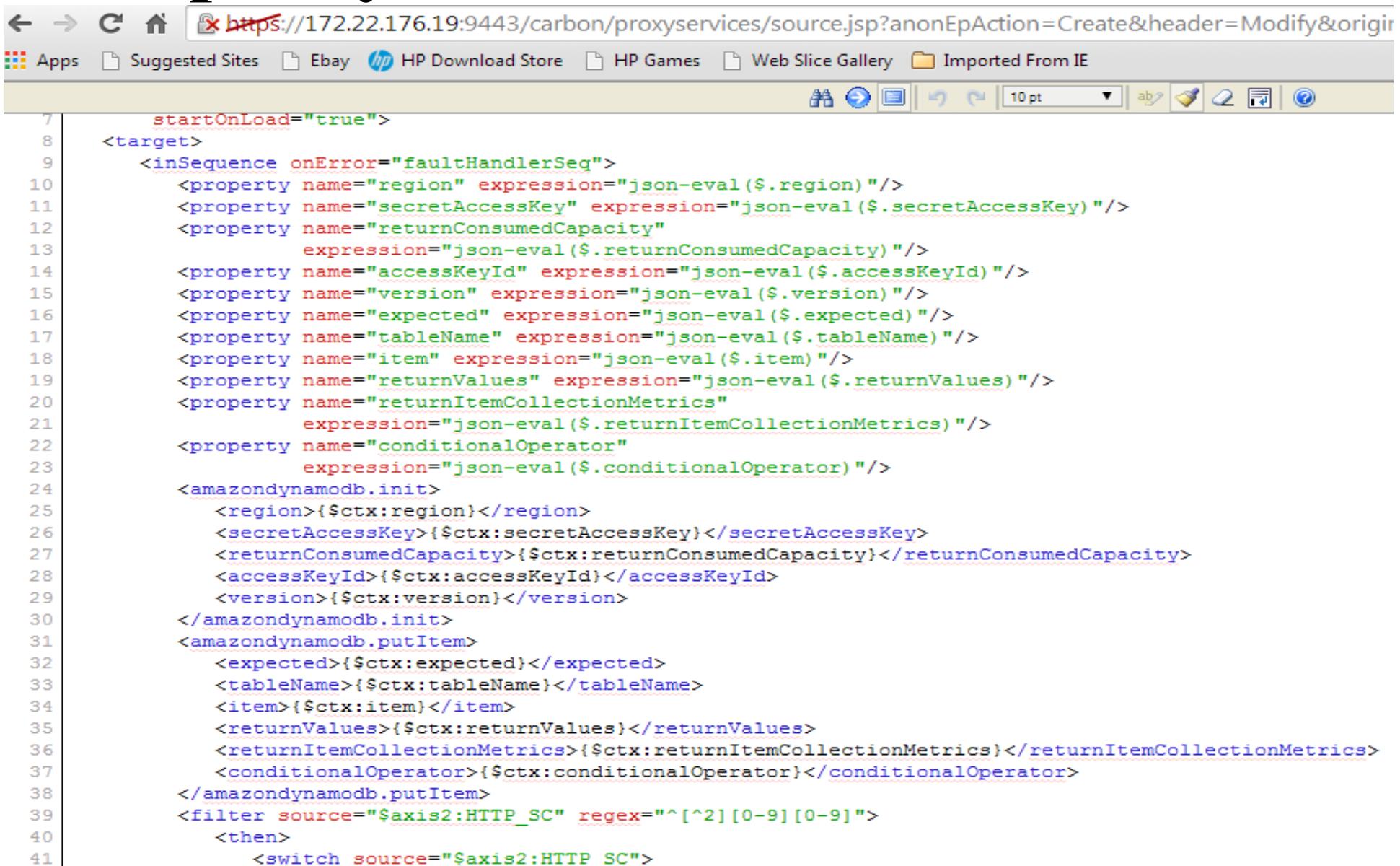
In a nutshell

	Message Channel	How does one application communicate with another using messaging?
	Message	How can two applications connected by a message channel exchange a piece of information?
	Pipes and Filters	How can we perform complex processing on a message while maintaining independence and flexibility?
	Message Router	How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?
	Message Translator	How can systems using different data formats communicate with each other using messaging?
	Message Endpoint	How does an application connect to a messaging channel to send and receive messages?

Proxy Services

- **Pass Through Proxy** - Forwards messages to the endpoint without performing any processing on them. This proxy service is useful as a catch-all, so that messages that do not meet the criteria to be handled by other proxy services are simply forwarded to the endpoint.
- **Secure Proxy** - Uses WS-Security to process incoming requests and forward them to an unsecured backend service.
- **WSDL Based Proxy** - A proxy service that is created from the remotely hosted WSDL of an existing web service. The endpoint information is extracted from the WSDL.
- **Logging Proxy** - Logs all the incoming requests and forwards them to a given endpoint. It can also log responses from the backend service before routing them to the client.
- **Transformer Proxy** - Transforms all the incoming requests using XSLT and then forwards them to a given endpoint. It can also transform responses from the backend service.
- **Custom Proxy** - A custom proxy service in which you customize the sequences, endpoints, transports, and other QoS settings.

Custom proxy service.



The screenshot shows a web browser window with the URL <https://172.22.176.19:9443/carbon/proxyservices/source.jsp?anonEpAction=Create&header=Modify&origin>. The page displays XML code for a custom proxy service configuration. The code includes sections for target properties, Amazon DynamoDB initialization, and putting an item. The browser interface includes a toolbar with various icons and a status bar at the bottom.

```
7     startOnLoad="true">
8     <target>
9       <inSequence onError="faultHandlerSeq">
10      <property name="region" expression="json-eval($.region)"/>
11      <property name="secretAccessKey" expression="json-eval($.secretAccessKey)"/>
12      <property name="returnConsumedCapacity"
13        expression="json-eval($.returnConsumedCapacity)"/>
14      <property name="accessKeyId" expression="json-eval($.accessKeyId)"/>
15      <property name="version" expression="json-eval($.version)"/>
16      <property name="expected" expression="json-eval($.expected)"/>
17      <property name="tableName" expression="json-eval($.tableName)"/>
18      <property name="item" expression="json-eval($.item)"/>
19      <property name="returnValues" expression="json-eval($.returnValues)"/>
20      <property name="returnItemCollectionMetrics"
21        expression="json-eval($.returnItemCollectionMetrics)"/>
22      <property name="conditionalOperator"
23        expression="json-eval($.conditionalOperator)"/>
24     <amazonynamodb.init>
25       <region>{$ctx:region}</region>
26       <secretAccessKey>{$ctx:secretAccessKey}</secretAccessKey>
27       <returnConsumedCapacity>{$ctx:returnConsumedCapacity}</returnConsumedCapacity>
28       <accessKeyId>{$ctx:accessKeyId}</accessKeyId>
29       <version>{$ctx:version}</version>
30     </amazonynamodb.init>
31     <amazonynamodb.putItem>
32       <expected>{$ctx:expected}</expected>
33       <tableName>{$ctx:tableName}</tableName>
34       <item>{$ctx:item}</item>
35       <returnValues>{$ctx:returnValues}</returnValues>
36       <returnItemCollectionMetrics>{$ctx:returnItemCollectionMetrics}</returnItemCollectionMetrics>
37       <conditionalOperator>{$ctx:conditionalOperator}</conditionalOperator>
38     </amazonynamodb.putItem>
39     <filter source="$axis2:HTTP_SC" regex="^[^2][0-9][0-9]">
40       <then>
41         <switch source="$axis2:HTTP_SC">
```

Sample proxy services

The screenshot shows the WSO2 Enterprise Service Bus Management Console interface. The top navigation bar includes the WSO2 logo, 'Enterprise Service Bus', 'Management Con...', 'Signed-in as: admin@carbon.super | Sign-out | Docs |', and a help icon. The left sidebar has tabs for Main (selected), Monitor, Configure, and Tools. Under Main, the 'Manage' section is expanded, showing 'Services' (List, Add, Proxy Service selected), 'Service Bus' (Sequences, Scheduled Tasks, Templates, Endpoints), 'Local Entries', 'Message Processors', 'Message Stores', 'Priority Executors', 'APIs', 'Source View', 'Connectors' (List, Add), and 'Secure Vault Tool'. The main content area is titled 'Deployed Services' and displays a message: '19 active services. 19 deployed service group(s.)'. It includes filters for 'Service Type' (ALL) and 'Service' search, and pagination controls ('<< first <prev 1 2 next > last >>'). Below these are links for 'Select all in this page', 'Select all in all pages', and 'Select none', followed by a 'Delete' button. A table lists five services:

	Service Name	Type	Status	WSDL1.1	WSDL2.0	Action	Design View	Source Vi
<input type="checkbox"/>	amazondynamodb_putitem	proxy	Unsecured	WSDL1.1	WSDL2.0	Try this service	Design View	Source Vi
<input type="checkbox"/>	amazoneses_listIdentities	proxy	Unsecured	WSDL1.1	WSDL2.0	Try this service	Design View	Source Vi
<input type="checkbox"/>	bugherd_addTaskComment	proxy	Unsecured	WSDL1.1	WSDL2.0	Try this service	Design View	Source Vi
<input type="checkbox"/>	bugherd_createProjectTask	proxy	Unsecured	WSDL1.1	WSDL2.0	Try this service	Design View	Source Vi
<input type="checkbox"/>	bugherd_deleteTaskAttachment	proxy	Unsecured	WSDL1.1	WSDL2.0	Try this service	Design	Source Vi

Connector Template Implementation

```
<template name="updateProjectTask" xmlns="http://ws.apache.org/ns/synapse">
    <parameter name="taskId" description="ID of the Project Task to be updated."/>
    <parameter name="task" description="The task JSON object to be sent to the API."/>
    <sequence>
        <property name="uri.var.task" expression="$func:task"/>
        <property name="uri.var.taskId" expression="$func:taskId"/>

        <payloadFactory media-type="json">
            <format>
                {"task":$1}
            </format>
            <args>
                <arg expression="get-property('uri.var.task')"/>
            </args>
        </payloadFactory>

        <property name="DISABLE_CHUNKING" value="true" scope="axis2"/>

        <call>
            <endpoint>
                <http method="put" uri-template="{uri.var.apiUrl}/api_v2/projects/{uri.var.projectId}/tasks/{uri.var.taskId}.json"/>
            </endpoint>
        </call>

        <!-- Remove custom Headers from the Response -->
        <header name="Via" scope="transport" action="remove"/>
        <header name="ETag" scope="transport" action="remove"/>
        <header name="X-Runtime" scope="transport" action="remove"/>
        <header name="X-Powered-By" scope="transport" action="remove"/>
        <header name="X-Rack-Cache" scope="transport" action="remove"/>
        <header name="X-Request-Id" scope="transport" action="remove"/>
        <header name="X-Frame-Options" scope="transport" action="remove"/>
        <header name="X-UA-Compatible" scope="transport" action="remove"/>
        <header name="X-XSS-Protection" scope="transport" action="remove"/>
```

ESB Connector

➤ Aggregation of **mediators** => is called **sequence**

➤ Aggregation of **sequences** => is called **template**

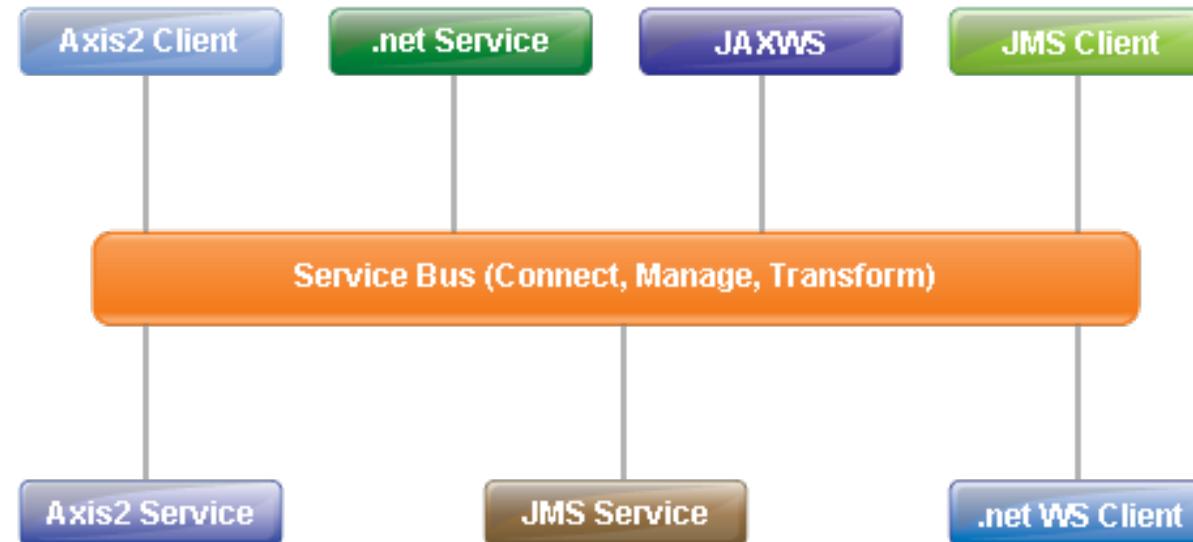
➤ Aggregation of **templates** => is called **connector**

The screenshot shows the WSO2 ESB Manager interface. On the left, there is a sidebar with various management options: Manage, Services (selected), List, Add, Proxy Service, Service Bus, Sequences, Scheduled Tasks, Templates, Endpoints, Local Entries, and Message Processors. The 'Services' option is highlighted with a purple background. The main area is titled 'Connectors' and shows a 'Connector List'. The table has columns for Library Name, Package, Description, Status, Actions, and Buttons for Delete and Download.

Library Name	Package	Description	Status	Actions
bugherd	org.wso2.carbon.connector	wso2 connector library for BugHerd	<input checked="" type="checkbox"/> Enabled	
amazonses	org.wso2.carbon.connector	wso2 connector library for AmazonSES	<input checked="" type="checkbox"/> Enabled	
amazondynamodb	org.wso2.carbon.connector	Amazon Dynamo DB connector library	<input checked="" type="checkbox"/> Enabled	
magento	org.wso2.carbon.connector	Magento connector library	<input checked="" type="checkbox"/> Enabled	

Service Bus and Mediation

- Service Bus is the **common communication channel** that is used by all the parties in messaging.
- Mediation is the process of facilitated communication between parties by providing intermediary conflict resolutions.



References

- <https://docs.wso2.com/display/ESB480/Using+a+Connector>
- <https://docs.wso2.com/display/ESB480/JIRA+Connector>
- <https://docs.wso2.com/display/ESB480/Managing+Connectors+in+Your+ESB+Instance>
- <https://docs.wso2.com/display/ESB480/Mediators>



Architectural Structures and Views

Software Architecture
3rd Year – Semester 1
Lecture 8

Architectural Structures and Views

- **View**

A view is a representation of a coherent set of architectural elements, as written by and read by system stakeholders. It consists of a representation of a set of elements and the relations among them

- **Structure**

The set of elements itself, as they exist in software or hardware

Purpose:

- Restrict our attention at any one moment to one (or a small number) of the software system's structures.
- To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment

Example:

Software Module - View Vs. Structure

- **View**

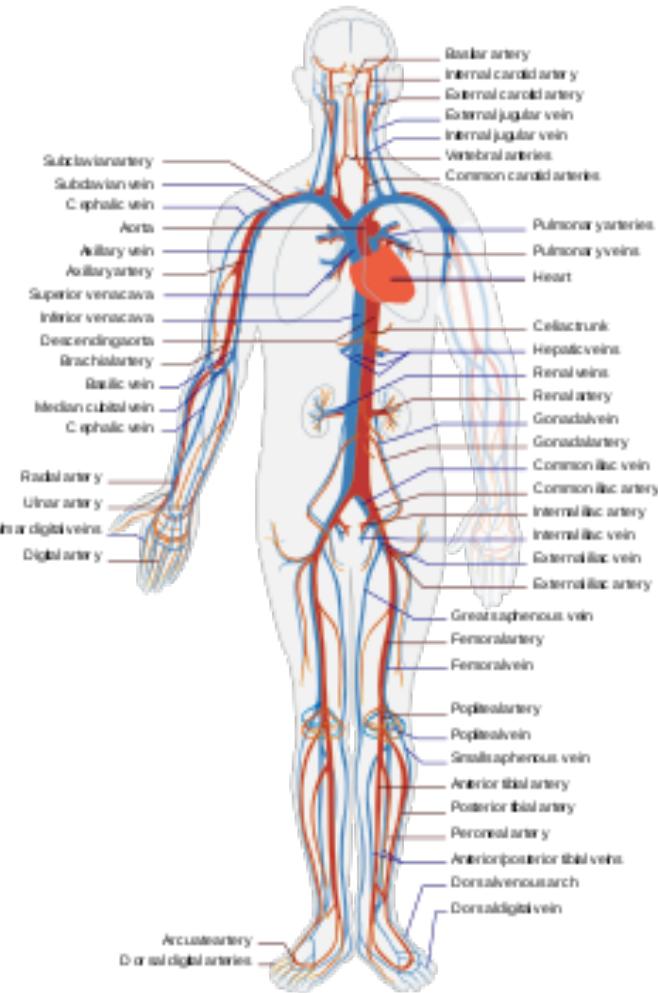
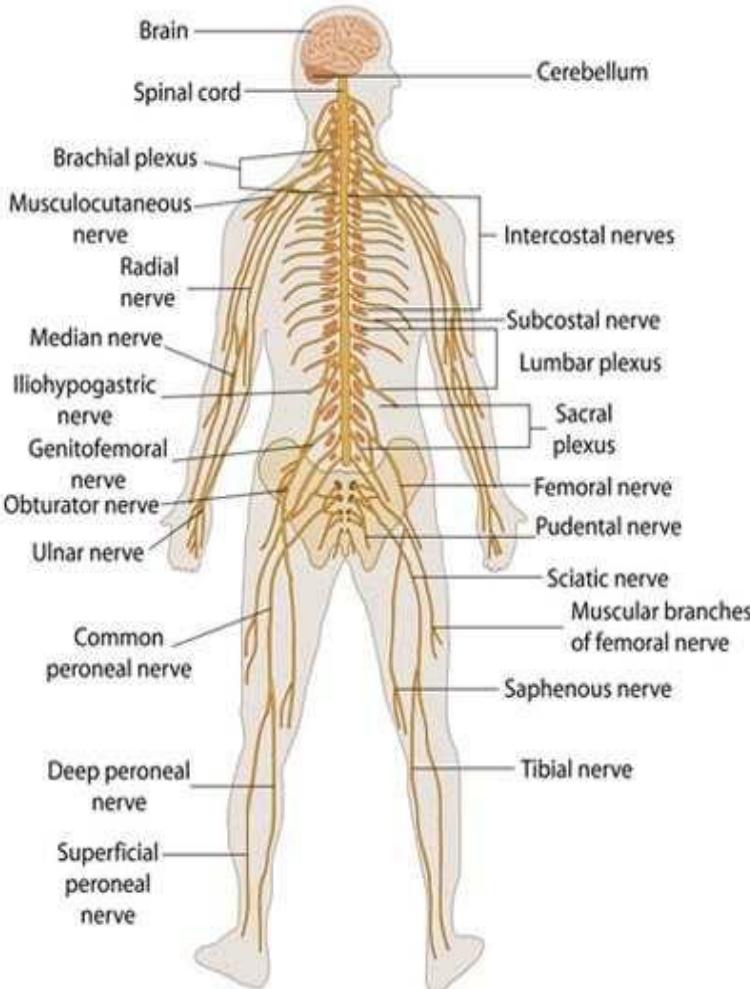
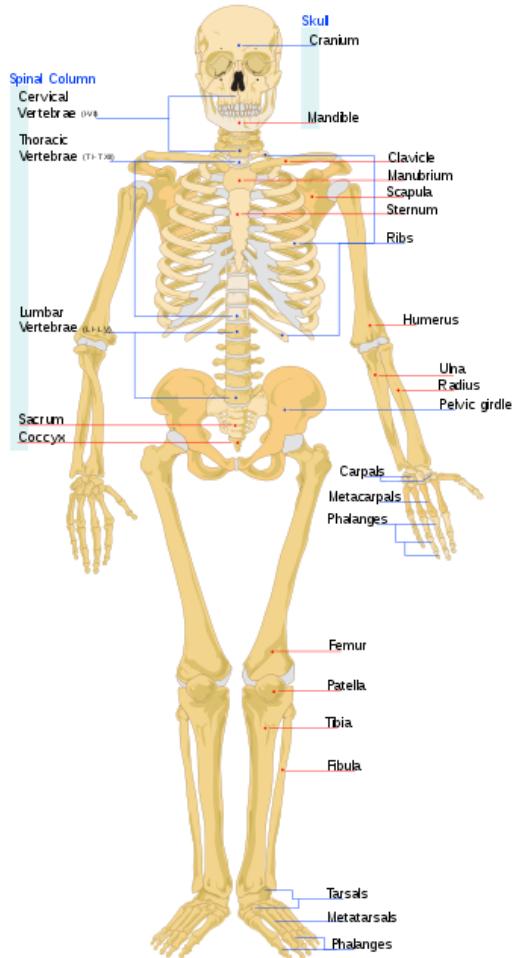
A module view is the representation of that structure, as documented by and used by some system stakeholders

- **Structure**

A module structure is the set of the system's modules and their organization

Note: These terms are often used interchangeably, but we will adhere to these definitions

An example from elsewhere...

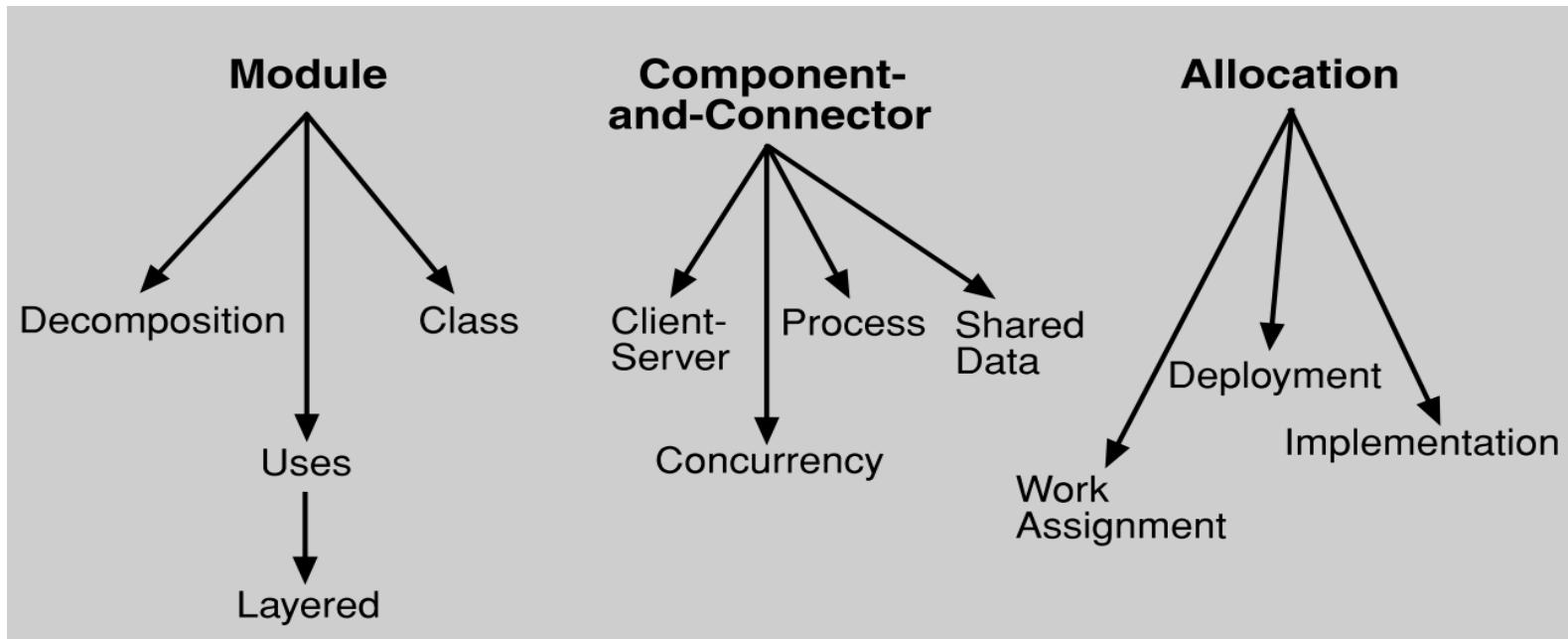


Architectural Considerations

- How is the system to be structured as a set of code units?
 - Modules
- How is the system to be structured as a set of elements that have runtime behavior and interactions?
 - Components
 - Connectors
- How is the system to relates to non-software structures in its environment?
 - CPU, File System, Network
 - Development Team (non-software)

Software Structures

- Module Structures
- Component & Connector Structures
- Allocation Structures



Module Structures

Elements are modules - which are units of implementation

- What is the primary functional responsibility assigned to each module?
- What other software elements is a module allowed to use?
- What other software does it actually use?
- **Decomposition:** Shows how larger modules are decomposed into smaller ones recursively
- **Uses:** The units are; modules, procedures or resources on the interfaces of modules. The units are related by the uses relation
- **Layers:** Structured into layers by using relations
- **Class (Generalization):** Shows the inherits-from or is-an-instance-of relations among the modules

Component-and-connector Structures

Elements are runtime components and connectors. The relation is attachment, showing how the components and connectors are linked together

- What are the major executing components and how do they interact?
- What are the major shared data stores?
- Which parts of the system are replicated?
- How does data progress through the system?
- What parts of the system can run in parallel?
- How can the system's structure change as it executes?
- **Process (or communicating processes):** Units are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations
- **Concurrency:** The units are components and the connectors are Logical threads, a logical thread is a sequence of computation that can be allocated to a separate physical thread
- **Shared Data (or repository):** This structure comprises components and connectors that create, store, and access persistent data
- **Client-Server:** The components are the clients and servers, and the connectors are protocols and messages

Allocation Structures

The relationship between the software elements and the elements in one or more external environments

- What processor does each software element execute on?
- In what files is each element stored during development, testing, and system building?
- What is the assignment of software elements to development teams?
- **Deployment:** Shows how software is assigned to hardware-processing (execution) and communication elements. Relations are allocated-to and migrates-to if the allocation is dynamic
- **Implementation:** How software elements (usually modules) are mapped to the file structure(s)
- **Work Assignment:** Assigns responsibility for implementing and integrating the modules to development teams

Summary of Structures of a System

<u>Software Structure</u>	<u>Relations</u>	<u>Useful for</u>
Decomposition	Is a submodule of; shares secret with	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control
Uses	Requires the correct presence of	Engineering subsets; engineering extensions
Layered	Requires the correct presence of; uses the services of; provides abstraction to	Incremental development; implementing systems on top of "virtual machines" portability
Class	Is an instance of; shares access methods of	In object-oriented design systems, producing rapid almost-alike implementations from a common template
Client-Server	Communicates with; depends on	Distributed operation; separation of concerns; performance analysis; load balancing
Process	Runs concurrently with; may run concurrently with; excludes; precedes; etc.	Scheduling analysis; performance analysis
Concurrency	Runs on the same logical thread	Identifying locations where resource contention exists, where threads may fork, join, be created or be killed
Shared Data	Produces data; consumes data	Performance; data integrity; modifiability
Deployment	Allocated to; migrates to	Performance, availability, security analysis
Implementation	Stored in	Configuration control, integration, test activities
Work Assignment	Assigned to	Project management, best use of expertise, management of commonality

Relating Software Structures to each other

- Different Structures provide different perspectives / concerns
- Different Structure are not independent – Elements of one Structure may be related to Elements of another
 - E.g. A module in the Decomposition Structure may be related to another structure in component-and-connector structure (to represent its runtime)
- Structures represent the primary engineering leverage points of an architecture

What Structures to Document?

- It's not practical (and often not needed) to document all Structures
 - Individual structures bring with them the power to manipulate one or more quality attributes – Architect should decide what are the key Architectural Qualities to achieve
 - Dominant Structures: There can be a structure that is dominant for a particular system which may need to satisfy the key requirements

Views

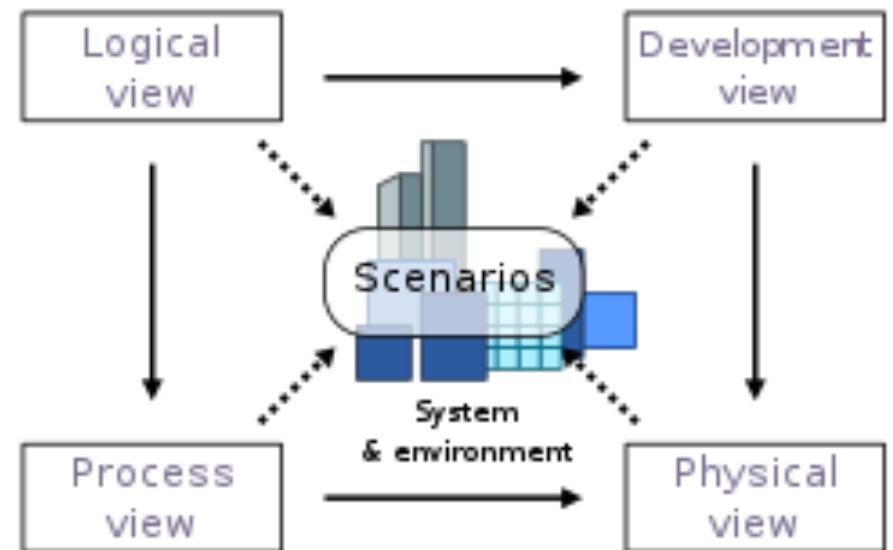
- The concept of a view, which you can think of as capturing a structure, provides us with the basic principle of documenting software architecture
- Different views support different goals and uses
 - For the Architect:
 - Deployment view will let you reason about your system's performance and reliability
 - Layered view will tell you about your system's portability
 - For the Developer:
 - Class view (Class Diagram) will tell you about Generalization/Inheritance and help you to reason about collections of similar behavior or capability

Stakeholders Vs. Architecture Documentation

<u>Stakeholder</u>	Module Views				<u>Component & Connector Views</u>	Allocation Views	
	Decomposition	Uses	Class	Layer		Deployment	Implementation
Project Manager	S	S		S		D	
Member of Development Team	D	D	D	D	D	S	S
Testers and Integrators		D	D		S	S	S
Maintainers	D	D	D	D	D	S	S
Product Line Application Builder		D	S	O	S	S	S
Customer					S	O	
End User					S	S	
Analyst	D	D	S	D	S	D	
Infrastructure Support	S	S		S		S	D
New Stakeholder	X	X	X	X	X	X	X
Architect (Current & Future)	D	D	D	D	D	D	S
<i>Information Level:</i>				D – Detailed O – Overview	S - Some X - Any		

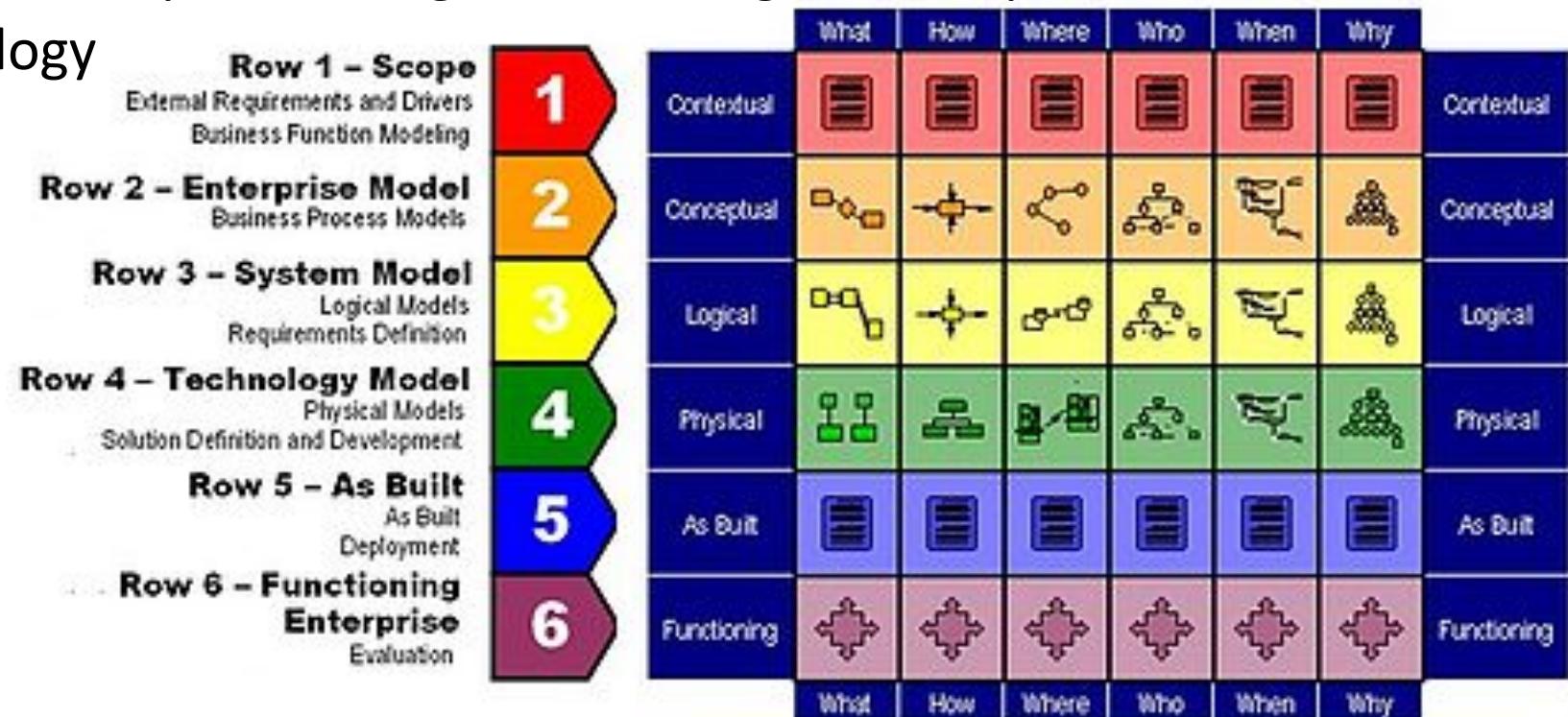
4+1 View Model

- The views are used to describe the system from the viewpoint of different stakeholders
- Concentrates on:
 - Logical View
 - Development View
 - Process View
 - Physical View
 - + Scenarios (Use Cases)



Enterprise Architectural Viewpoints

- Zachman Framework
 - Formal and structured way of viewing and defining an enterprise
 - An Enterprise Ontology



References

- <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch02lev1sec5.html>
- <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch09lev1sec3.html>
- <https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>
- <https://www.zachman.com/>



Architectural Patterns & Styles

**Software Architecture
3rd Year – Semester 1
Lecture 9**

What are Architectural Styles?

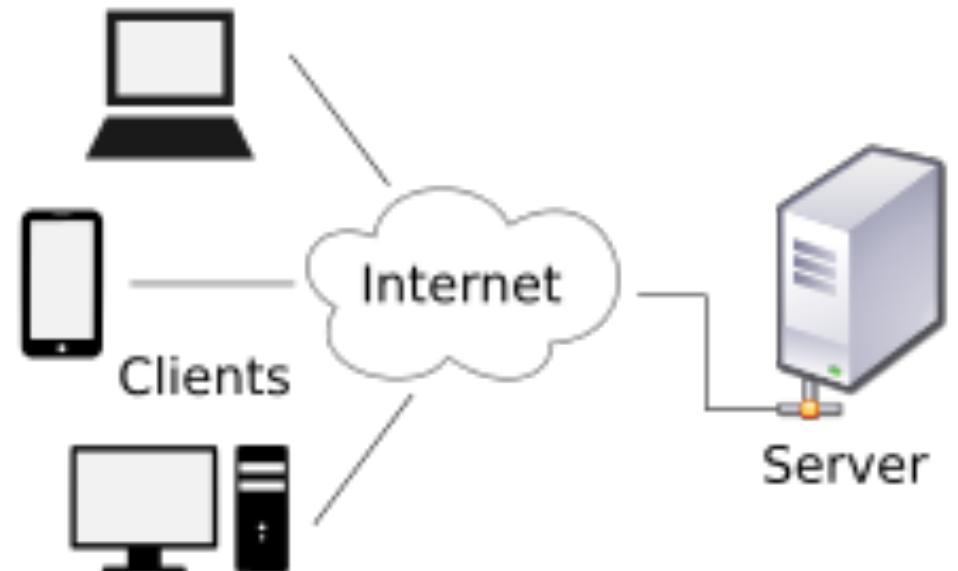
- An architectural style, sometimes called an architectural pattern, is a set of principles that shapes an application, a system or a system of systems.
- An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems.
- Provides a common language to understand systems – often not coupled with specific technologies/frameworks (Java, .NET, etc.) thus facilitates higher-level conversations.

Common Architectural Styles

- Monolithic
- Client-Server
- Component-based
- Layered
- N-Tier
- Object Oriented
- Blackboard
- Event Driven
- Domain Driven
- Plugin
- Microservice
- Peer-to-Peer
- Rule-based
- Service Oriented
- Message Bus
- Pipe and Filter
- REST
- Publish-Subscribe
- Cloud

Client-Server Architecture

- Server can serve multiple clients at a time
- Server usually provides a function, data, content, etc... to the client
- Centralized
- Client knows how to locate the server
- Connection could be HTTP, RPC, etc....

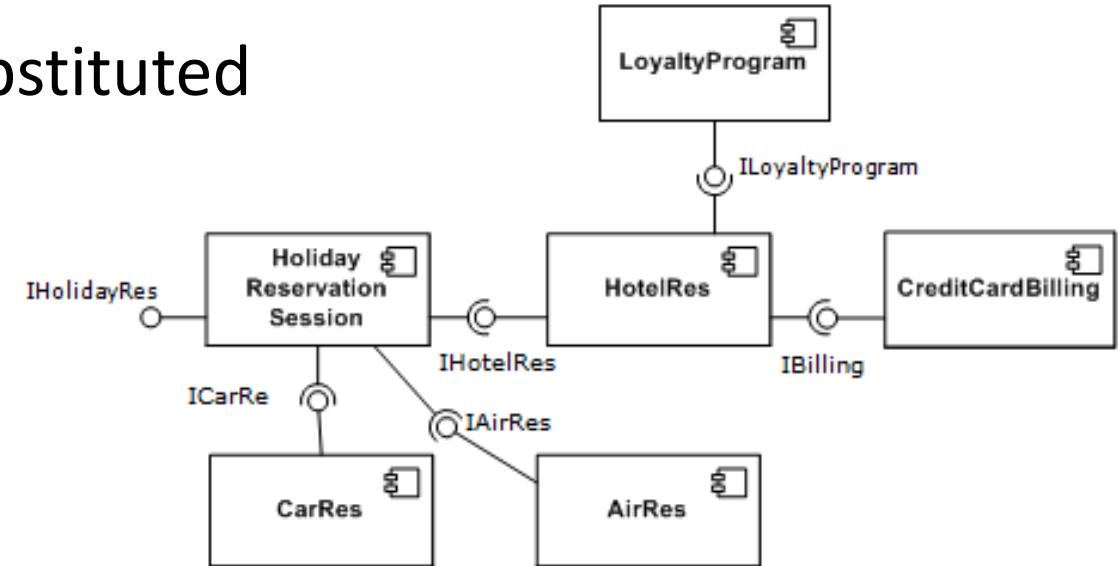


Client-Server Architecture in practice

- Examples:
 - Outlook Email - Outlook [Thick Client] connects to Microsoft Exchange Server via SMTP/POP
 - Gmail - Web Browser [Thin Client] connects to Google Mail Server via HTTP
- Advantages:
 - High Security (centralized)
- Disadvantages:
 - Single point of failure (server centric)
 - Maintenance & Downtime issues

Component-based Architecture

- Emphasize on Separation of Concerns
- Components are reusable
- Components are highly cohesive and loosely coupled
- Components are made to be substituted

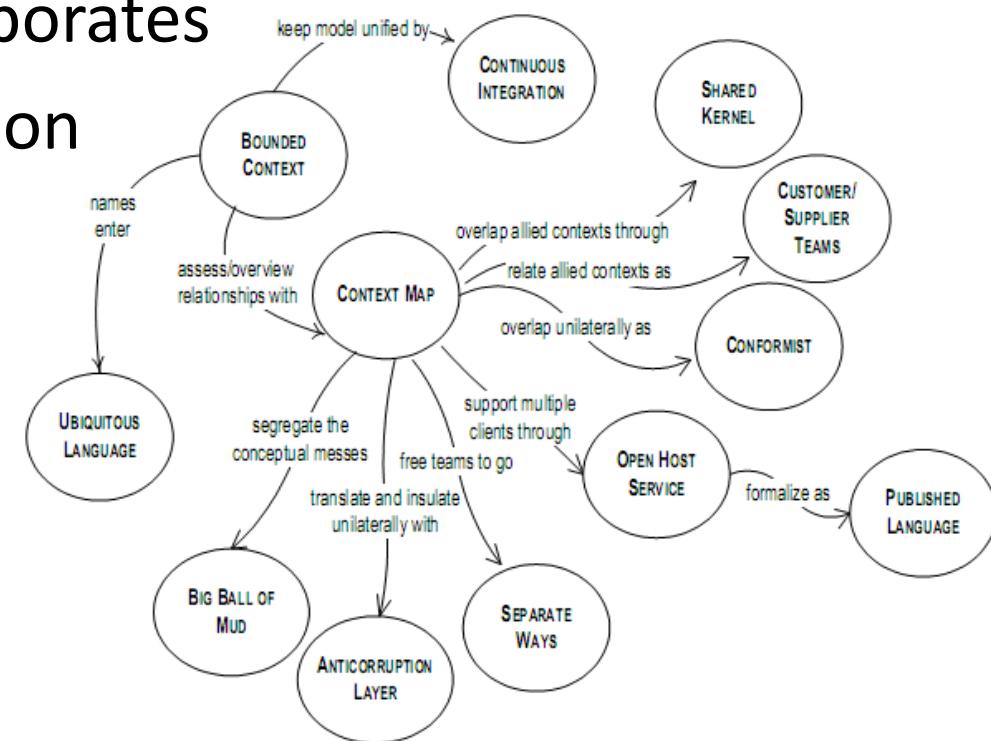


Component-based Architecture in practice

- Examples:
 - Java libraries (.jar files)
 - Windows OS .dll files
- Advantages:
 - Reusability (reduce development cost)
 - Extendibility – each component can be further adjusted
- Disadvantages:
 - Managing a large component base may be harder

Domain Driven Architecture

- Focus mainly on Domain & Logic around it
- Technical and Domain experts collaborates
- Ontology – Knowledge Representation

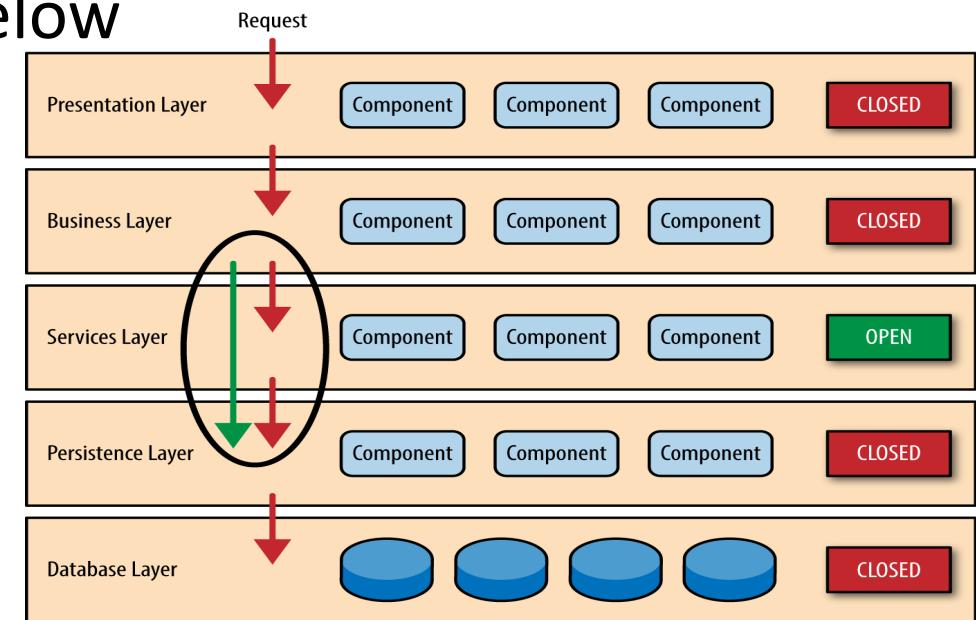


Domain Driven Architecture in practice

- Examples:
 - Web Content Management Systems
- Advantages:
 - Easy for the Domain experts
- Disadvantages:
 - When there is a larger team the system gets very complex and disorganized

Layered Architecture

- Groups related functions in to layers
- Layers are stacked on top of each other
- Typically components in one Layer can communicate with components in same layer or Layers below
- Layering helps Separation of Concerns

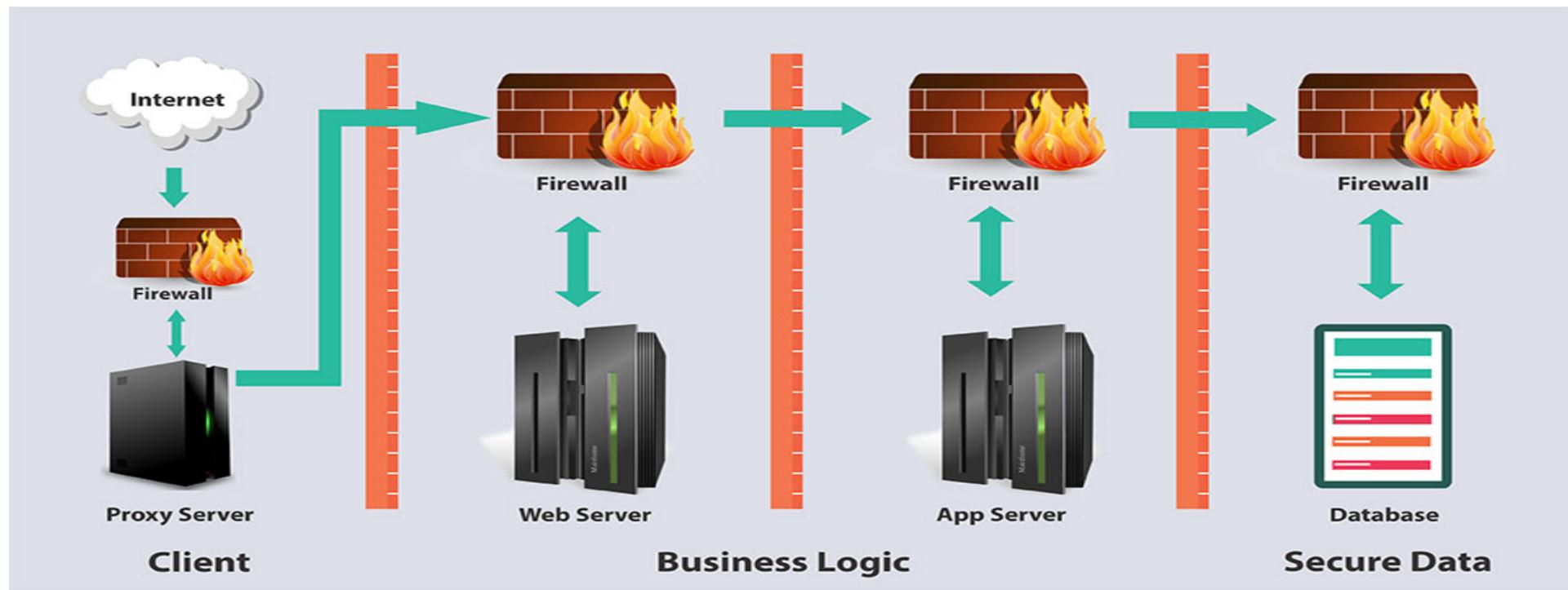


Layered Architecture in practice

- Examples:
 - Applications with Presentation, Service and Data Layers
 - TCP/IP
- Advantages:
 - Shares many advantages similar to Component Based Architecture
 - Layered Architecture can extend to N-Tier Model
- Disadvantages:
 - Components in bottom layers cannot communicate with top layers without Cyclic dependencies

N-Tier Architecture

- Can be considered as an extension to Layered Architecture with each layer having the ability of executing on different physical locations

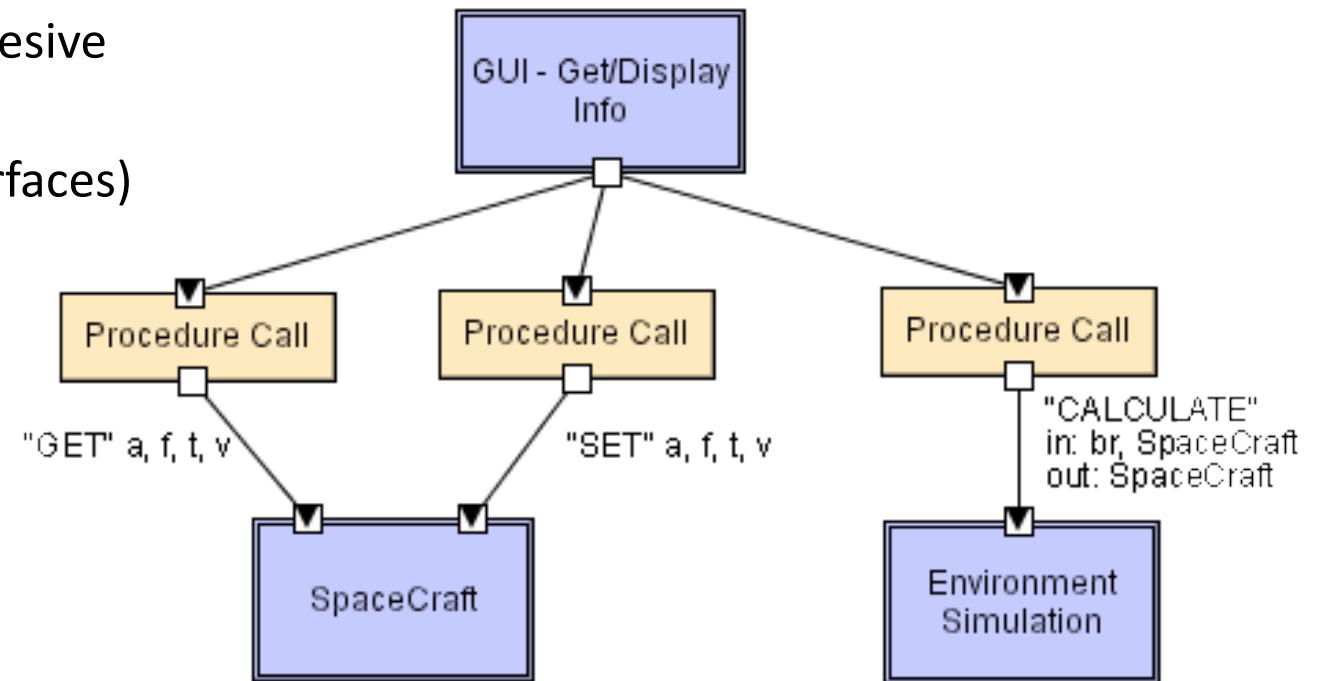


N-Tier Architecture in practice

- Examples:
 - Commercial Web Applications
- Advantages:
 - Shares many advantages similar to Layered Architecture
 - Can be scaled up to support increasing demand
 - Multiple nodes can be allocated to a tier that requires more resources
- Disadvantages:
 - Maintenance of multiple nodes
 - Data communication cost

Object Oriented Architecture

- Views the system as a set of cooperating objects
 - Components are Objects:
 - Objects contain data and behaviors
 - Objects are reusable and cohesive
 - Connectors are messages:
 - Method invocations (via interfaces)
- Based on Key Principles:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

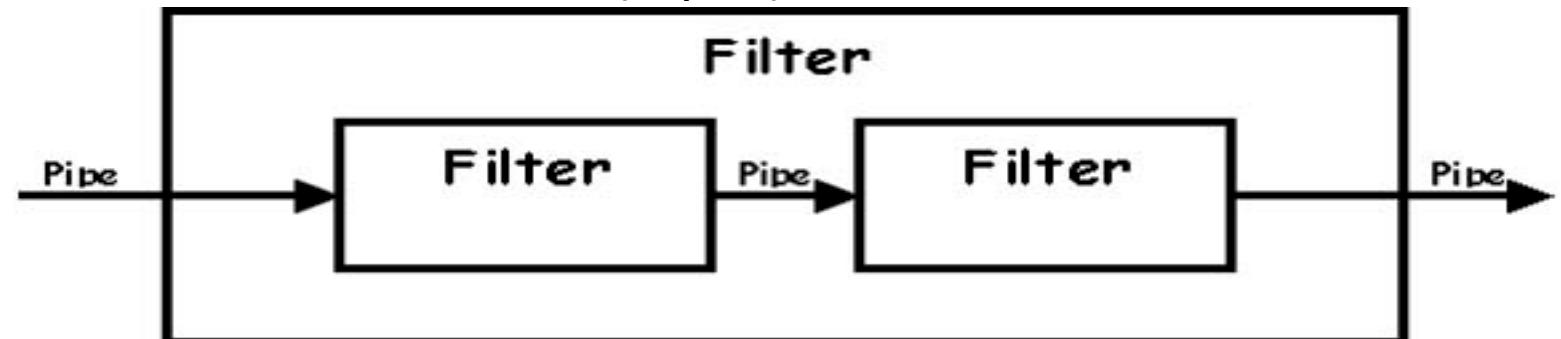


Object Oriented Architecture in practice

- Examples:
 - Most of modern applications
- Advantages:
 - Reusability
 - Extensibility
 - Highly Cohesive
 - Support of many Design/Development tools (e.g. UML)
- Disadvantages:
 - Speed
 - Effort (short term Cost implications)

Pipe and Filter Architecture

- Components are filters
 - Transform input data streams into output data streams
 - Possibly incremental production of output
 - Filters are independent (no shared state)
 - Filter has no knowledge of up- or down-stream filters
- Connectors are pipes
 - Pass data (output) of one filter to another (input)

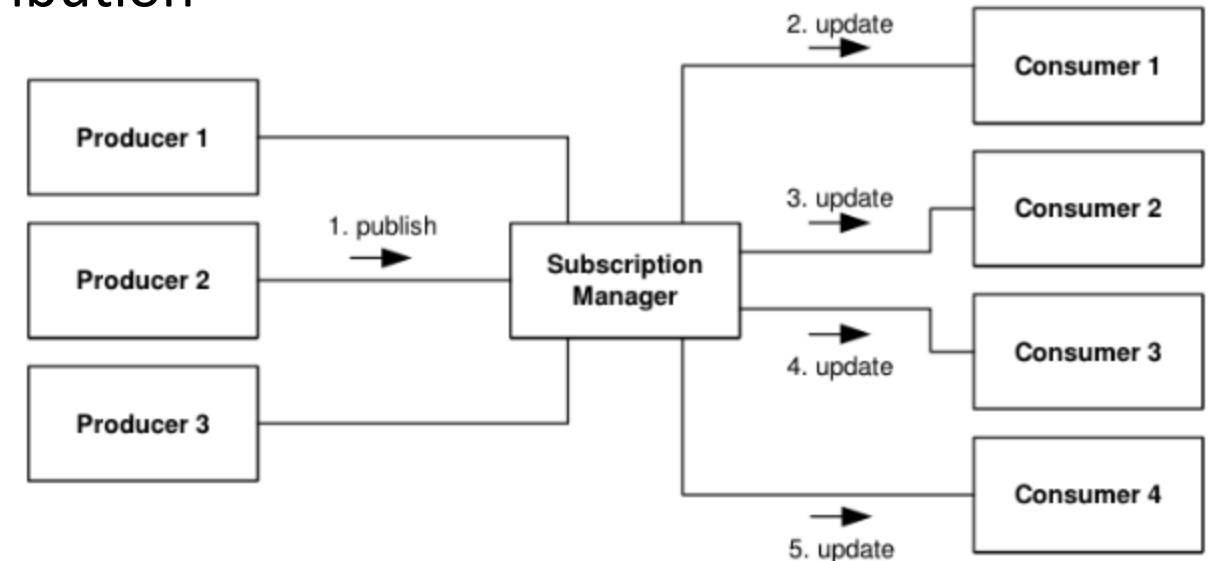


Pipe and Filter Architecture in practice

- Examples:
 - Unix/Linux Shell (Single Processing)
 - Compilers: consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation
- Advantages:
 - Can add/remove filters easily
 - Concurrent Execution: each filter can be implemented as a separate task and be executed in parallel with other filters
- Disadvantages:
 - Performance – may force a lowest common denominator on data transmission
 - No filter cooperation

Publish-Subscribe Architecture

- Subscribers register/deregister to receive specific messages or specific content
- Publishers broadcast messages to subscribers
 - May use Proxies to manage distribution
 - Topic based or Content based

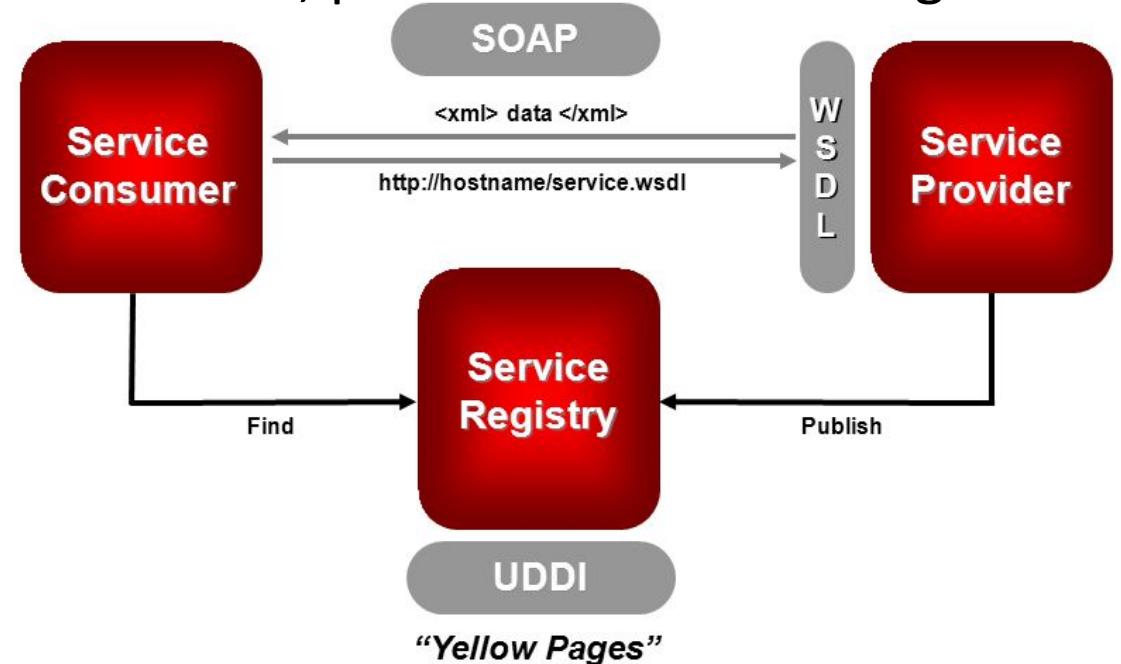


Publish-Subscribe Architecture in practice

- Examples:
 - Mobile News Alerts, Mobile App Push Notifications (e.g. GCM)
- Advantages:
 - Can avoid Polling for new messages (save bandwidth / power)
 - Can use queues / message bus to manage
 - Highly scalable
- Disadvantages:
 - Focus mainly on one-way communication
 - Decoupling of Subscriber from Publisher

Service Oriented Architecture

- Application functionality is provided as a set of remote services
 - Uses standard communication protocols
 - Service and Clients are independent of vendors, products and technologies
- Based on key principles:
 - Autonomous
 - Standard Service Contract
 - Abstracted & Encapsulated
 - Distributable & Discoverable
 - Etc.

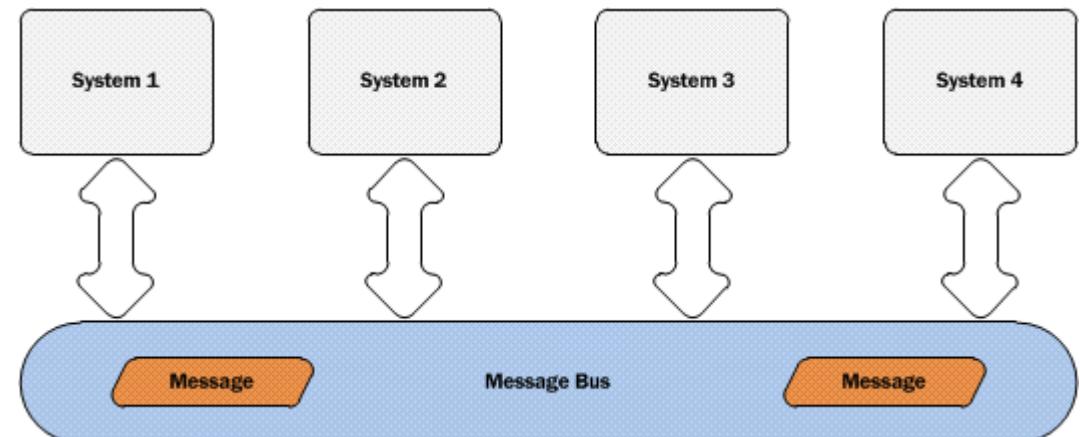


Service Oriented Architecture in practice

- Examples:
 - Many SOAP based Web Services
- Advantages:
 - Interoperability – can integrate products built with different technologies
 - Reusability
 - Widely used – well defined standards & tools
- Disadvantages:
 - Requires high availability

Message Bus Architecture

- Systems communicate with each other by passing messages [Asynchronous] via a common mediator [Bus]
- Widely used for Enterprise Application Integration (EAI)
 - Many SOA systems use message oriented middleware

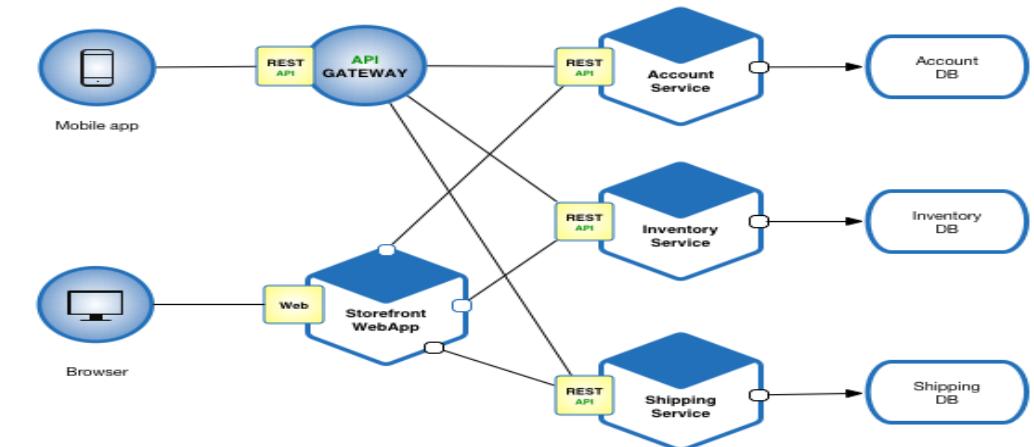


Message Bus Architecture in practice

- Examples:
 - Enterprise Service Bus (JBoss, Mule, WSO2, ...)
- Advantages:
 - Extensibility - Can easily add/remove applications from the bus
 - Can integrate with different technologies (via standard communication protocols)
 - Highly Scalable
- Disadvantages:
 - Requires middleware

Microservices Architecture

- Similar to Service Oriented Architecture (SOA)
 - Structures the system as a collection of loosely coupled services
- Decomposes services into much smaller but more cohesive computation units
- Uses lightweight protocols for communication

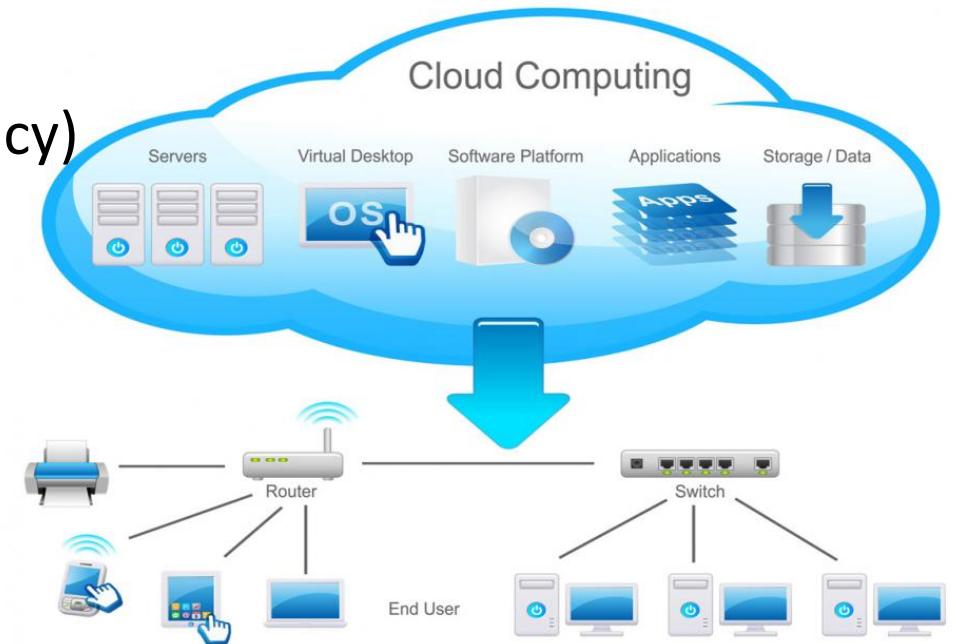


Microservices Architecture in practice

- Examples:
 - Netflix, Twitter, Amazon
- Advantages:
 - Having light weight communication protocols allow thin clients to connect
 - Supports better Continuous Integration & Delivery CI/CD
 - Easy to deploy and scale services independently
- Disadvantages:
 - Maintenance require special [Dev Ops] skills
 - Increase Network Communication within the System

Cloud Architecture

- Enables access to shared pool of resources
 - Can be rapidly provisioned to a new consumer
- Let the business focus on its core business instead of infrastructure
- Basic models of Could Computing:
 - IaaS, PaaS, SaaS (can achieve multi tenancy)



Cloud Architecture in practice

- Examples:
 - Amazon AWS based systems, SalesForce
- Advantages:
 - Elasticity – can scale up and down on-demand
 - Pay as you grow
- Disadvantages:
 - Security Concerns – All information with third parties

Combining Different Architecture Styles

- The overall Architecture of a System is most often a combination of multiple Architectural Styles
 - E.g. Layered combined with Object-Oriented with a Component-based deployment
- Factors involved:
 - Knowledge/Experience/Capabilities of the Development Team
 - Organizational Constraints (i.e. Data Security Vs. Cloud / SaaS)

References

- Software Architecture Patterns; by Mark Richard
- <https://msdn.microsoft.com/en-us/library/ee658117.aspx>
- Software Architecture: Foundations, Theory, and Practice; by Taylor & Medvidovic



Quality Attributes

**Software Architecture
3rd Year – Semester 1
Lecture 11**

Software Requirements

- Expectations from the Software/System
- Stakeholders/users/roles may have different requirements
 - End User: Needs the functionality work without issues, Wish the system to be easy to use
 - Project Manager: Low Cost
 - Admin/Security/Maintenance: May have their own specialized wishes
 - Sometimes there may be conflicts of requirements
- Needs Vs. Wants

Requirements...

Q: When to identify requirements?

A: At the start of the Project

Q: Who is responsible for requirements?

A: Business Analyst + Architect

Software Architect must identify “Architectural Significant Requirements” (ASR)

Software Architect should find a balance between needs & wants

Q: Why the Requirements should be clear?

A: Changes are costly, specially if the changes come in later stages

Types of Requirements

- Functional
 - What the system must do
 - How it must behave
 - How it must react
- Non-Functional
 - Quality Attributes
 - Business Requirements
 - Constraints

Exercise

- Write down the requirements for a Calculator on a Mobile Phone
- Functional
 - Basic + - * / operations
 - Support for Square root, etc...
 - Support for Brackets
- Non Functional
 - Should work for Android and iOS and later on Windows
 - Buttons should be placed in Number pad order
 - Support to add more operations later on (e.g. Log, Sin, Cos, ...)
 - Landscape UI Support ??
 - How many digits to display on the UI ?? (e.g. 0.33333333333333333333)

Definition of Quality & Functionality

- Functionality: The capability of the software product to provide functions which meets stated and implied needs when software is used under specified conditions
- Quality: The extend to which a product satisfies stated and implied needs when used under specified conditions

Functionality & Quality

- Functionality does not determine the Architecture
- For a given set of Functional Requirement you can create an endless amount of Architectures to Satisfy the requirements
- Functionality and Quality Attributes are Orthogonal (difference dimensions)

Non-Functional Requirements

- Non-Functional requirements are not directly linked to any specific function
- They are qualifications that typically cover business and system quality requirements and have a big influence on the architecture
- When they are forgotten at the beginning of a project, it often results in major problems in later stages of the project

Quality Attributes (overview)

- Qualifications of the functional requirements or of the overall product
 - Runtime Qualities – Performance
 - User Qualities – User Friendliness / UX
- Measurable and Testable properties of the system that are used to indicate how well the system satisfies the needs of the stakeholders
- There are many Quality Attributes and the list keeps growing...

Business Requirements

- Business or strategic decisions
 - Cost
 - Time to Market
 - System Lifetime
- Strategic Trade-offs are made
 - E.g. Building Brand New Software Vs. Purchasing Existing Software

Constraints

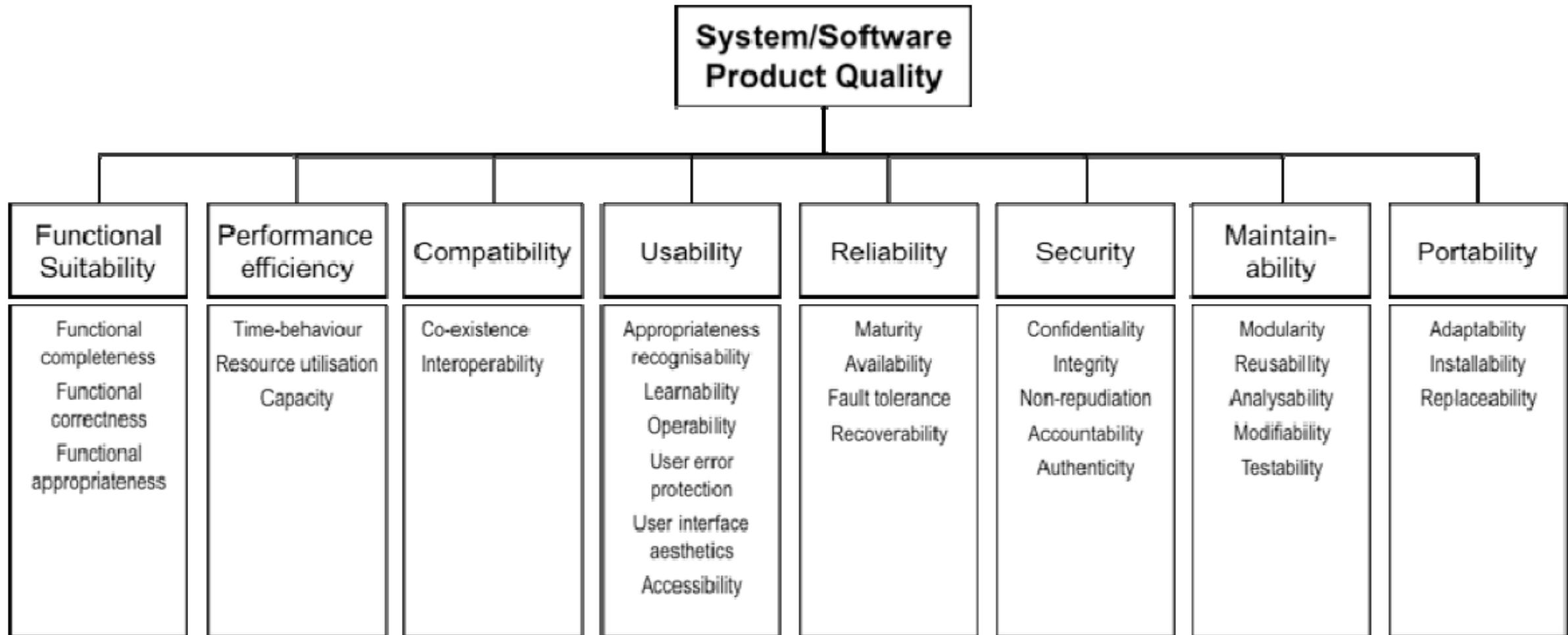
- Decisions/Agreements arrived beforehand
- No freedom to change and No Trade-offs can be made
 - E.g. Must use Open Source Software for Development

Software Architect has to make sure the Architecture Adheres to the constraints

Quality Attributes

Design Qualities	Run-time Qualities	System Qualities	User Qualities
<ul style="list-style-type: none">• Modifiability• Reusability• Maintainability	<ul style="list-style-type: none">• Performance• Reliability• Availability• Security• Scalability• Interoperability	<ul style="list-style-type: none">• Testability• Supportability	<ul style="list-style-type: none">• Usability• Accessibility

Quality Attribute – ISO 25010 Classification



Nature of Quality Attributes

- Quality Attributes may be related to one use case or part/module of the system or system as a whole
- Quality Attributes typically relate to different phases of the system life cycle such as design time and runtime qualities
- Quality Attributes may impact on each other
 - Performance is affected by almost all the other Quality Attributes

Role of an Architect on Quality Attributes

- Understand the Importance and Priority of the Quality Requirements of the System
- Evaluate and make Trade-Offs to meet the Quality Levels to satisfy the Stakeholders

Issues with Quality Attributes Definitions

- Grouped differently by different authors
- Overlapping (or Similar) Attributes
 - Integrity & Security
- Confusion with definition/meaning of Attributes
 - Performance: Responsiveness, Efficiency (CPU % Use)

Quality Attribute: Availability

- System Services are available when and where the users need to use them
- Proportion of time that the system is functional and working
- Affected by
 - System Errors
 - Maintenance Tasks
 - Infrastructure Failures
 - Malicious Attacks
 - System Load
 - Dependent Services
- Can be measured as a percentage of the total system downtime over a predefined period

Quality Attribute: Interoperability

- Ability of a system to exchange information successfully by communicating with other systems
- Interoperability Considerations
 - Syntactic: Ability to communicate between systems using specified data & communication protocols
 - Semantic: Ability to automatically interpret the information exchanged meaningfully and accurately in order to produce useful results

Quality Attribute: Modifiability

- Cost to make a change
 - Lower cost to change → higher in Modifiability
- Change Types
 - Functional Change
 - Environment Change (Host OS/Platform)
 - Protocols (Communication Protocols, etc...)
 - Dependencies (MySQL → Oracle Vs. MySQL → MongoDB)

Quality Attribute: Performance

- Performance is an indication of the responsiveness of a system to execute any action within a given time interval.
- It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event.
- Throughput is the number of events that take place within a given amount of time.

Quality Attribute: Reliability

- Ability of a system to remain operational over time
- Probability that a system will not fail to perform its intended functions over a specified time interval
- How users and other systems can be dependable of a given software, protocol, hardware, etc...
- Affected by other attributes
 - Availability
 - Accuracy
 - Predictability

Quality Attribute: Reusability

- Capability for components and subsystems to be suitable for use in other applications and in other scenarios
- Minimizes the duplication of components and also the implementation time
- E.g. Software Libraries

Quality Attribute: Scalability

- Ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged
 - Load
 - Functions
 - Geographic
- Methods
 - Horizontal: Add more Servers
 - Vertical: Add/Improve hardware (e.g. CPU) of the same Server

Quality Attribute: Security

- Capability of preventing Malicious Attacks
 - Virus
- Preventing unauthorized usage
- Protecting System Assets
 - Data
 - Hardware

Question:

Denial of Service: Is this a Security issue?
Is this only a Security issue?

Quality Attribute: Testability

- Create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met
 - How much can be tested?
 - How much time it takes to test?
- Testability makes it more likely that faults in a system can be isolated in a timely and effective manner

Quality Attribute: Usability

- How well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.
- Usability Considerations
 - How easy it is to learn the features of the system
 - How efficiently the user can use the system
 - How well the system handles user errors
 - How well the system adapts to user needs
 - To what degree the system gives the user confidence in the correctness of its actions

Architectural Attributes & Considerations

- Architectural quality attributes are also similar to system quality attributes, but concerned with aspects of the architecture itself.
 - Conceptual integrity : The architecture should do similar things in similar ways.
 - Correctness and completeness : Concerned with checking the architecture for errors and omissions.
 - Buildability : Allows the system to be completed by the available team in a timely manner and to be open to certain changes as development progresses

Quality Attribute Scenarios (QAS)

- Universal for Formal way to express Quality Attributes
- The goal of QAS is to capture and document unambiguous and testable requirements
 - Document similar to Use Case scenarios

NEXT LECTURE:

Template to express Quality Attributes

How to apply it

Concrete System Specific Qualities

Tactics

- An architectural tactic is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural decisions

NEXT LECTURE:

Ways to improve Quality Attributes

Tactics Framework



Quality Attribute Scenarios

**Software Architecture
3rd Year – Semester 1
Lecture 12**

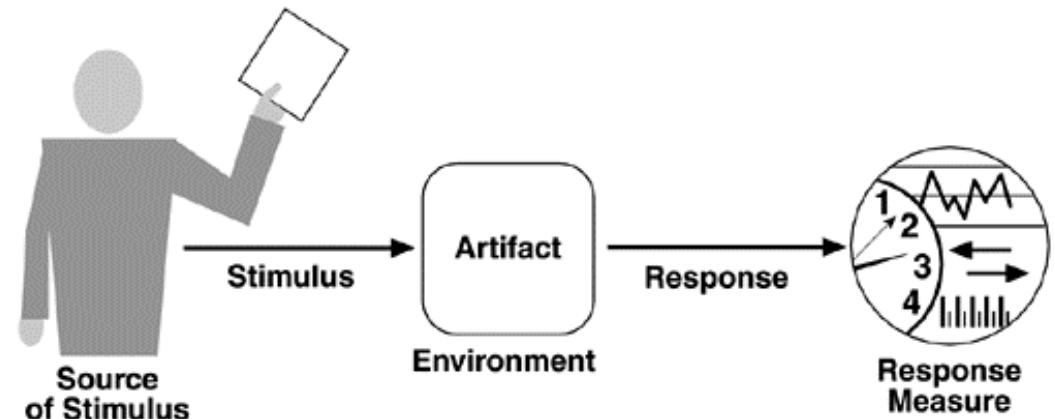
Quality Attribute Scenarios (QAS)

- A universal and formal way to express Quality Attributes.
- The goal of a QAS is to capture unambiguous and testable quality requirements in the same way as use case scenarios do for functional requirements.

General Vs. Concrete Quality Attribute Scenarios

- A general scenario is system independent and can, potentially, pertain to any system.
- A concrete scenario is specific to the particular system under consideration.
- Concrete scenarios are needed to make the quality requirements operational.
- A collection of concrete scenarios can be used as the quality attribute requirements for a system.

Template for QAS



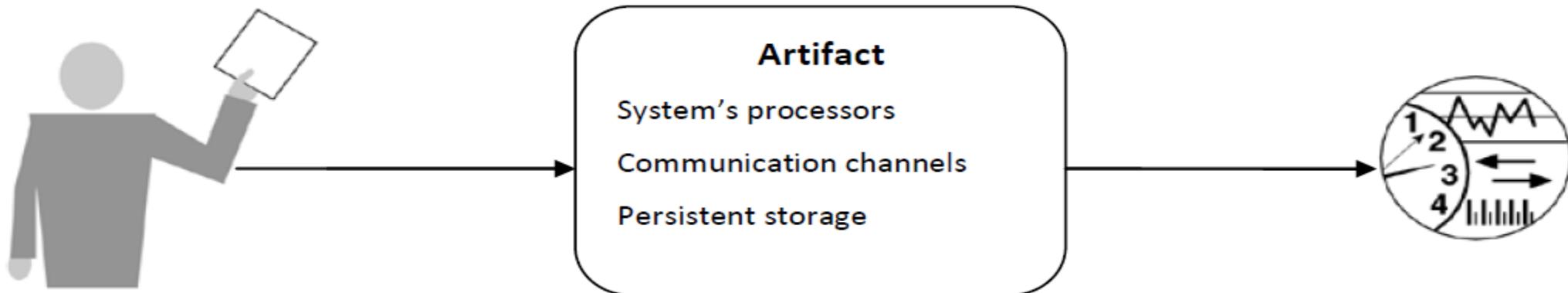
1. **SOURCE:** identifies the originator of the event or action: it can be a user or another system [*who*]
2. **STIMULUS:** describes the action or the external event that arrives at the system [*action*]
3. **ENVIRONMENT:** describes the external circumstances under which the quality requirement needs to be met [*when*]
4. **ARTIFACT:** indicates the part of the system to which the quality requirement applies [*what*]
5. **RESPONSE:** tells us how the system reacts to the stimulus [*result*]
6. **RESPONSE MEASURE:** provides metrics and quantifies the quality attribute [*measurement*]

Availability (QAS)

- Concerned with system failure and it's consequences
- Faults and failures
 - Using the wrong algorithm for computation
 - Miscalculation / incorrect output
- Concerns on failure
 - Frequency
 - Results
 - Non-operative time
 - Prevention
 - Notifications
- The availability of a system is the probability that it will be operational when it is needed

$$\alpha = \frac{\text{mean time to failure}}{\text{mean time to failure} + \text{mean time to repair}}$$

Availability General Scenario



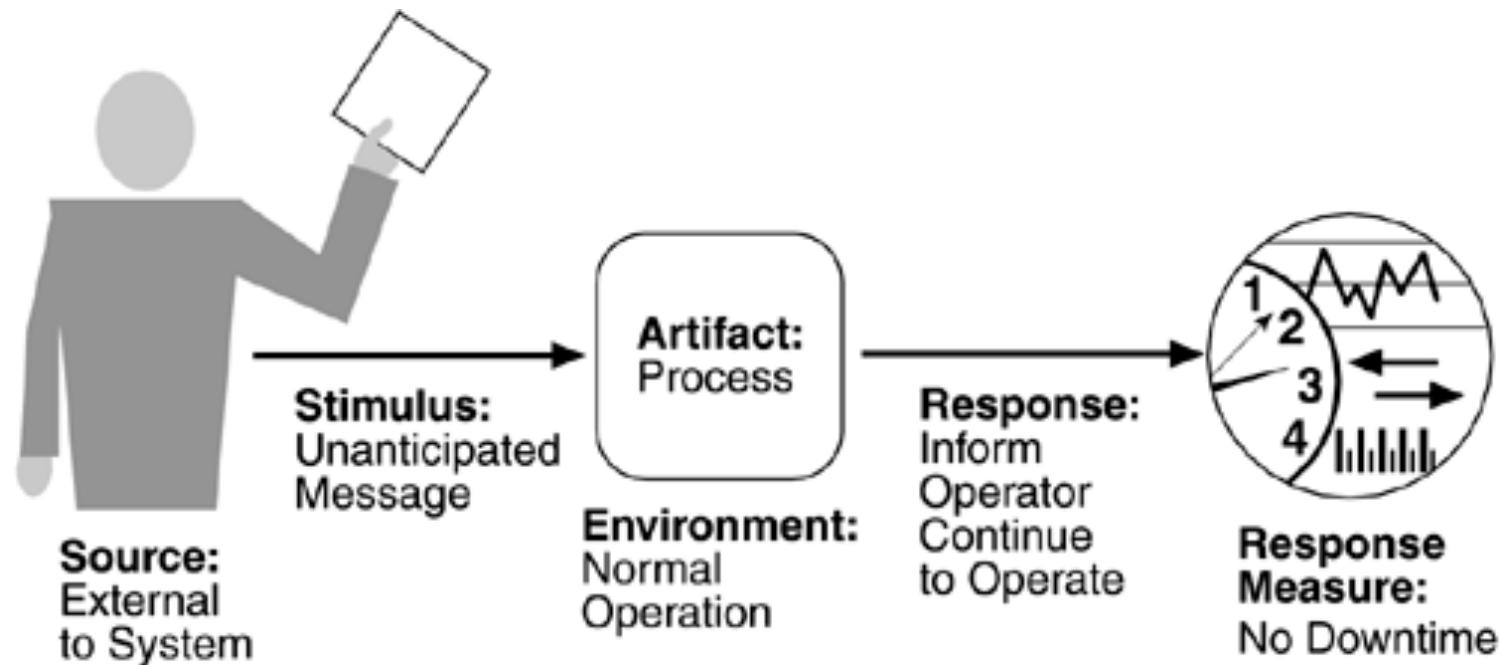
Source	Stimulus	Environment	Response	Measure
Internal to system	Crash	Normal operation	Prevent the failure	Time interval available
	Omission	Startup	Log the failure	Availability %
External to system	Timing	Shutdown	Notify users / operators	Detection time
	No response	Repair mode	Disable source of failure	Repair time
	Incorrect response	Degraded (failsafe) mode	Temporarily unavailable	Degraded mode time interval
		Overloaded operation	Continue (normal / degraded)	Unavailability time interval

Availability General Scenario

Portion of Scenario	Possible Values
Source	Internal to the system; external to the system
Stimulus	Fault: omission, crash, timing, response
Artifact	System's processors, communication channels, persistent storage, processes
Environment	Normal operation; degraded mode (i.e., fewer features, a fall back solution)
Response	System should detect event and do one or more of the following: record it notify appropriate parties, including the user and other systems disable sources of events that cause fault or failure according to defined rules be unavailable for a prespecified interval, where interval depends on criticality of system continue to operate in normal or degraded mode
Response Measure	Time interval when the system must be available Availability time Time interval in which system can be in degraded mode Repair time

Availability Concrete Scenario

E.g. An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and continues to operate with no downtime.



Exercise: Availability Concrete Scenario

Q: Write a Concrete Quality Attribute Scenario for Word Application.

A: Kill Signal is received from the Windows OS to Ms. Word Application during process Not Responding state and the application saves unsaved work in a temp file and process terminates without any data loss.

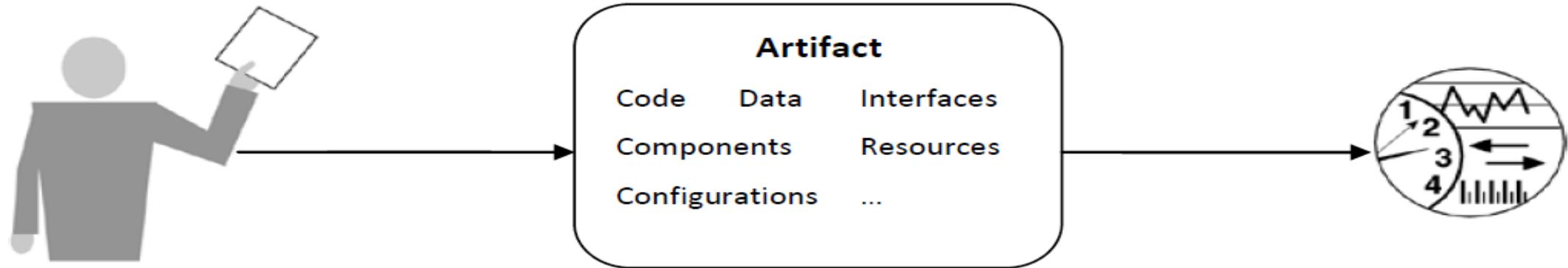
Identify:

- Source
- Stimulus
- Artifact
- Environment
- Response
- Response Measure

Modifiability (QAS)

- Modifiability is about the cost of change.
- It brings up two concerns:
 - What can change (the artifact)?
 - Functions, Platform, Environment, Protocol, Qualities, Capacity
 - When is the change made and who makes it (the environment)?
 - A developer, an end user, or a system administrator.
 - At Implementation, Compilation, Build, Configuration, Execution

Modifiability General Scenario



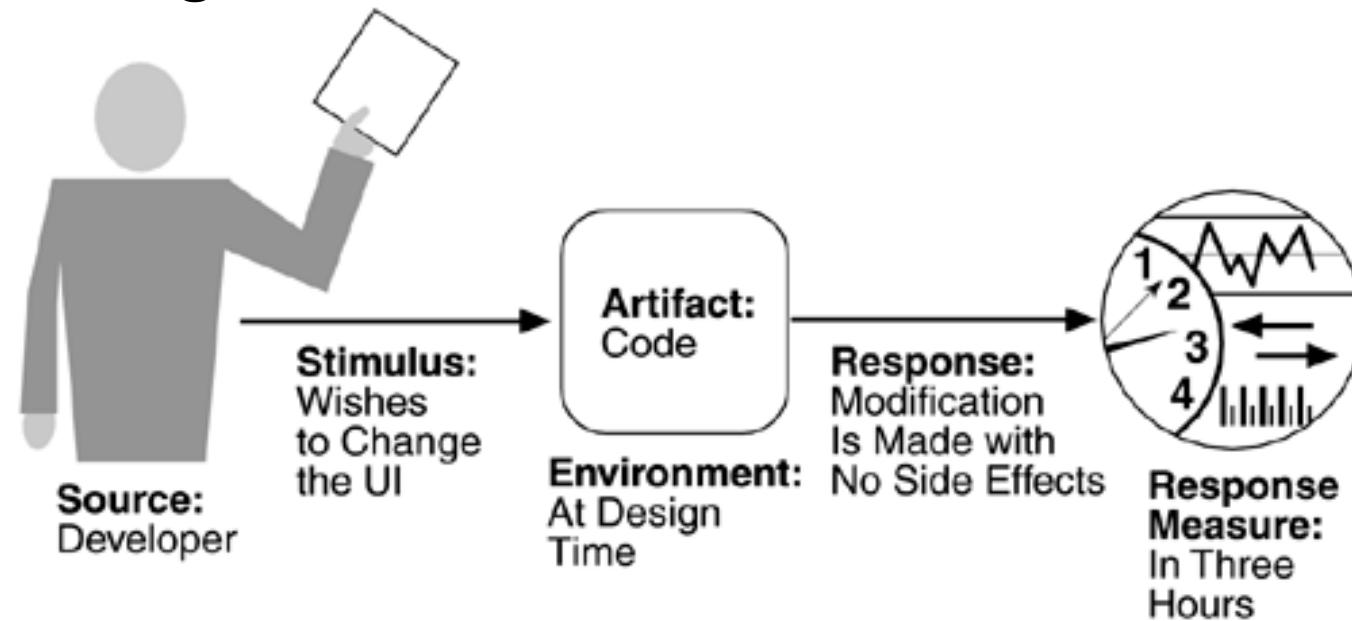
Source	Stimulus	Environment	Response	Measure
End-user	Add / delete / modify functionality,	Runtime	Make modification	Cost in effort
Developer	quality attribute, capacity or technology	Compile time	Test modification	Cost in money
System-administrator		Build time	Deploy modification	Cost in time
		Initiation time		Cost in number, size, complexity of affected artifacts
		Design time		Extent affects other system functions or qualities
				New defects introduced

Modifiability General Scenario

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	Wishes to add/delete/modify/vary functionality, quality attribute, capacity
Artifact	System user interface, platform, environment; system that interoperates with target system
Environment	At runtime, compile time, build time, design time
Response	Locates places in architecture to be modified; makes modification without affecting other functionality; tests modification; deploys modification
Response Measure	Cost in terms of number of elements affected, effort, money; extent to which this affects other functions or quality attributes

Modifiability Concrete Scenario

E.g. A developer wishes to change the user interface to make a screen's background color blue. This change will be made to the code at design time. It will take less than three hours to make and test the change and no side effect changes will occur in the behavior.



Exercise: Modifiability Concrete Scenario

Q: Write a Concrete Quality Attribute Scenario for updating database Password on a 3 Tier application.

A: System Administrator wishes to change the password of the database configuration file on the Data Tier at the System Maintenance Time; the activity takes 2 minutes and the application is able to connect to the Database without any issues.

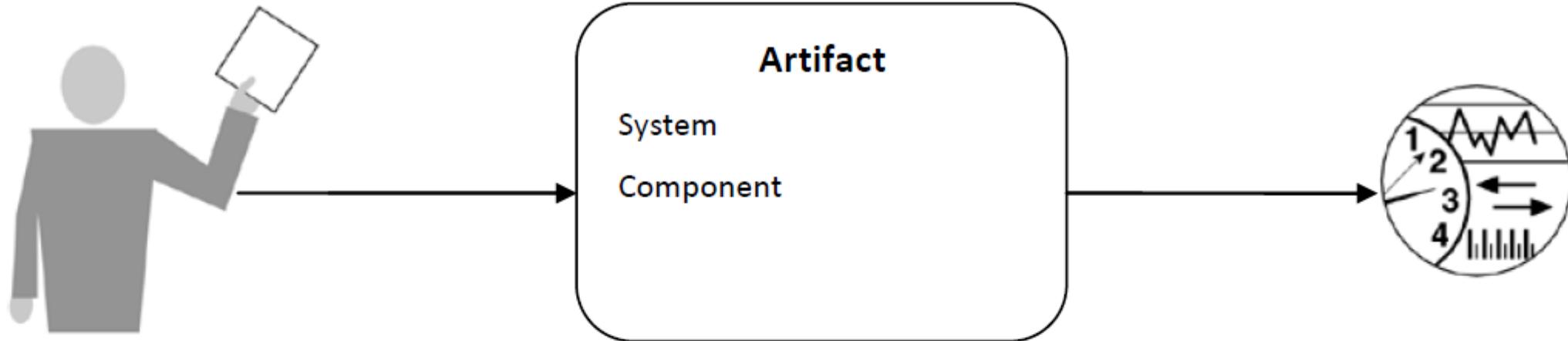
Identify:

- Source
- Stimulus
- Artifact
- Environment
- Response
- Response Measure

Performance (QAS)

- Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them.
- There are a variety of characterizations of event arrival and the response but basically performance is concerned with how long it takes the system to respond when an event occurs

Performance General Scenario



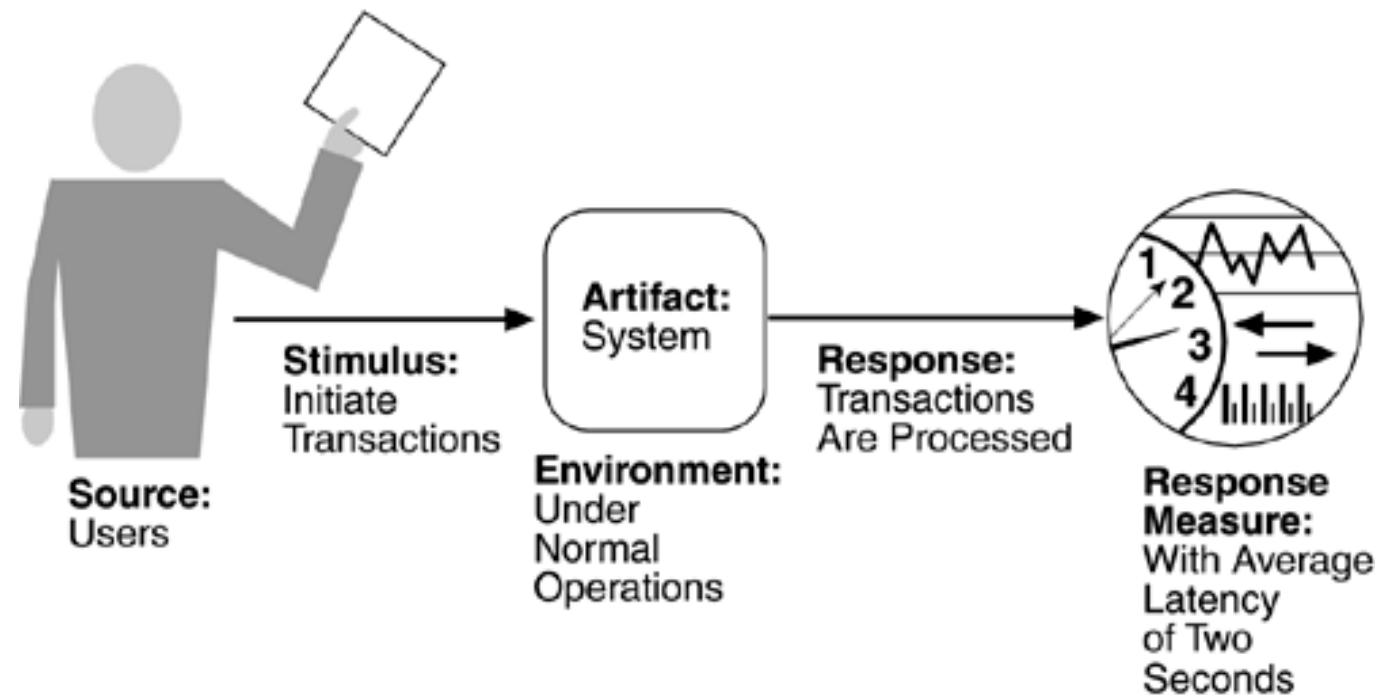
Source	Stimulus	Environment	Response	Measure
Internal to the system	Periodic events	Normal mode	Process events	Latency
	Sporadic events	Overload mode	Change level of service	Deadline
External to the system	Bursty events	Reduced capacity mode		Throughput
	Stochastic events	Emergency mode		Jitter
		Peak mode		Miss rate
				Data loss

Performance General Scenario

Portion of Scenario	Possible Values
Source	One of a number of independent sources, possibly from within system
Stimulus	Periodic events arrive; sporadic events arrive; stochastic events arrive
Artifact	System
Environment	Normal mode; overload mode
Response	Processes stimuli; changes level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss

Performance Concrete Scenario

E.g. Users initiate 1,000 transactions per minute randomly under normal operations, and these transactions are processed with an average latency of two seconds.



Exercise: Performance Concrete Scenario

Q: Write a Concrete Quality Attribute Scenario for A banking application for a Weekly Transaction Report.

A: The Finance Analyst schedules weekly report of the Banking Application during Normal Operational Time (8am-5pm), the process starts to execute at the Off-Peak Time (between 10pm-2am) and the Report Excel File generates successfully within 30 minutes.

Identify:

- Source
- Stimulus
- Artifact
- Environment
- Response
- Response Measure

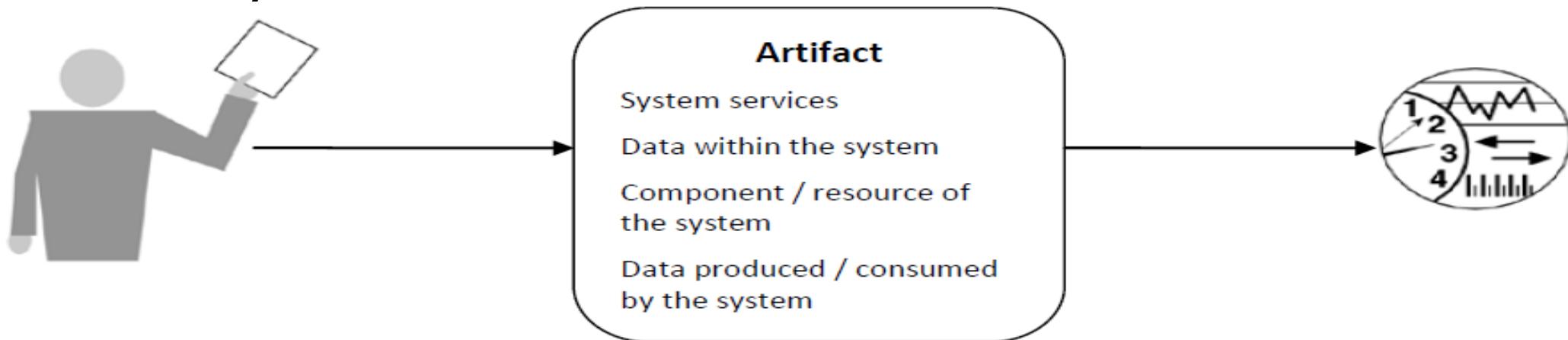
Max Latency = 18 hours (8am → 2am)

Max Deadline = 30 mins

Security (QAS)

- Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users.
- An attempt to breach security is called an attack and have many forms;
 - Unauthorized attempt to access data
 - Modify data
 - Intended to deny services to legitimate users.
- Characterization
 - Nonrepudiation : Transaction cannot be denied by any of the parties
 - Confidentiality : Data or services are protected from unauthorized access.
 - Integrity : Data or services are being delivered as intended.
 - Assurance or authenticity : The parties to a transaction are who they purport to be
 - Availability (no denial of service) : The system will be available for legitimate use
 - Auditing : The system tracks activities

Security General Scenario



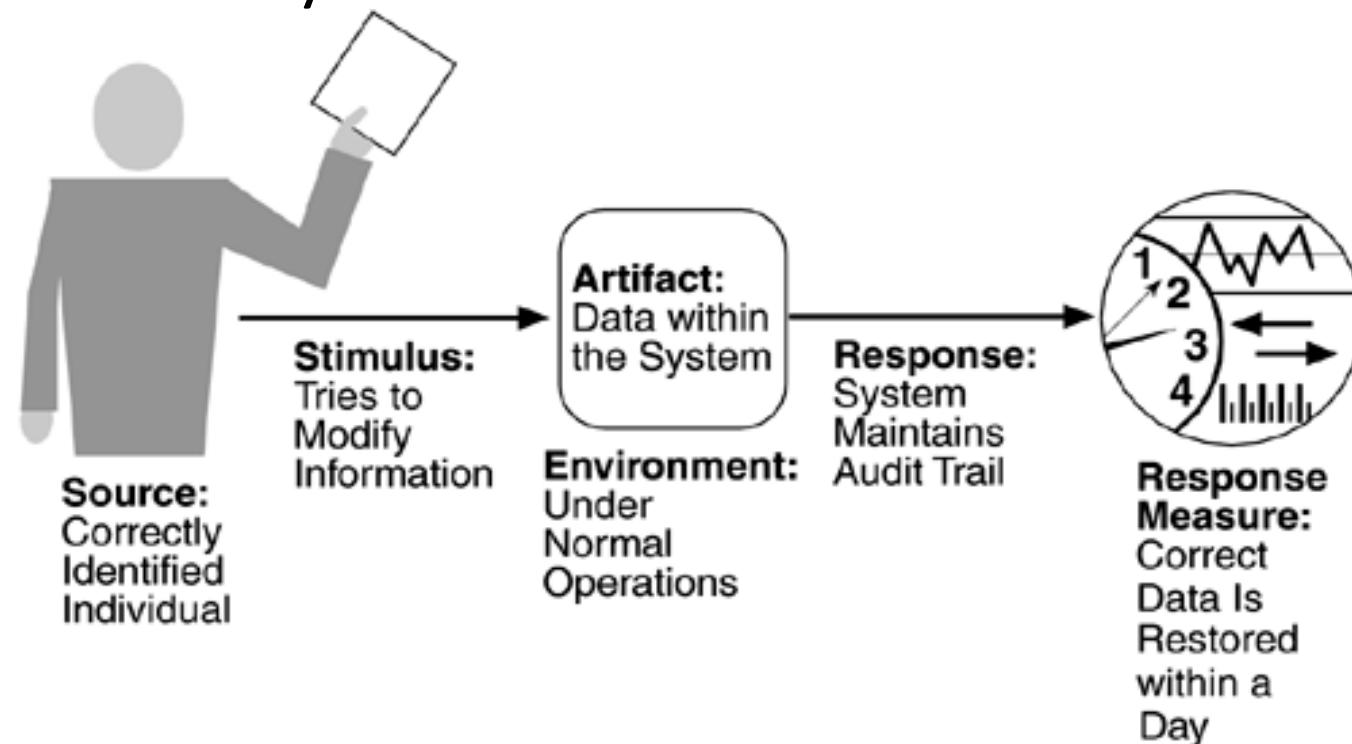
Source	Stimulus	Environment	Response	Measure
Identified user	Attempt to display data	Online or Offline	User Authentication Identification	Time/Effort/Resources
Unknown user		Connected or Disconnected	Allow/Blocks Access	Probability of Success
Hacker from outside the organization	Attempt to modify data	Firewalled or Open	Grant/Withdraw Permission	Probability of Detection
Hacker from inside the organization	Attempt to delete data		Data Readability	Percentage of Accessibility
	Access system services			
	Change system's behavior			
	Reduce availability			

Security General Scenario

Portion of Scenario	Possible Values
Source	Individual or system that is correctly identified, identified incorrectly, of unknown identity who is internal/external, authorized/not authorized with access to limited resources, vast resources
Stimulus	Tries to display data, change/delete data, access system services, reduce availability to system services
Artifact	System services; data within system
Environment	Either online or offline, connected or disconnected, firewalled or open
Response	Authenticates user; hides identity of the user; blocks access to data and/or services; allows access to data and/or services; grants or withdraws permission to access data and/or services; records access/modifications or attempts to access/modify data/services by identity; stores data in an unreadable format; recognizes an unexplainable high demand for services, and informs a user or another system, and restricts availability of services
Response Measure	Time/effort/resources required to circumvent security measures with probability of success; probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied

Security Concrete Scenario

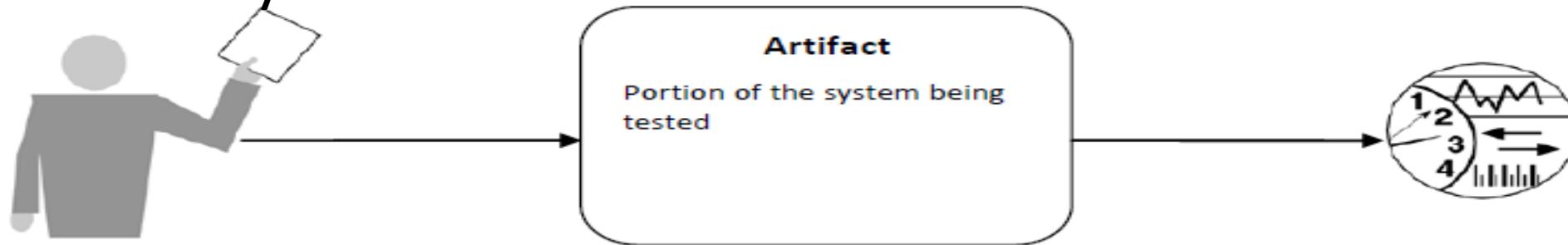
E.g. A correctly identified individual tries to modify system data from an external site; system maintains an audit trail and the correct data is restored within one day.



Testability (QAS)

- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.
 - At least 40% of the cost of developing well-engineered systems is taken up by testing.
- For a system to be properly testable, it must be possible to control each component's internal state and inputs and then to observe its outputs

Testability General Scenario



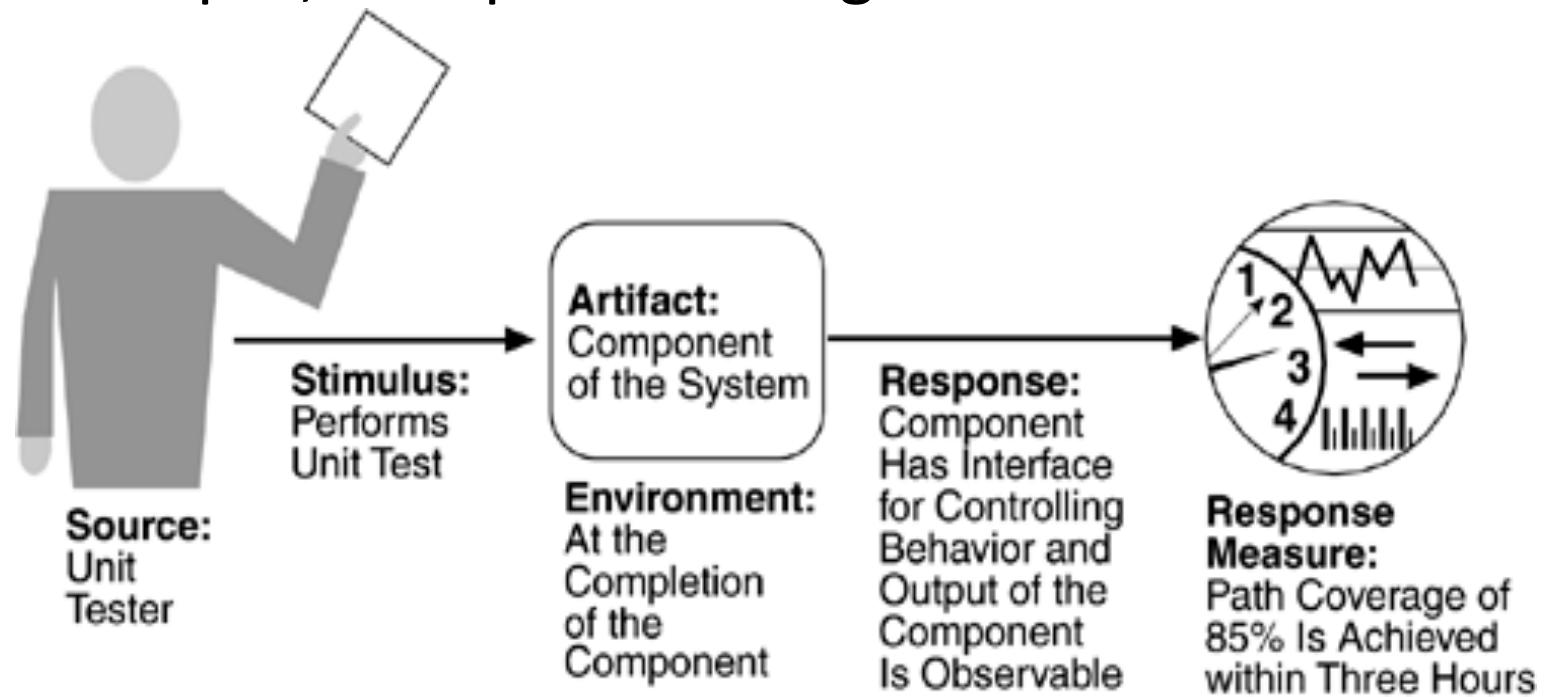
Source	Stimulus	Environment	Response	Measure
Unit tester	Execution of tests due to completion of code increment	Design time	Execute test suite & capture results	Effort to find fault
Integration tester		Development time	Capture cause of fault	Effort to achieve coverage %
System tester		Compile time	Control & monitor state of the system	Probability of fault being revealed by next test
Acceptance tester		Integration time		
End user		Deployment time		
Automated testing tools		Run time		

Testability General Scenario

Portion of Scenario	Possible Values
Source	Unit developer; Increment integrator; System verifier; Client acceptance tester; System user
Stimulus	Analysis, architecture, design, class, subsystem integration completed; system delivered
Artifact	Piece of design, piece of code, complete application
Environment	At design time, at development time, at compile time, at deployment time
Response	Provides access to state values; provides computed values; prepares test environment
Response Measure	Percentage of the statements executed; Probability of failure if fault exists; Time to perform tests; Length of longest dependency chain in a test Length of time to prepare test environment

Testability Concrete Scenario

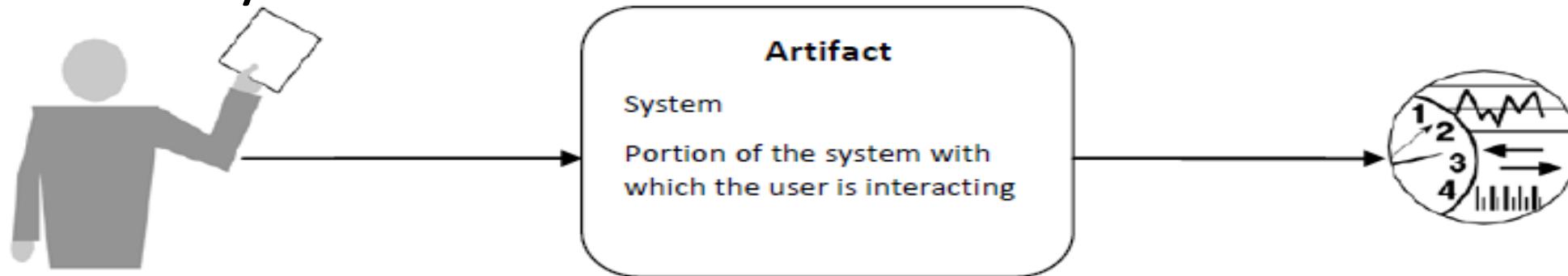
E.g. A unit tester performs a unit test on a completed system component that provides an interface for controlling its behavior and observing its output; 85% path coverage is achieved within three hours.



Usability (QAS)

- How easy it is to learn the features of the system
- How efficiently the user can use the system
- How well the system handles user errors
- How well the system adapts to user needs
- To what degree the system gives the user confidence in the correctness of its actions.

Usability General Scenario



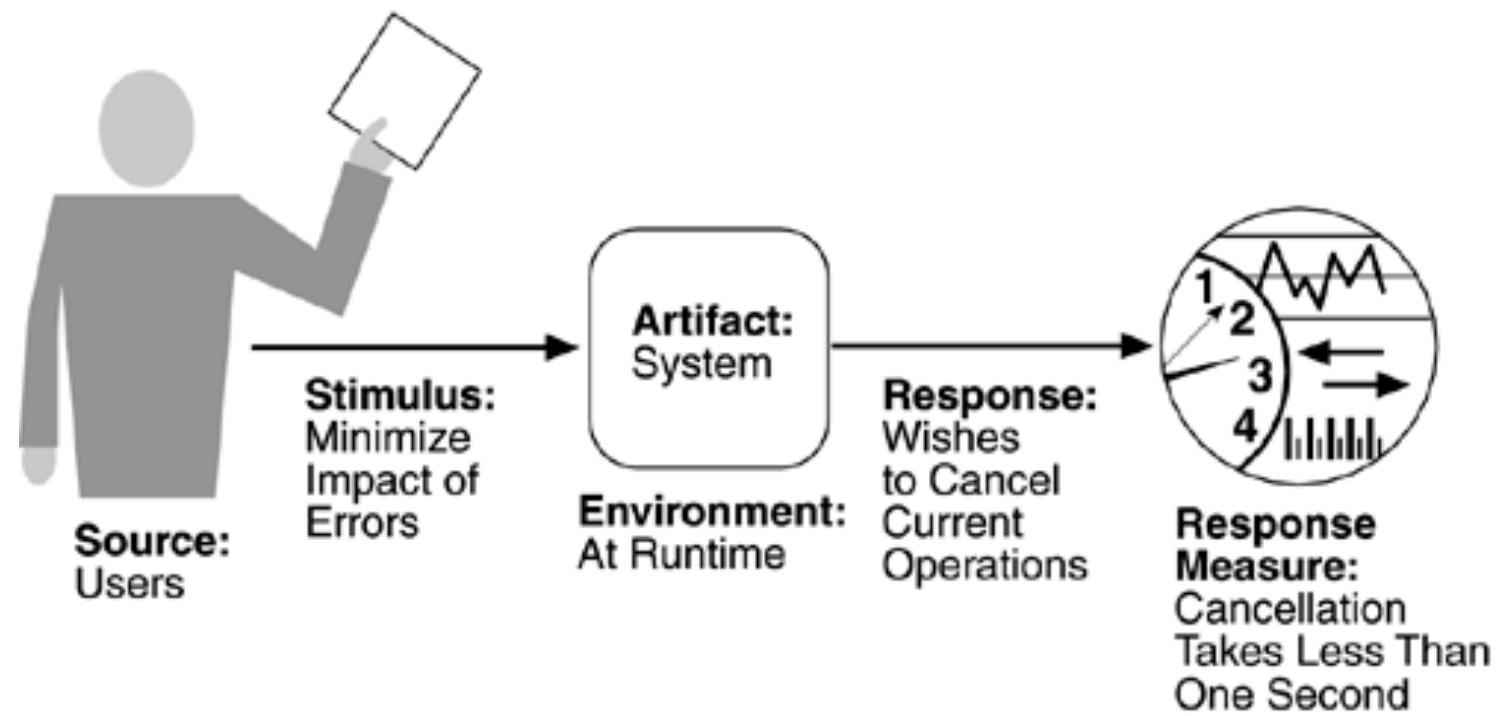
Source	Stimulus	Environment	Response	Measure
End user (possibly special role)	Use the system efficiently	Runtime	Provide features needed	Task time
	Learn to use the system	Configuration time	Anticipate the user's needs	Number of errors
	Minimize impact of errors			Number of tasks accomplished
	Adapt the system			User satisfaction
	Configure the system			Gain of user knowledge
				Ratio of successful operations to total operations
				Amount of time / data lost when error occurs

Usability General Scenario

Portion of Scenario	Possible Values
Source	End user is always the source (can be broken down to user roles/actors)
Stimulus	Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable
Artifact	System (or a part of the system the user is interacting)
Environment	At runtime or configure time
Response	System provides one or more of: <ul style="list-style-type: none">• To support learn system features• To support use system efficiently• To minimize impact of errors• To adapt system: customizability; internationalization• To feel comfortable: display system state; work at the user's pace
Response Measure	Task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, amount of time/data lost

Usability Concrete Scenario

E.g. A user, wanting to minimize the impact of an error, wishes to cancel a system operation at runtime; cancellation takes place in less than one second.



Exercise (offline)

- Refer a few existing software systems (e.g. your Group Case Study) and identify 2-3 quality attribute scenarios for the Main Quality Attribute(s) Under consideration
- Check with the actual Software System documentation / agreements / guides if the above identified Quality Attribute response measures are stated
- Propose new Quality Attribute Scenarios for your system under considerations

Tactics

- An architectural tactic is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural decisions

NEXT LECTURE:

Ways to improve Quality Attributes

Tactics Framework

References

- <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch04lev1sec4.html>
- <http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Two+Creating+an+Architecture/Chapter+4.+Understanding+Quality+Attributes/4.4+Quality+Attribute+Scenarios+in+Practice/>

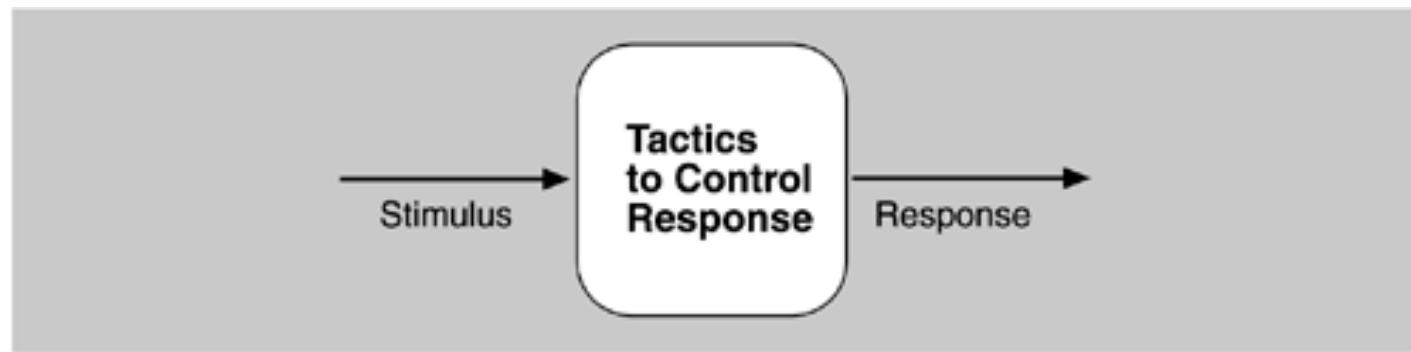


Quality Attribute Tactics

**Software Architecture
3rd Year – Semester 1
Lecture 13**

What are Tactics?

- An architectural tactic is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural decisions



- A planned way to achieve quality goals

More on Tactics...

- A tactic is a design decision that influences the control of a quality attribute response
- A collection of tactics an Architectural Strategy
- Architects usually choose architectural patterns to realize some tactic. But patterns inevitably implement multiple tactics. This can make architectural analysis more difficult.

Achieving Quality Goals

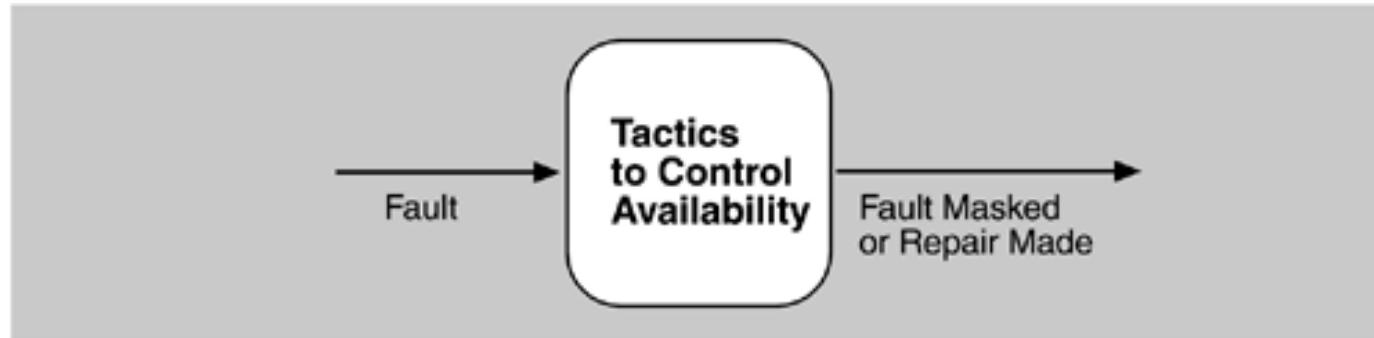
- For each quality, we need to find or formulate tactics to achieve that quality
- Each tactic is a design option for the architect
 - E.g. Increasing Availability: One of the tactics is introducing redundancy. This is one option the architect has to increase availability, but not the only one. Usually achieving high availability through redundancy implies a need for synchronization

Applying Tactics

- Tactics can refine other tactics:
 - We identified redundancy as a tactic. As such, it can be refined into redundancy of data (in a database system) or redundancy of computation (in an embedded control system). Both types are also tactics. There are further refinements that a designer can employ to make each type of redundancy more concrete. For each quality attribute that we discuss, we organize the tactics as a hierarchy.
- Patterns package tactics:
 - There can be multiple tactics to achieve a single goal . For example availability.
 - Availability tactics package include Redundancy tactic and a Synchronization tactic

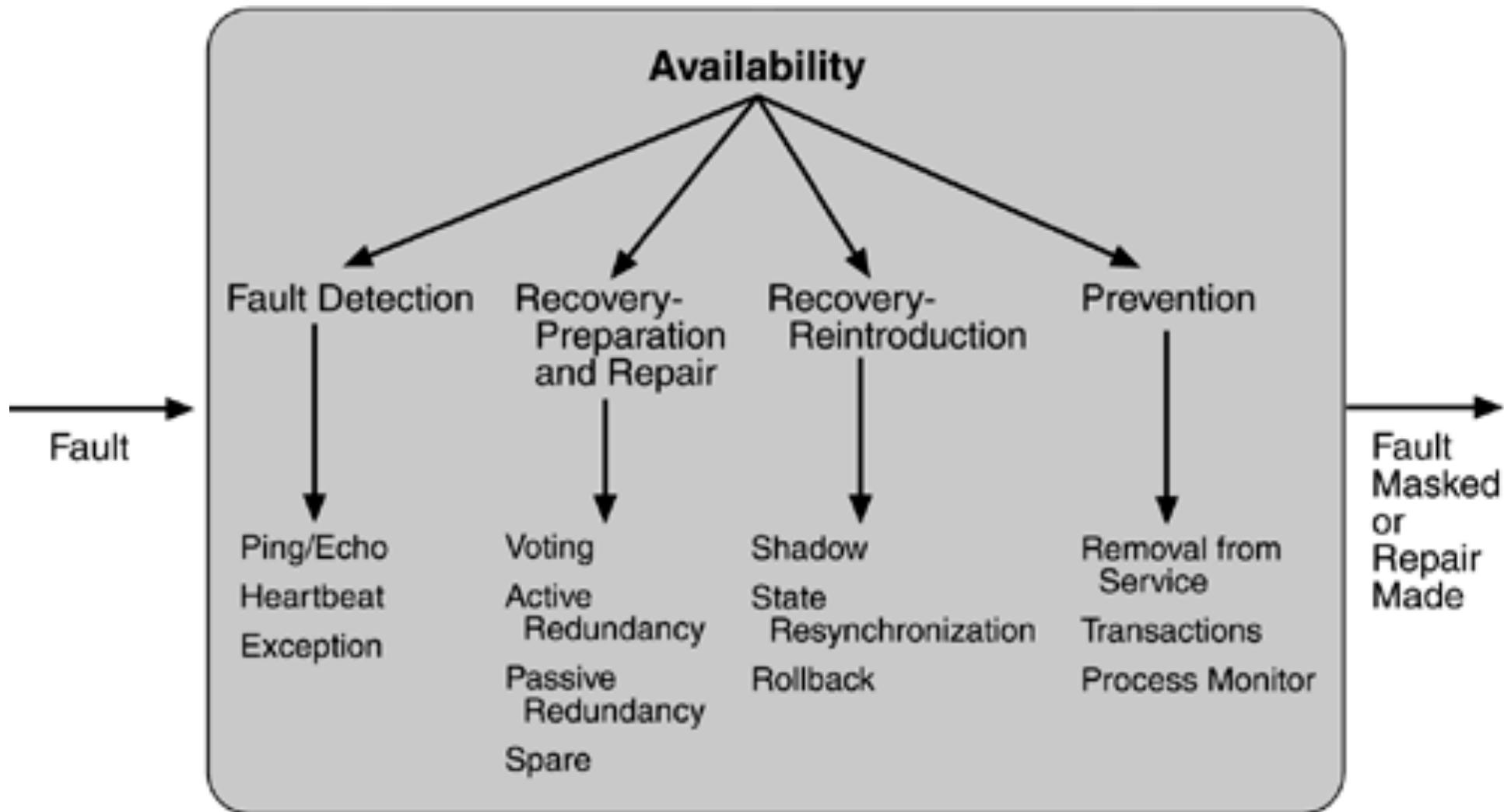
Availability Tactics

- A fault (or combination of faults) has the potential to cause a failure and would affect availability.



- Availability Tactics will keep faults from becoming failures or at least bound the effects of the fault and make repair possible.
 - Fault Detection
 - Fault Recovery
 - Fault Prevention

Availability Tactics Framework



Availability Tactics in Detail

- Fault Detection
 - Ping/echo: send the component a ping and wait for an echo. If not received, notify the fault correction system.
 - Are you alive?
 - Heartbeat: have the component emit a heartbeat periodically, and have another component listen. If not received, notify the fault correction system.
 - I am alive...
 - Exceptions: trigger an exception handler when a fault occurs (most suitable for within the process/component)
 - I'm dead

Availability Tactics in Detail

- Fault Recovery (Preparation & Repair)
 - Voting: have redundant processors and have the processes vote for the answer. If one is different, fail it. No downtime.
 - Majority rules or Preferred Component priority
 - Active redundancy: all redundant components respond to all events, output is taken from the first to respond
 - Passive redundancy: only the master responds to events, but the backup's state is updated so it can take over whenever necessary. Downtime is seconds.
 - Spare: Keep a standby spare component to replace many different failed components. Downtime is minutes.
 - Here you make checkpoints of the system state periodically and log all state changes

Availability Tactics in Detail

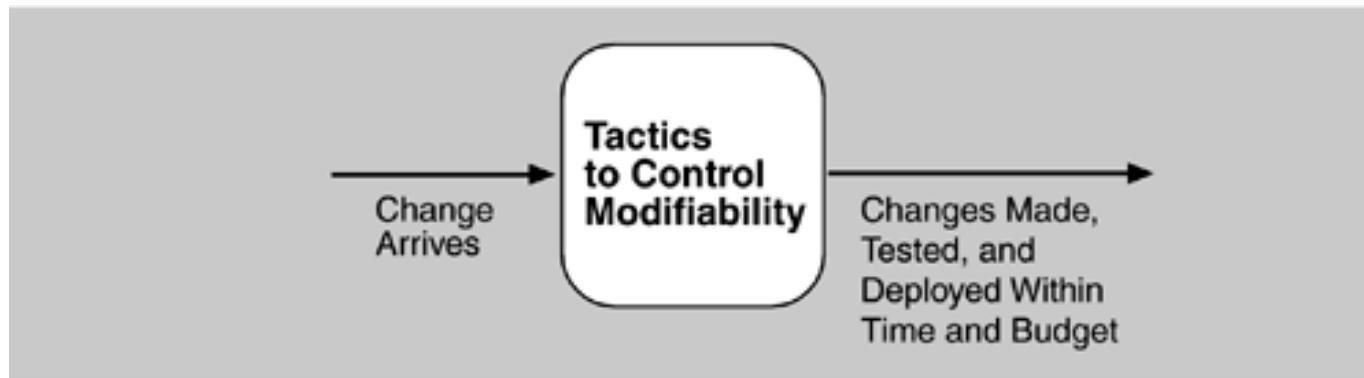
- Fault Recovery (Reintroduction)
 - Shadow operation: have a previously failed component mimic the backup for a while to verify correct operation.
 - State resynchronization: when a failed component is returned to service, its state needs to be resynchronized with the backup.
 - Checkpoint/rollback: record consistent states, and when a fault occurs, restore to the checkpoint.

Availability Tactics in Detail

- Fault Prevention
 - Removal from surface: take a component down periodically to prevent anticipated failures, e.g. reboots to prevent memory leaks from leading to failure.
 - Transactions: bundle sequential steps into chunks that can be undone all at once in case of a fault on an intermediate step.
 - Process monitor: monitor for faulting processes, and kill and restart when a fault is detected.

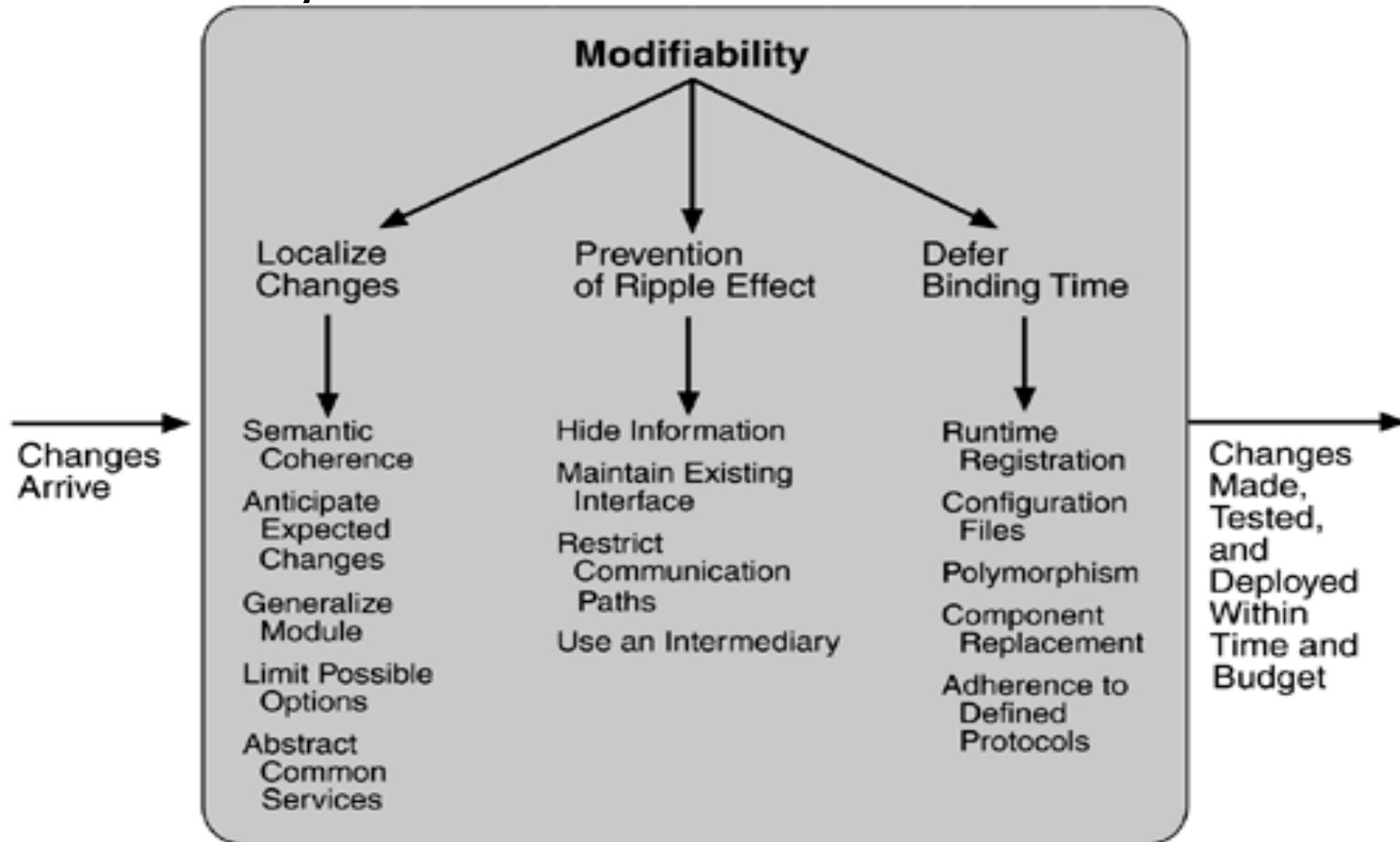
Modifiability Tactics

- Goal is to control the time and cost to implement, test, and deploy changes.



- Tactics for modifiability is organized according to their goals
 - Localize modifications: Reducing the number of modules affected
 - Prevent ripple effects: Limiting the modification to localized modules
 - Defer binding time: Controlling deployment time and cost

Modifiability Tactics Framework



Modifiability Tactics in Detail

- Goal: Reduce the number of modules affected by a change
- Localize Modifications
 - Maintain semantic coherence — keep things that are related together
 - Anticipate expected changes — keep things that are likely to change in one place
 - Generalize the module — making a module more general allows it to compute a broader range of functions based on input
 - Limit possible options — don't allow much change

Modifiability Tactics in Detail

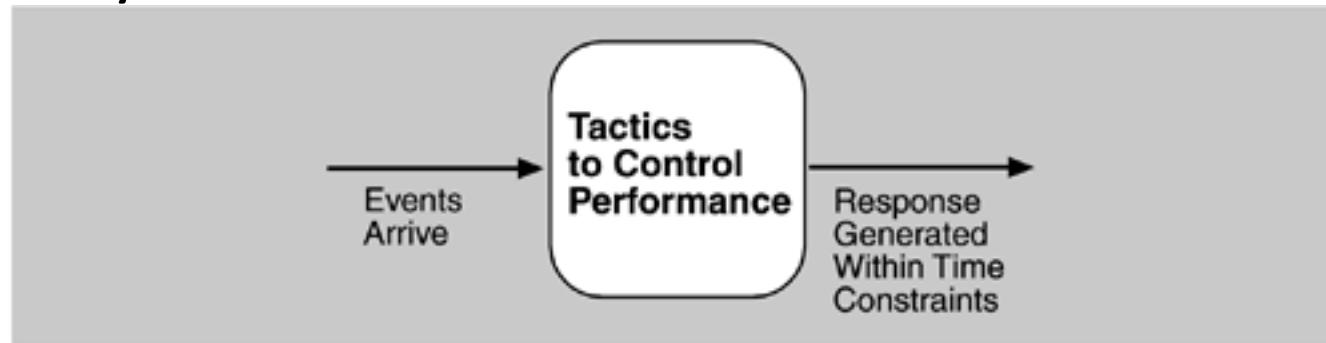
- Goal: Limited modifications of the module due to changes on the dependent modules
- Prevent Ripple Effect
 - Hide information — less visible information means fewer things to be dependent on
 - Maintain existing interfaces — don't change interfaces, use patterns such as an adapter
 - Restrict communication paths — don't allow data to flow through too many modules (reduce consumptions and production)
 - Use an intermediary — break dependency chain (not for semantic changes), use patterns such as Façade, Mediator, Delegate, Proxy

Modifiability Tactics in Detail

- Goal: Control deployment time and cost, allow deployment or operational time changes
- Differ Binding Time
 - Runtime registration — have participants identify themselves after the system has begun operation (plug & play)
 - Configuration files — initiation time
 - Polymorphism — what method names mean determined at runtime
 - Component replacement — runtime elements can be changed at load time
 - Adherence to defined protocols — runtime binding of independent processes

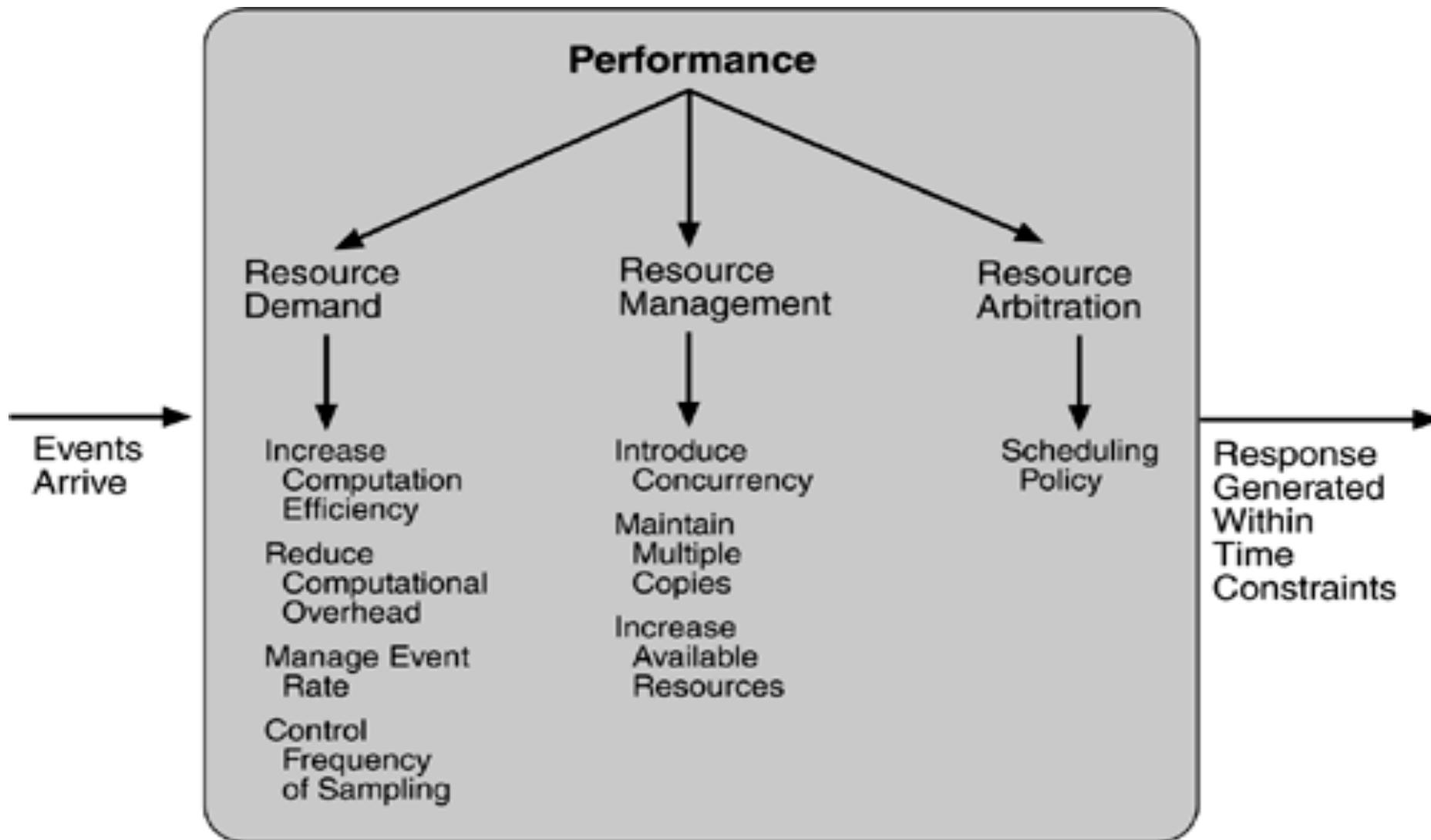
Performance Tactics

- The goal of performance tactics is to generate a response to an event arriving at the system within some time constraint.



- Main contributing factors are...
 - Resource Consumption – e.g. CPU, Disk I/O, etc...
 - Blocked Time – Availability and Dependencies of the Resources, Multi Process Priorities or Policies

Performance Tactics Framework



Performance Tactics in Detail

- Resource Demand
 - Increase computational efficiency – More efficient Algorithms. E.g. Bubble sort Vs. Quick Sort
 - Reduce computational overhead – e.g. without calculating a value in multiple places calculate once and refer to the value (final static double pi = 22/7)
 - Control Frequency of Sampling – Reduce the sampling frequencies (e.g. listeners, watchers)

Performance Tactics in Detail

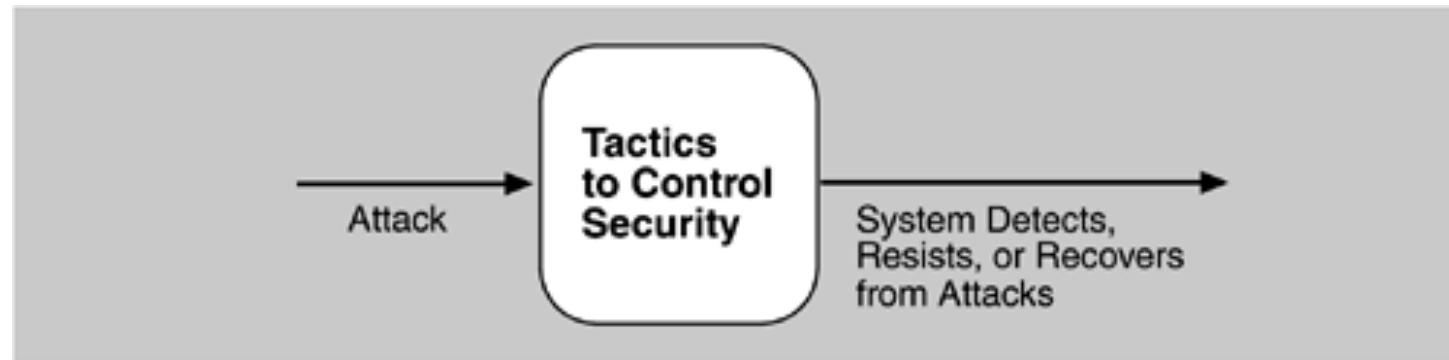
- Resource Management
 - Introduce concurrency - If requests can be processed in parallel , the blocked time can be reduced
 - Maintain multiple copies of either data or computations – cache should be consistent and synchronized.
 - Increase available resources – faster or additional processors, memory or networks

Performance Tactics in Detail

- Resource Arbitration
 - Scheduling Policy – When there is a contention for a resource (Disk IO, Network, etc...) processes should be scheduled
 - Optimal Resource Usage
 - Maximize Throughput
 - Ensure fairness
 - Prevent starvation
 - Scheduling Policies – FIFO, Fixed, Round Robin, etc...

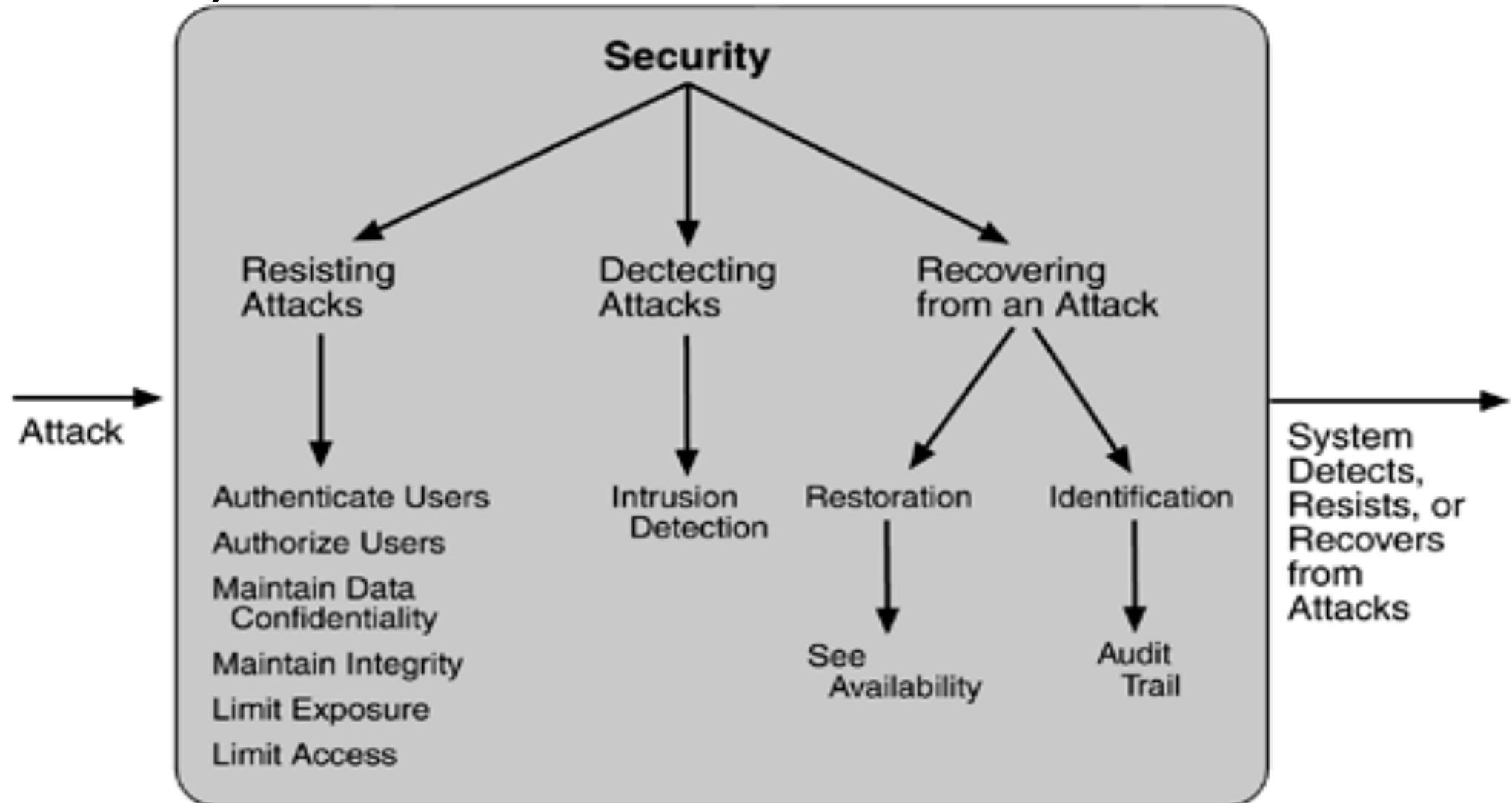
Security Tactics

- Goal is to ensure Non-repudiation, Confidentiality, Integrity and Assurance.



- Main concerns are...
 - Resisting Attacks
 - Detecting Attacks
 - Recovery from Attacks

Security Tactics Framework



Security Tactics in Detail

- Resisting Attacks
 - Authenticate users – Ensuring that a user is actually who it's purport to be
 - Authorize users – Ensuring that an authenticated user has the right to access
 - Maintain data confidentiality – Data should be protected from unauthorized access — encrypt data and communication links (VPN, SSL)
 - Limit exposure – avoid single point of failure, limited services per host
 - Limit access — firewall, DMZ

Security Tactics in Detail

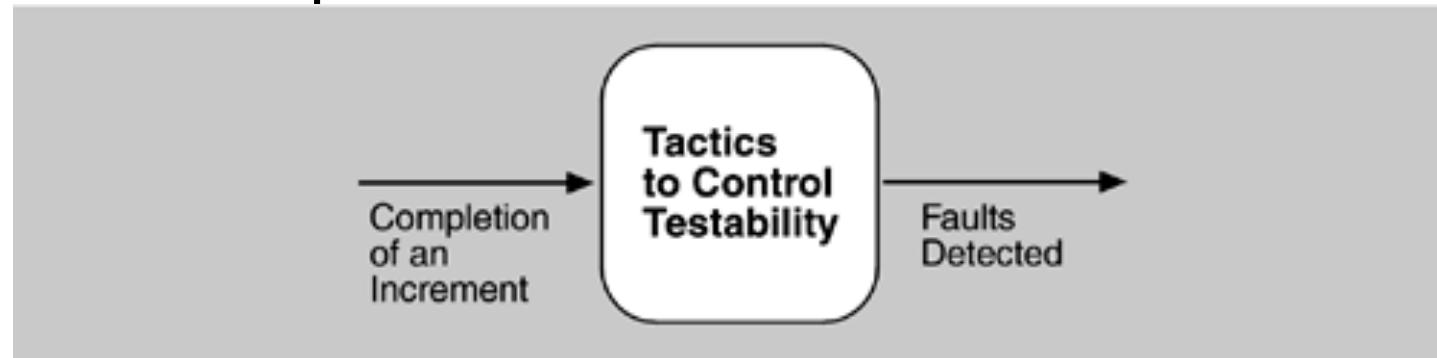
- Detecting Attacks
 - Intrusion Detection – Historical Statistics can be compared with current
 - Server Traffic
 - Resource Usage
 - Monitoring Networks

Security Tactics in Detail

- Recovery from Attacks
 - Restoration
 - Availability tactics can be used with special attention
 - Maintaining redundant copies of system & data
 - Identification (To identify an attacker)
 - Maintain an audit trail (can it be attacked?)
 - Can be used to trace the actions of an attacker
 - So it becomes an attacked target, need to be maintained in a trusted environment

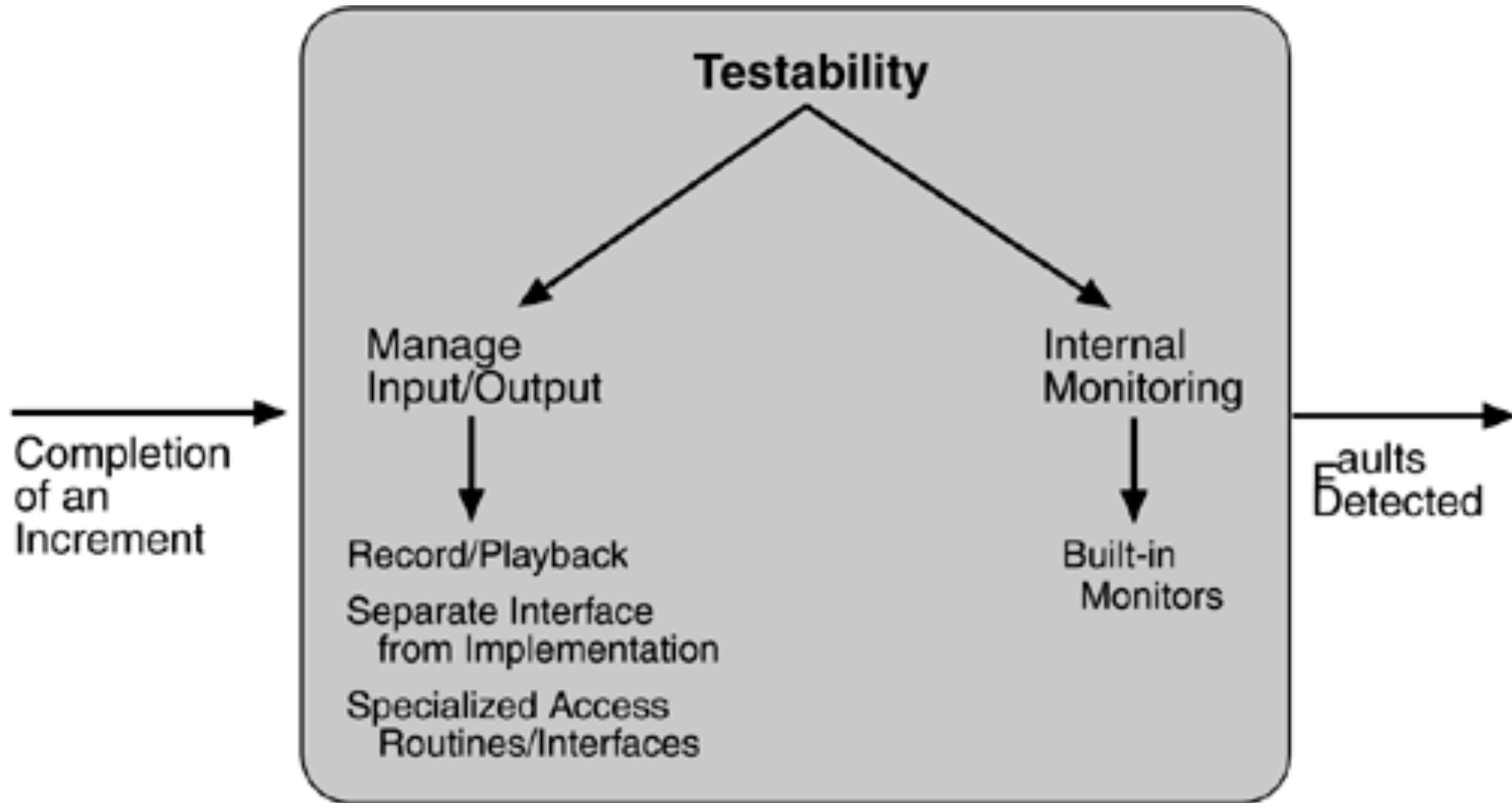
Testability Tactics

- Goal is to allow easier testing when an increment of software development is completed



- The main focus is runtime testing. This requires that input be provided to the software and output be captured
 - Input / Output
 - Internal Monitoring

Testability Tactics Framework



Testability Tactics in Detail

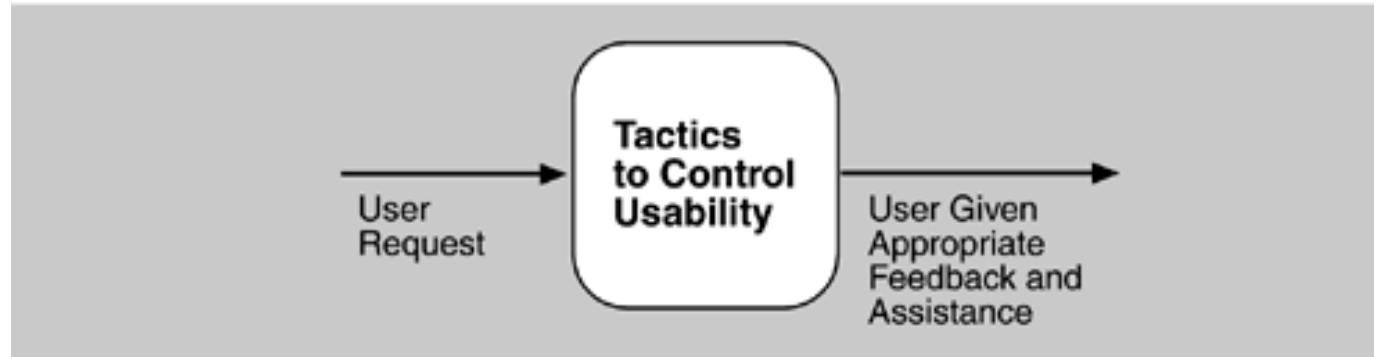
- Input / Output
 - Record/playback – capturing information crossing an interface using it as input into the test harness
 - Separate interface from implementation - Stubbing implementations allows the remainder of the system to be tested in absence of the component being stubbed (mock testing)
 - Specialize access routes/interfaces – hierarchy of test interfaces (have to be cautious on exposing unwanted interfaces)

Testability Tactics in Detail

- Internal Monitoring
 - Built-in monitors
 - This interface can be a permanent interface of the component or it can be introduced temporarily.
 - Would help to test other Runtime Quality Attributes such as Performance, Availability, Security, etc...

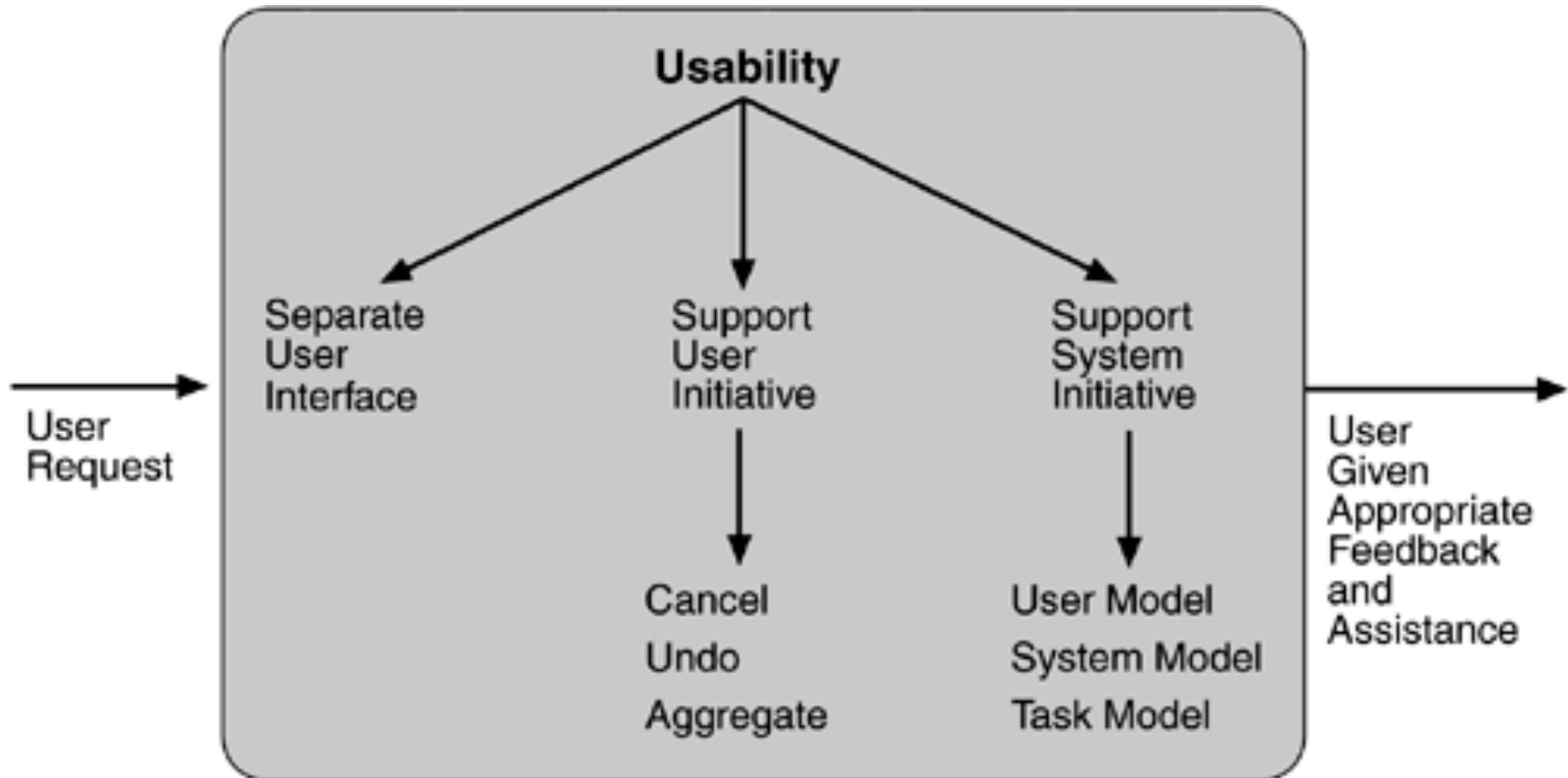
Usability Tactics

- Goal is to provide an easy way to the user to accomplish a desired task and the kind of support the system provides to the user



- The main areas are...
 - Runtime Tactics
 - Design Time Tactics

Usability Tactics Framework



Usability Tactics in Detail

- Runtime Tactics
 - User Initiatives – Operations to help the user such as Pause Operation, Cancel, Undo, etc...
 - System Initiatives – Provide feedback, suggestions, notifications, conformations, etc...
 - Mixes Initiatives – When user hits “Upload” the system gives a Progress Bar

Usability Tactics in Detail

- Design Time Tactics
 - Separate the user interface from the rest of the application
 - Separate User Interface Components – MVC
 - Maintain Semantic Coherence – Localizing expected changes.

References

- <http://proquestcombo.safaribooksonline.com/0321154959>
- <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05.html>

Exercise (offline)

- Refer a few existing software systems (e.g. your Group Case Study) and identify what are the important quality attributes for that system
 - Identify about 4-5 Quality Attributes
- Suggest suitable tactics for implementing the identified Quality Attributes
 - Provide justification for your answers
- Cross check how the system actually improve the identified quality attribute with what you have suggested
 - If the system does not follow the tactic you suggested research on why? Hint Trade-Offs



Software Architecture Evaluation

**Software Architecture
3rd Year – Semester 1
Lecture 14**

Why Software Architecture Evaluation?

- How can you be sure whether the architecture chosen for your software is the right one?
- How can you be sure that it won't lead to disaster?
- Architecture evaluation aims to improve and find deficiencies in an architecture as early as possible in the development process.
 - Modifying the architecture during the architectural design phase is cheap, modifying the architecture later is costly!
- Architecture determines the structure of the project: schedules and budgets, performance goals, team structure, documentation organization, and testing and maintenance activities.

How

- Evaluation can be based on questioning or measuring techniques.
- Qualitative Evaluation and Quantitative Evaluation.
- Method Types
 - Scenario-based
 - Mathematical model-based
 - Simulation based
 - Experience-based reasoning

When

- The classical application of architecture evaluation occurs when the architecture has been specified but before implementation has begun.
- Early
 - At any stage in the architecture creation process to examine those architectural decisions already made and choose among architectural options.
 - Based on specification and description of the software architecture, and other sources of information, such as interviews with architects.
- Late
 - Takes place when the architecture is nailed down and the implementation is complete. Mainly used when architecture is inherited from legacy system.
 - Based on metrics. E.g. cohesion and coupling of architectural components

Challenges

- Requires an expert evaluation team to deal with unpredictable risks along with the known risks
- Lack of common understanding of the high level design
- Different stakeholders have different interests
- The quality requirements for a system may not be written properly or may not be finished when the architecture is designed

Who is involved?

- Evaluation Team
 - People who will conduct the evaluation and perform the analysis.
- Stakeholders
 - People who have a vested interest in the architecture and the system.

Planning & Process

- Before you start an evaluation, you need
 - Clearly articulated goals and requirements for the architecture
 - Select an evaluation method
 - Controlled scope (a small number of explicit goals)
 - Cost-effectiveness (it might not be necessary to do full-scale reviews for small projects)
 - Ensure the key personnel (representatives of each group of stakeholder) are available
 - Have a competent evaluation team, ideally separate from the architects and developers
 - Managed expectations
- The results of the review need to be circulated to all stakeholders afterwards in draft form, with a ranked list of potential issues found.

What are the outputs?

- Prioritized Statement of Quality Attribute Requirements
 - Having a prioritized statement of the quality attributes serves as an excellent documentation record to accompany any architecture and guide it through its evolution
- Mapping of Approaches to Quality Attributes
 - Produces a mapping that shows how the architectural approaches achieve (or fail to achieve) the desired quality attributes
- Risks and Non-risks
 - Risks are potentially problematic architectural decisions
 - Non-risks are good decisions that rely on assumptions that are frequently implicit in the architecture

Advantages

- Forces an Articulation of Specific Quality Goals
- Results in the Prioritization of Conflicting Goals
- Puts Stakeholders in the Same Room
- Improves the Quality of Architectural Documentation
- Uncovers Opportunities for Cross-Project Reuse
- The average architecture evaluation adds no more than a few days to the project schedule (small % on total project time)
- Architecture created in haste will precipitate disaster: performance goals not met, Security goals failing, customer dissatisfaction, system that is too hard to change, and schedules and budgets through the roof

How to validate a software architecture?

- Most of the common evaluation methods evaluate only 1 Quality Attribute
- There are more advanced evaluation methods available to evaluate multiple Quality Attributes
- Identifying Trade-Offs
- A suite of three methods, all developed at the Software Engineering Institute.
 - ARID: Active Reviews for Intermediate Designs
 - SAAM: Software Architecture Analysis Method
 - ATAM: Architecture Tradeoff Analysis Method

Active Reviews for Intermediate Designs (ARID)

- Method for reviewing preliminary software designs (such as for a component or a subsystem) for suitability in its intended usage context and environment
- Result in a high-fidelity design review coupled with high-quality familiarization with the design
- Stakeholder Centric: Requires active stakeholder participation
- Easy and Lightweight approach
- Does not require complete documentation

ARID: Steps

- Identify Reviewers
 - The designer who has commissioned the review works with the ARID facilitator to identify the best reviewers. These are usually the software engineers who will be expected to use the design, as they are in the best position to judge its adequacy
- Overview & Presentation
 - The designer prepares a briefing explaining the design, which is reviewed by the ARID facilitator. The designer presents the overview to the reviewers and walks through examples of using the design. A scribe captures questions and answers
- Brainstorming
 - The reviewers brainstorm scenarios for using the design to solve problems they expect to face. After a facilitated prioritization, the resulting set of scenarios operationally defines what it means for the design to be usable: If it performs well under the adopted scenarios, then it must be agreed that the design has passed the review
- Artifacts
 - The reviewers begin to jointly craft code (or pseudo-code) that uses the design's services to solve the problem posed by each high-priority scenario. The scribe records issues, problems, and places where the stakeholders get stuck

ARID: Benefits

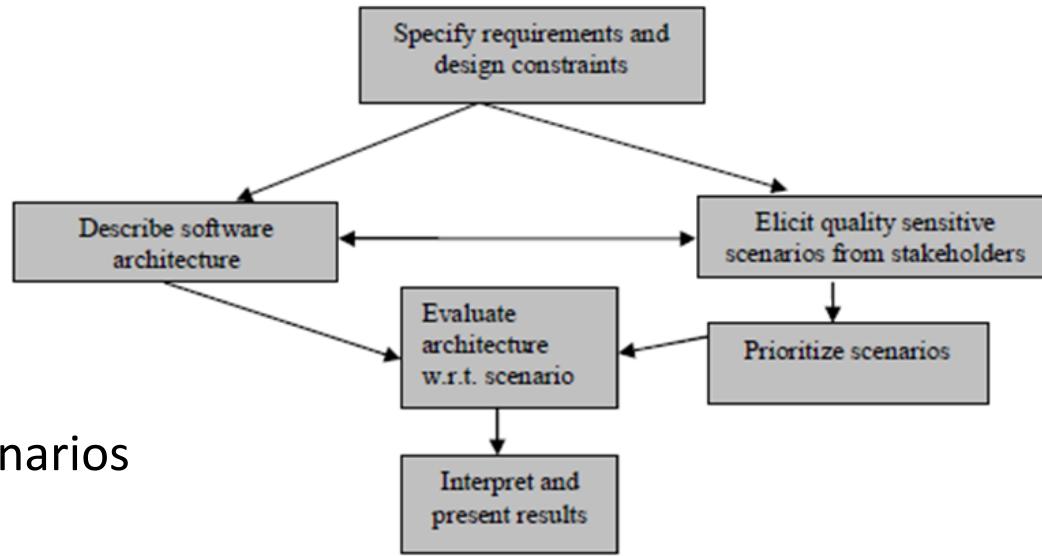
- Helps the designers of architecture engage stakeholders and get their buy-in early in the design process
- Informs designers about whether their design is suitable for the overall system being developed
- Valuable early insight into the design's viability and allows for timely discovery of errors, inconsistencies, and inadequacies

Software Architecture Analysis Method (SAAM)

- Aims to predict the quality of a system before it has been developed
- The quality of the architecture is validated by analyzing the impact of predefined scenarios on architectural components
- Addresses concerns at the architecture design level which inherently crosscut multiple architectural components
- Scenario based evaluation

SAAM: Steps

- Specify
 - Collects Requirements & Constraints and Develop Scenarios
- Describe Architectures
 - Present Candidate Architecture(s), static & dynamic representation of the system
- Elicit Scenarios
 - Scenarios are simulated with the presence of the relevant stakeholders for Brainstorming
- Classify and Prioritize Scenarios
 - Direct & Indirect Scenarios. Voting can be used to prioritize/identify what is most likely to occur
- Evaluate
 - Evaluate Architecture with respect to Scenarios. Impact on Scenarios for Architecture is exposed
- Results



SAAM: Benefits

- This analysis helps assess the risks inherent in an architecture
- Compare candidate software architectures
- Guides the inspection of the architecture, focusing on potential trouble spots such as requirement conflicts or incomplete design specification from a particular stakeholder's perspective

Architecture Tradeoff Analysis Method (ATAM)

- A structured technique for understanding the tradeoffs inherent in the architectures of software-intensive systems
- Provides a principled way to evaluate a software architecture's fitness with respect to multiple competing quality attributes
- Is a spiral model of design: one of postulating candidate architectures followed by analysis and risk mitigation, leading to refined architectures

ATAM

- Architecture Tradeoff Analysis Method

NEXT LECTURE:

Trade-offs

ATAM

Other Architecture Evaluation Methods

- SALUTA: Scenario-based Architecture Level Usability Analysis
- ALMA: Architecture Level Modifiability Analysis
- Software Architecture-based Reliability Analysis
- Software Architecture-based Performance Analysis
 - Analyze to estimate performance attributes quantitatively
- FAAM: Family-Architecture Assessment Method
 - Focus on two related quality aspects (e.g. interoperability and extensibility)

Mathematical Model-based Software Architecture Evaluation

- It is important to quantitatively assess operational quality attributes
- These methods model software architectures using well-known mathematical equations
- Then, these methods use the models to obtain architectural statistics, for instance, mean execution time of a component
- These architectural statistics are used to estimate operational quality attributes. Reliability and performance are two important operational quality attributes

Late Evaluation Methods Applied to Software Architecture

- Late software architecture evaluation methods identify the difference between the actual and planned architectures
- These methods provide useful guidelines of how to reconstruct the actual architecture, so that it conforms to the planned architecture
- During the testing phase, late software architecture evaluation methods are also applied to check the compliance of the source code to the planned design

References

- <https://www.sei.cmu.edu/architecture/tools/evaluate/>
- <http://www.win.tue.nl/oas/architecting/aimes/papers/Scenario-Based%20SWA%20Evaluation%20Methods.pdf>



Trade-off Analysis

**Software Architecture
3rd Year – Semester 1
Lecture 15**

What is a Trade-off?

- A trade-off is a situation that involves losing one quality or aspect of something in return for gaining another quality or aspect
- How much must I give up to get a little more of what I want most?

What is ATAM

- ATAM (Architecture Trade-Off Analysis Method) is an architecture evaluation method developed in the Software Engineering Institute at the end of the 90s
- The objective of Architecture Trade-off Analysis (ATAM) is to provide a justifiable way to understand a Software Architecture's fitness with respect to multiple competing Quality Attributes

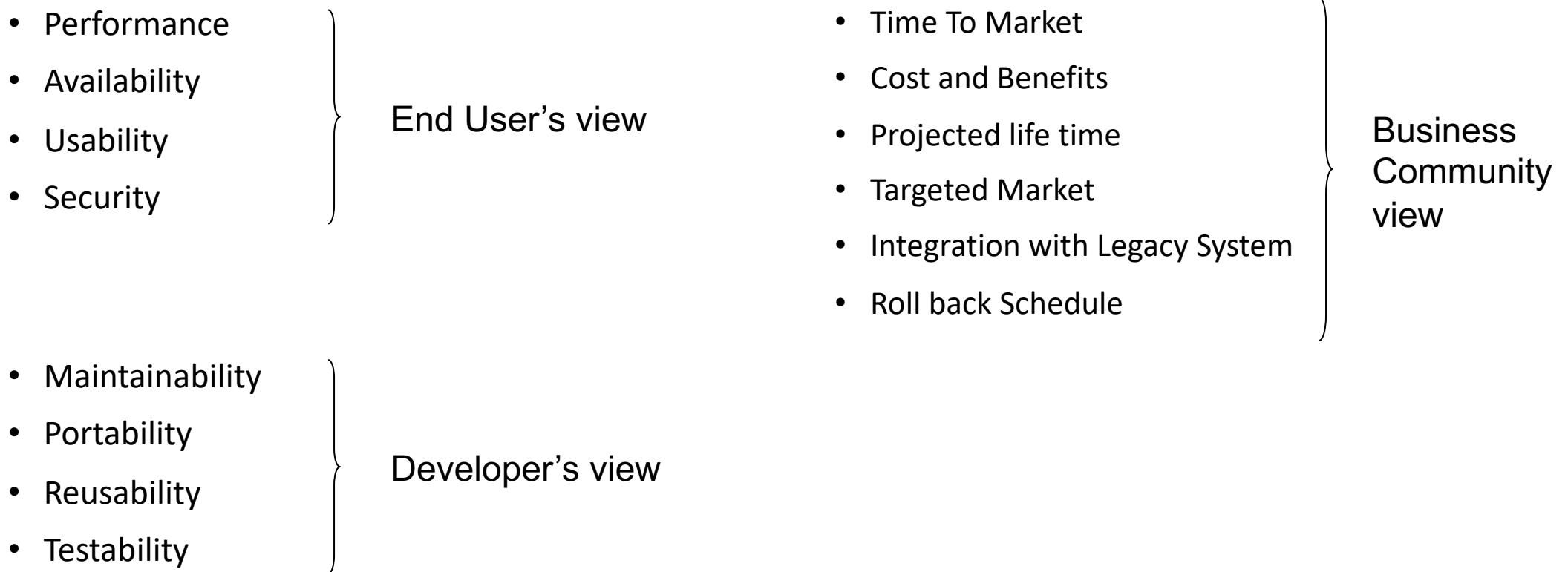
Why ATAM?

- The purpose of ATAM is to assess the consequences of architectural design decisions in the light of quality attributes
- ATAM helps in foreseeing how an attribute of interest can be affected by an architectural design decision
- The Quality Attributes of interest are clarified by analyzing the stakeholder's scenarios in terms of stimuli and responses
- Once the scenarios had been defined, the next step is to define which architectural approaches can affect to those quality attributes

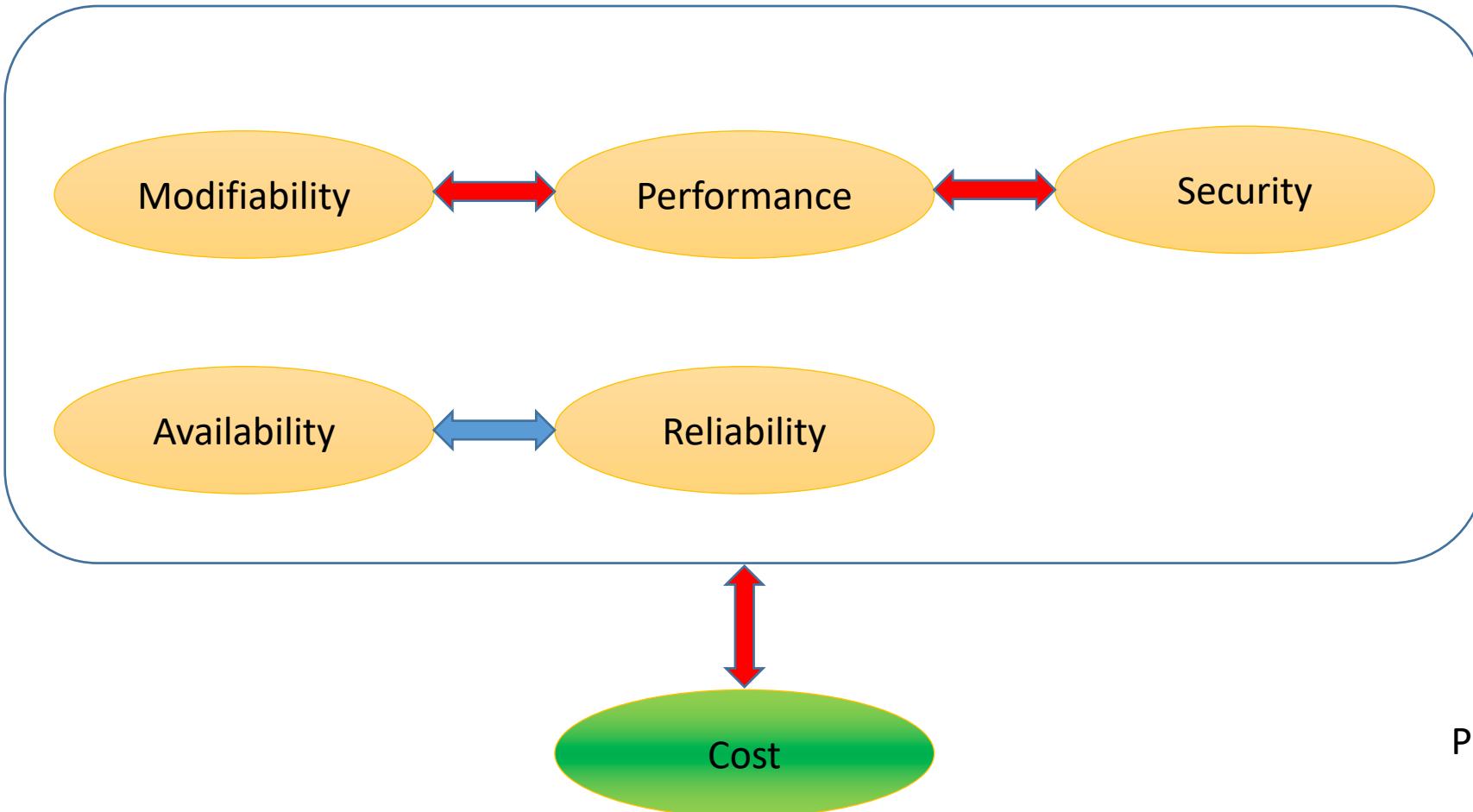
Quality Attributes and ATAM

- Quality Attributes can interact or conflict – improving one often comes at the price of worsening one or more of the others, thus it is necessary to trade off among multiple Software Quality Attributes at the time the Software Architecture of a system is specified, and before the system is developed
- ATAM helps above process

Stakeholder Expectations



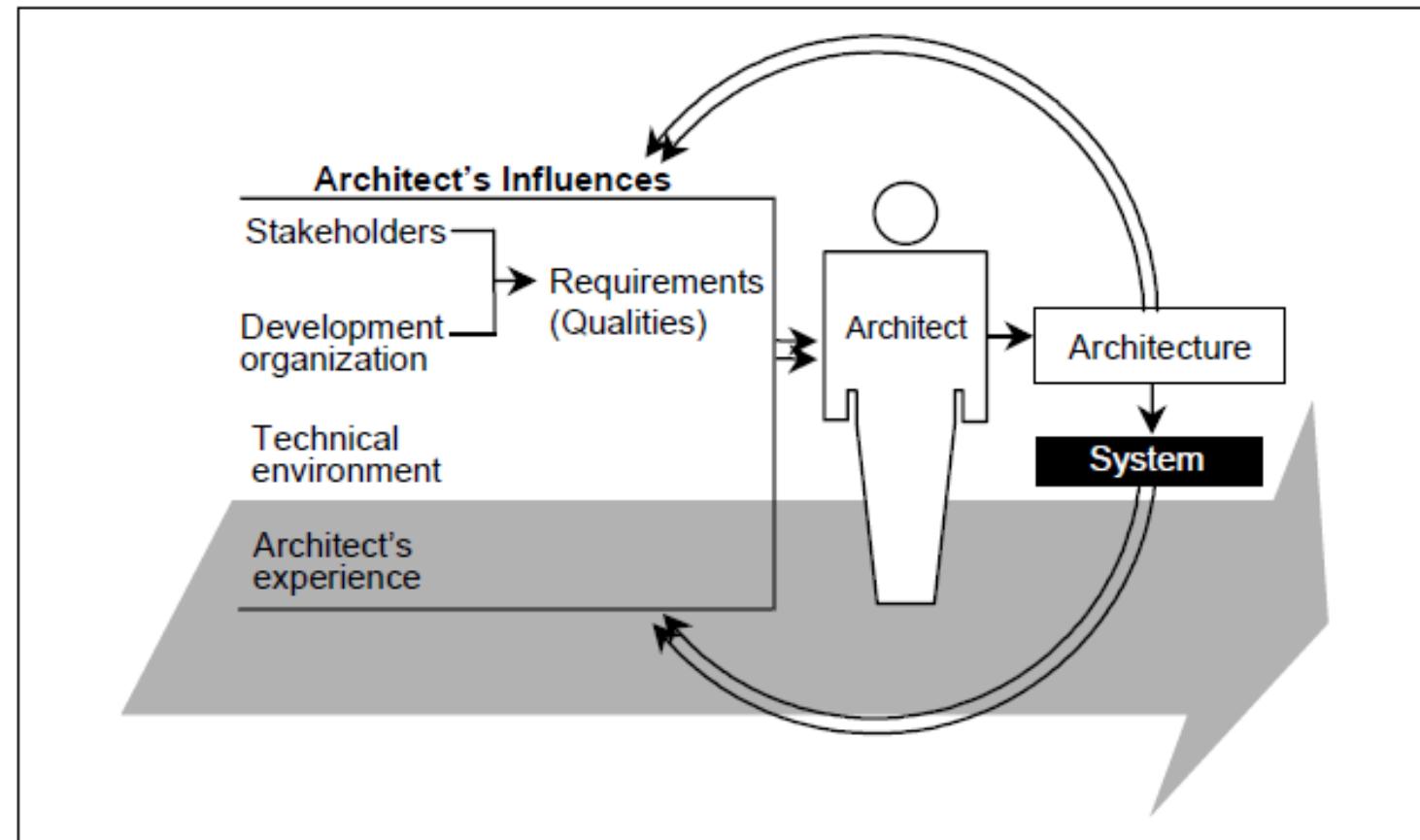
Attribute Impact



Project Management Triangle

Architect's Role & Influence

- Envision
- Realize
- Influence



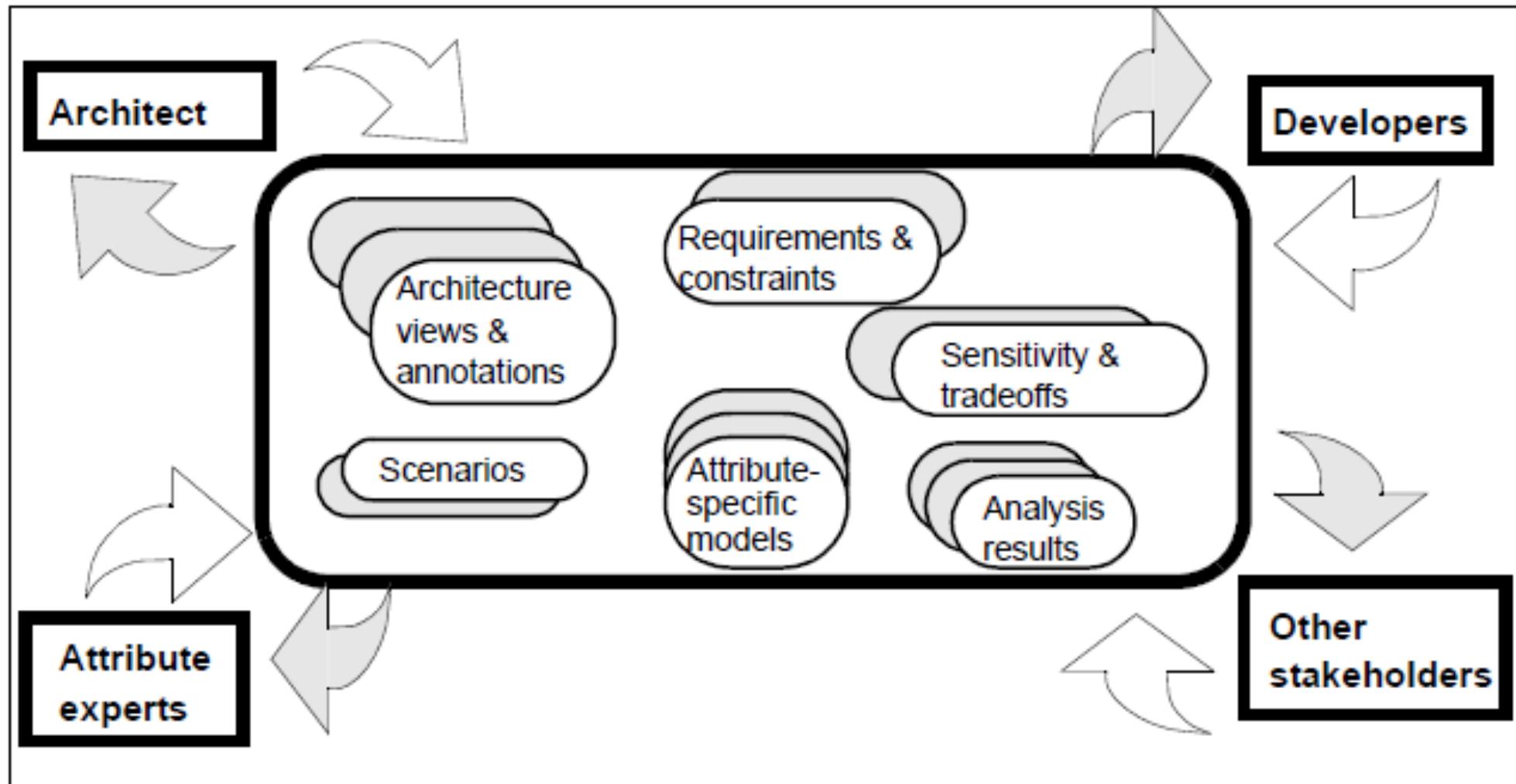
The Architecture Business Cycle

- Software architecture is a result of Technical, Business, and Social influences. Its existence in turn affects the technical, business, and social environments that subsequently influence future architectures. We call this cycle of influences, from the environment to the architecture and back to the environment, the Architecture Business Cycle
 - How organizational goals influence requirements and development strategy.
 - How requirements lead to an architecture.
 - How architectures are analyzed.
 - How architectures yield systems that suggest new organizational capabilities and requirements

Architect's Roles & Responsibilities

- Abstracts the complexity of a system into a manageable model that describes the essence of a system by exposing important details and significant constraints
- Sets quantifiable objectives that encapsulate quality attributes of a system
- Maintains control over the architecture lifecycle parallel to the project's software development lifecycle
- Stays on course in line with the long term vision
- Makes critical decisions that define a specific direction for a system in terms of implementation, operations, and maintenance
- Creates and distributes tailored views of software architectures to appropriate stakeholders at appropriate intervals
- Works closely with executives to explain the benefits and justify the investment in software architecture of a solution
- Acts as an agent of change in organizations where process maturity is not sufficient for creating and maintaining architecture centric development
- Inspires, mentors, and encourages colleagues to apply intelligently customized industry's best practices

Supporting roles for ATAM



Challenges

Most complex software systems are required to be modifiable and have good performance. They may also need to be secure, interoperable, portable, and reliable.

For any particular system:

- What precisely do these quality attributes such as modifiability, security, performance, and reliability mean?
- Can a system be analyzed to determine these desired qualities?
- How soon can such an analysis occur?
- How do you know if a software architecture for a system is suitable without having to build the system first?

Flow of ATAM



Participants of ATAM

- The Evaluation Team
 - A group external to the project. The team might be consultants or other members of the organization.
 - Typically 3–5 members, each member is assigned to a number of specific roles.
- Project Decision Makers
 - Empowered to speak for the development of project and have the authority to mandate changes to it.
 - Most importantly, the architect must be present. 2 or more members (e.g. Project Manager, Customer,...)
- Architecture Stakeholders
 - With a vested interest in the architecture performing correctly.
 - These are the stakeholder affected by the architecture, usually 12–15 members. (e.g. Modifiability – Developers, Maintainability & Security – System Administrators, etc...)

Phases of ATAM

- Phase 0: Partnership and Preparation
 - Assemble the Evaluation Team, leadership and key project decision makers
 - Prepares Architecture Description & Scenarios
- Phase 1: Evaluation – by Evaluation Team
 - Evaluation ground rules are presented
 - Business drivers (including quality attribute goals) are presented
 - The architecture itself is presented
 - Architectural approaches are identified
 - Quality attribute goals are prioritized and refined into specific scenarios
 - Architectural approaches are analyzed to determine how they satisfy the refined quality goals
- Phase 2: Evaluation – includes Stakeholders
- Phase 3: Follow-up
 - Addresses the issues raised
 - Deliver Final Report

Phase 1 - Steps

- Step 1 - Present the ATAM
 - Evaluation leader explain the process that everyone will be following and set the context and expectations
- Step 2 - Present Business Drivers
 - Decision Makers (Project Manager or Customer) presents a system overview from a business perspective
 - The system's most important functions
 - Any relevant technical, managerial, economic or political constraints
 - The business goals and context as they relate to the project
 - The major stakeholders
 - The architectural drivers (major quality attribute goals that shape the architecture)
- Step 3 - Present the Architecture
 - Architect(s) presents the architecture covering technical constraints such as operating system, hardware, or middleware prescribed for use, and other systems with which the system must interact
- Step 4 - Identify Architectural Approaches
 - Architecture Approaches or Patterns are presented at a High-Level to cover the essence of the Architecture
- Step 5 - Generate Quality Attribute Utility Tree
 - Present the quality attribute goals in detail
 - Expressed as Concrete Quality Attribute Scenarios & assign priorities
- Step 6 - Analyze Architectural Approaches
 - Determine how the approaches satisfy the refined quality goals
 - Architectural risks, sensitivity points, and tradeoff points are identified

Phase 2 - Steps

- Step 7 - Brainstorm and Prioritize Scenarios
 - A larger set of scenarios are stimulated with the entire group of stakeholders
 - Set of scenarios are prioritized via a voting process
- Step 8 - Analyze Architectural Approaches
 - Same as Step #6 but High Priority Scenarios are taken for analysis with the larger group of stakeholders
 - Document additional architectural approaches, risks, sensitivity points, and tradeoff points
- Step 9 - Present Results
 - Presents the findings to the assembled stakeholders

Outputs of ATAM

- A concise presentation of the architecture.
- Articulation of the business goals.
- A set of quality requirements in the form of quality scenarios.
- Mapping of architectural decisions to quality requirements. This specifies how the architecture supports each of the quality scenarios.
- A set of identified sensitivity and tradeoff points. Architecture decisions that affect qualities are identified along with how they cause tradeoff (e.g. performance vs. reliability).
- A set of risks and non-risks. This will be the basis of a risk mitigation plan.
- A set of risk themes. The systemic weakness that explain the risks.

ATAM - Benefits

- Identified risks early in the life cycle
- Increased communication among stakeholders
- Clarified quality attribute requirements
- Improved architecture documentation
- Documented basis for architectural decisions

ATAM – Example #1

- Introduction
 - The Case Study software architecture we are going to evaluate has a component called “game space”
 - This “game space” services a game to 2 different game engines: one is a commercial off-the-shelf game engine (Torque1) and the second is a bespoke simulation engine
 - Using a “game space” allows adding a new game engine without the need to modify the game to suit the new engine

Case Study Reference: [Using ATAM to Evaluate a Game-based Architecture](#)



Adobe Acrobat
Document

<http://www.cs.rug.nl/~paris/ACE2006/papers/BinSubaih.pdf>

ATAM – Example #1

- Game based architecture
 - Developing games using game engines
- Problems
 - Game is developed in the game engine format – game is available for only that engine.
 - Have to modify the game when running on different game engine.
- Examines the suitability of employing ATAM to evaluate a game-based architecture for evaluating portability between game engines

Applying ATAM – Phase 1 | Step #1

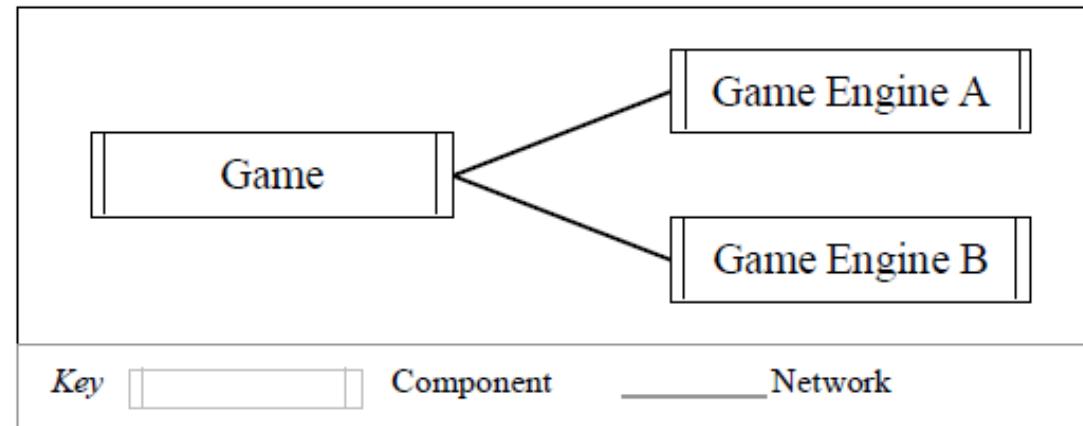
- Present ATAM
 - The first step starts by presenting the ATAM process to the stakeholders.
 - Explain the Process to the Evaluation Team
 - The evaluation leader describes the evaluation method to the assembled participants, tries to set their expectations, and answers questions they may have.

Applying ATAM – Phase 1 | Step #2

- Present Business Drivers
 - Business goals of the architecture
 - Break the current tightly coupled practices employed when developing games using game engines
 - Make the architecture flexible to accommodate different games
 - Describe
 - The system's most important functions
 - Any relevant technical, managerial, economic, or political constraints
 - The business goals and context as they relate to the project
 - The major stakeholders
 - The architectural drivers (the major quality attribute goals)

Applying ATAM – Phase 1 | Step #2

- Present Business Drivers
 - Quality Attributes
 - Portability - aims to run the same game on multiple game engines without modifying the game itself.



The portability attribute succeeds if the same game can be serviced to multiple game engines without any modification to the game

Applying ATAM – Phase 1 | Step #2

- Present Business Drivers
 - Quality Attributes
 - Modifiability - measured by the amount of changes required to the different components of the architecture and the less changes needed the better the modifiability is.
 - Performance - The stimuli can be user interactions or messages arriving over the network or method calls from other components or programs. The acceptable time to response for a stimulus is less than one second and the acceptable display rate at the game engine is no less than 20 frames per second

Applying ATAM – Phase 1 | Step #3

- Present the Architecture
 - Driving architectural requirements, measurable quantities associated with these, standards/models/approaches for meeting these
 - Important architectural information
 - Context diagram
 - Module or layer view
 - Component and connector view
 - Deployment view

Applying ATAM – Phase 1 | Step #3

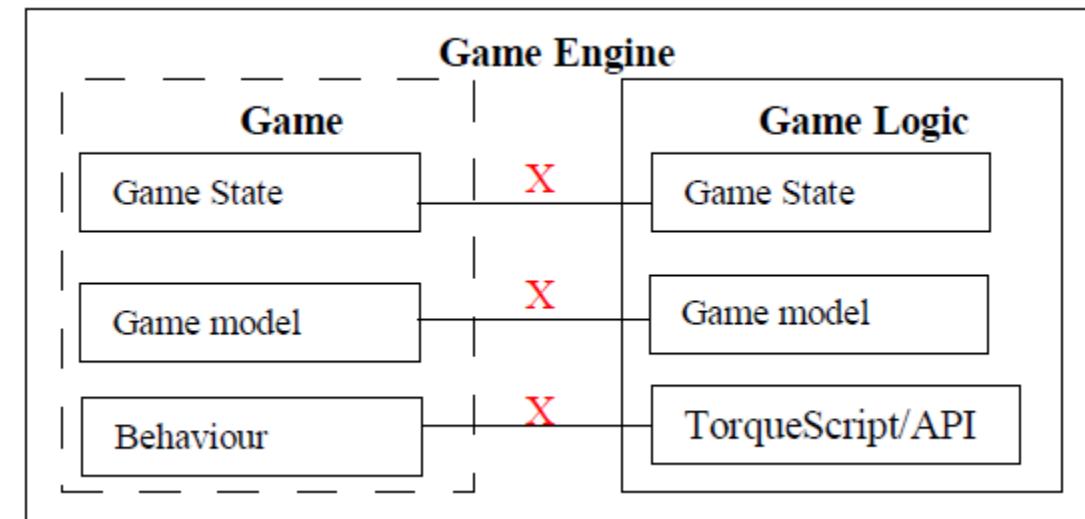
- Present the Architecture
 - Architectural approaches, patterns, tactics employed, what quality attributes they address and how they address those attributes
 - Use of COTS and their integration
 - Most important use case scenarios
 - Most important change scenarios
 - Issues/risk for meeting the diving requirements

Applying ATAM – Phase 1 | Step #4

- Identify Architectural Approaches
 - Catalog the evident patterns and approaches
 - Based on step 3
 - Serves as the basis for later analysis

Applying ATAM – Phase 1 | Step #4

- Identify Architectural Approaches
 - Architectural decisions to support portability
 - Have the game state living outside the game engine's game state and find a way to communicate between the two states
 - Model view controller (MVC) pattern
 - Asynchronous messaging



Applying ATAM – Phase 1 | Step #4

- MVC Architecture
 - Breaks dependency links – Portability
 - Separates core from view – Modifiability
 - Anticipate expected changes
 - Semantic coherence
 - Prevent ripple effects
 - The introduction of layers by using MVC to promote portability and modifiability also adds processing overhead on the architecture as information needs to be passed and sometimes translated between the layers.
 - Placing adapters
 - in its own application on its own machine – network overhead
 - in its own application on either the game space machine or the game engine machine - extra overhead to communicate across applications
 - placing the adapter in the same application as the game space - least overhead expense

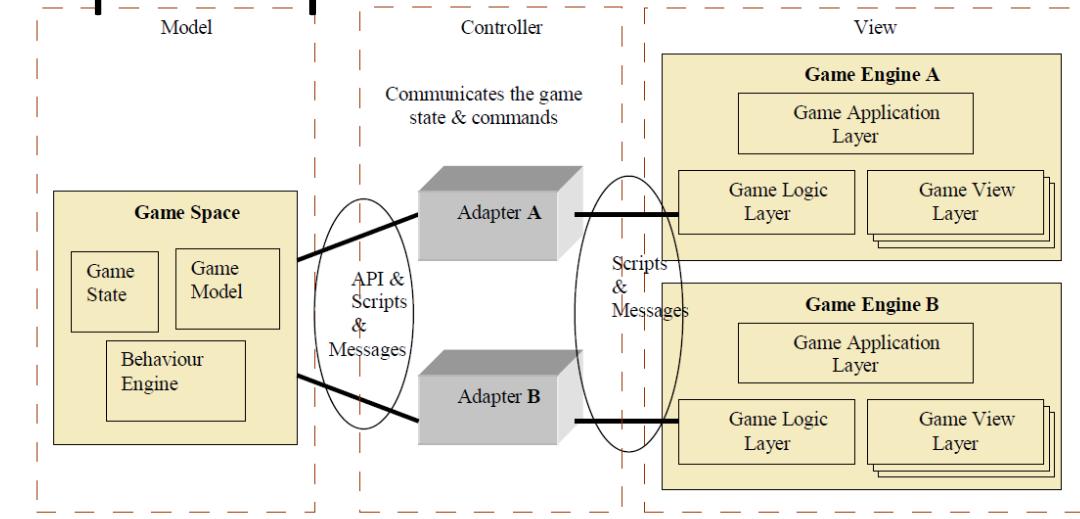


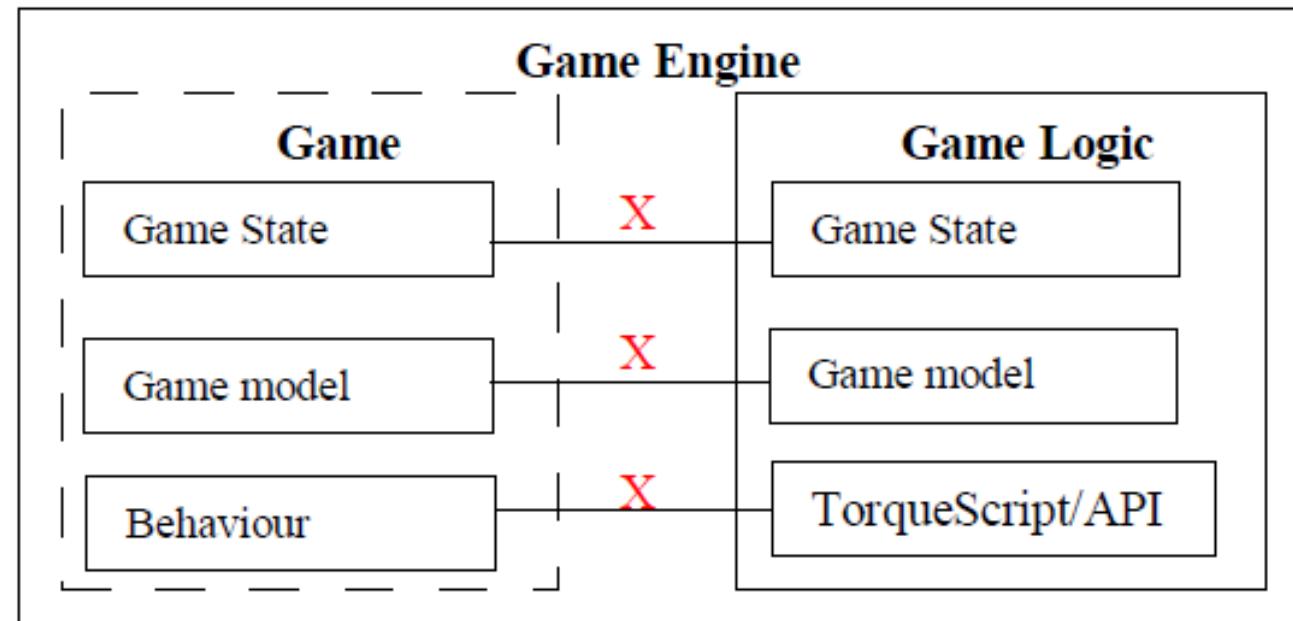
Figure 3: Game-based architecture using the MVC pattern

Applying ATAM – Phase 1 | Step #4

- Asynchronous Messaging
 - Synchronization of the game states
 - Messages from behaviour engine to be carried out to the game engine
 - To compensate for the network overhead the architecture uses an asynchronous communication mechanism to reduce the impact on the display rate.

Applying ATAM – Phase 1 | Step #4

- Architectural Decisions
 - To support portability: not to have a game model that is only accessible through the game engine's interface but have a game model that can be accessed outside the game engine
- Ontologies
- Mid-game scripting



Applying ATAM – Phase 1 | Step #4

- Ontologies
 - Ontologies allow creating concepts which have properties, and relations. We use these to create the classes of our domain. The concept becomes the class and its properties are the properties of the class and finally the relationships describe things like inheritance and association.
 - Used to specify the game knowledge representation independently from the game engine's game model.
 - Allows building classes on-the-fly

Applying ATAM – Phase 1 | Step #4

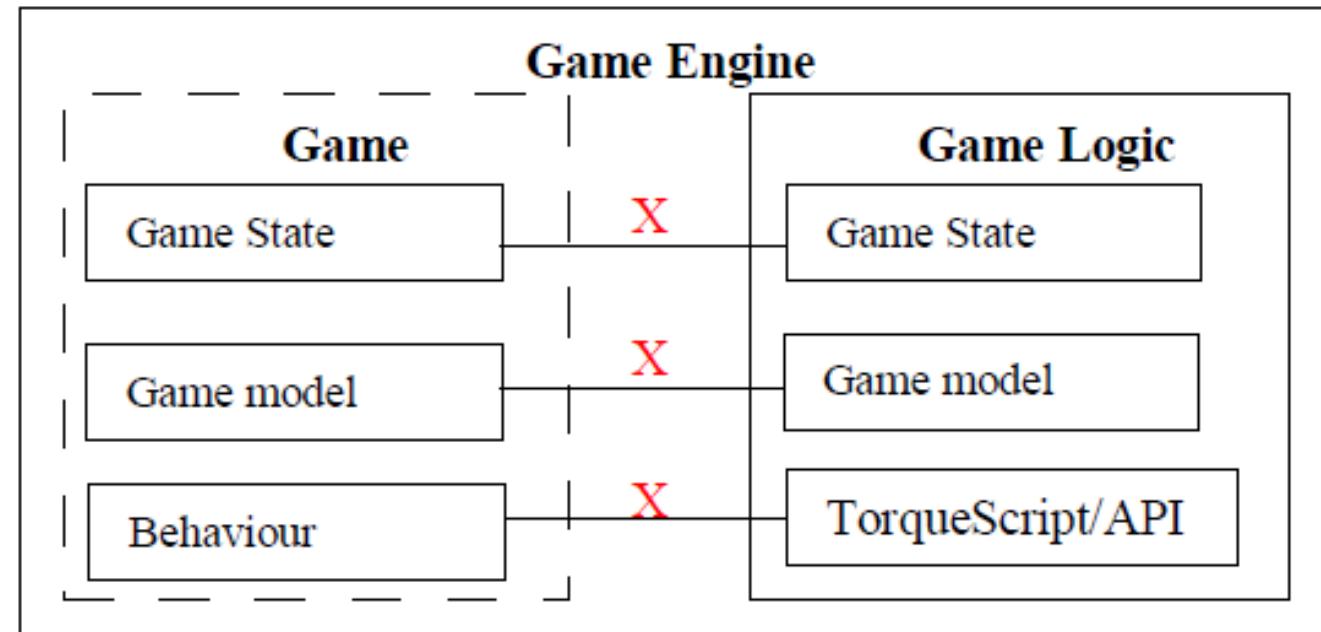
- Scripting
 - Enables manipulating the game state and setting the behaviour without recompiling the game space system.
 - Can modify game engine without affecting the game engine code
 - Mid-game scripting is used instead of pre-compiled scripting since using the latter would require some of the game logic to be put in the game engine's specific format.
 - For better modifiability mid-game scripting was used despite the fact that mid-game scripting runs at much slower speed than the precompiled code

Applying ATAM – Phase 1 | Step #4

- Architectural Decisions

- To support Portability: final flag should be raised when the game engine requires the behavior to be specified in the game engine's own language or format.

- Objects mapping table
- API



Applying ATAM – Phase 1 | Step #4

- API and Mapping
 - API - to interact with the game space through code or through scripting
 - Objects mapping table – link the corresponding objects on both sides

Applying ATAM – Phase 1 | Step #5

- Generate Quality Attribute Utility Tree
 - Utility Tree
 - Present the quality attribute goals in detail
 - Quality attribute goals
 - Identified, prioritized, refined
 - Expressed as scenarios
 - Utility is an expression of the overall goodness of the system
 - Quality attributes form the second level being components of utility
 - Prioritize Scenarios
 - Depending on how important they are
 - Depending on how difficult it will be for the architecture to satisfy a scenario

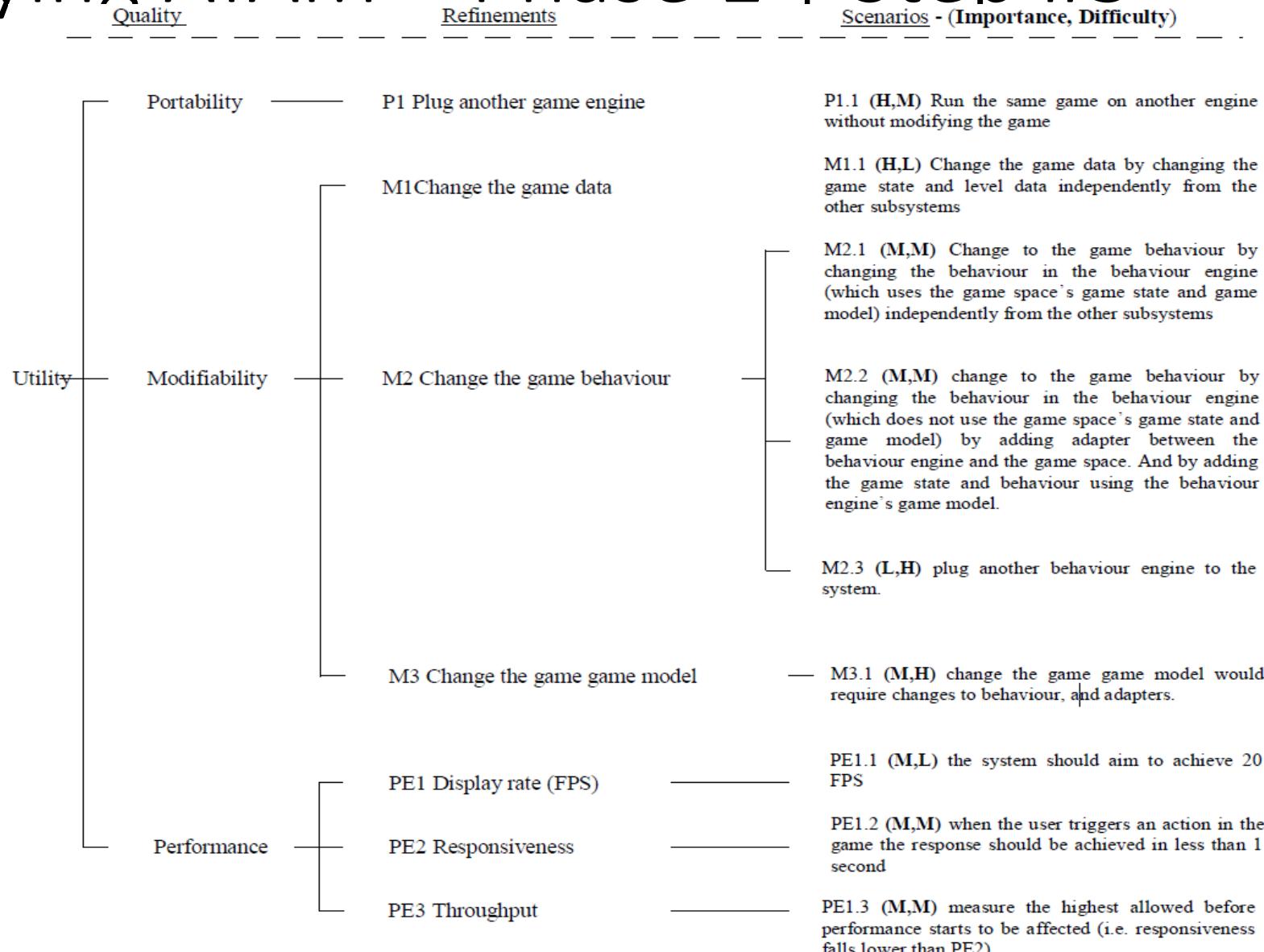
Applying ATAM – Phase 1 | Step #5

- Generate Utility Tree
 - Identify, prioritize, and refine the most important quality attribute goals by building a utility tree.
 - A utility tree is a top-down vehicle for characterizing the “driving” attribute-specific requirements
 - Select the most important quality goals to be the high-level nodes (typically performance, modifiability, security, and availability)
 - Scenarios are the leaves of the utility tree
 - Output: a characterization and a prioritization of specific quality attribute requirements.
 - Scenarios are used to represent stakeholders’ interests
 - Understand quality attribute requirements
 - A good scenario makes clear what the stimulus is that causes it and what responses are of interest.

Applying ATAM – Phase 1 | Step #5

- Quality Attribute Utility Tree Template
 - Quality Attribute Level
 - Refinements Level
 - The aim of the refinement is to decompose the quality attribute further if possible
 - Scenario Level
 - Holds the scenarios in the Concrete form
 - Holds Scenario Rankings
 - Importance/Priority – High/Medium/Low or 1 – 10 rating
 - Difficulty Factor – High/Medium/Low or 1 – 10 rating

Applying ATAM – Phase 1 | Step #5



Applying ATAM – Phase 1 | Step #6

- Analyze Architectural Approaches
 - Examine the highest ranked scenarios
 - The goal is for the evaluation team to be convinced that the approach is appropriate for meeting the attribute-specific requirements
 - Scenario walkthroughs
 - Identify and record a set of sensitivity points and tradeoff points, risks and non-risks
 - Sensitivity and tradeoff points are candidate risks

Applying ATAM – Phase 1 | Step #6

- Concrete Quality Attribute Scenarios & assign priorities: Portability

Analysing Scenario	P1.1			
Scenario	plug another game engine to the architecture			
Attributes	Portability			
Stimulus	running the same game on another game engine			
Environment				
Response	should be achieved by adding an adapter to connect the game space and the newly added game engine			
Architecture Decision	Sensitivity	Tradeoff	Risk	Nonrisk
AD1 MVC		T1	R1,R2	N1
AD2 Messaging	S1,S2	T2	R3	N2
AD3 Mid-game scripting		T3	R4	N3
AD4 Ontologies		T4		
AD6 COTS behaviour engine	S1,S2		R2	N4

Applying ATAM – Phase 1 | Step #6

Sensitivities:

- S1: Concern over network latency
- S2: Concern over messages load
- S3: In the event that a single unique identifier cannot be set this architecture decision becomes very sensitive to any modification as they have to be added manually to the table
- S4: Using ontologies allows for better game model scalability but it makes the architecture very sensitive to change as the change propagates to behaviour and adapters.

Tradeoffs:

- T1: Portability (+) and Modifiability (+) vs. Performance (-) – separating the architecture into layers adds an overhead for exchanging information between layers which affects the performance.
- T2: Portability (+) vs. Performance (-)
- T3: Portability (+) and Modifiability (+) vs. Performance (-) – mid-game scripting allows better portability and modifiability but runs at 10x slower than pre-compiled code.
- T4: Portability (+) and Modifiability (+) vs. Performance (-) – using ontologies allows for having a game model that is independent from the game engine's game model however it has adverse effect on performance.
- T5: Modifiability (-) vs. Performance (+)

Risks:

- R1: The risk is caused by the tight coupling between the controller and the model which is a known liability of using this pattern.
- R2: Data integrity
- R3: The risk is introduced by the tight coupling as a consequence of using pre-determined messages for the messages arriving from the game engine which means changes to them would require deployment of new system every time.
- R4: The risk raised is a consequence the game engine or the game space which might not have fully exposed their functionality through the scripting.
- R5: If no unique id can be set this means the mapping table should be done manually.

Nonrisks:

- N1: The nonrisk exists because of the decision taken to remove the other liability of using MVC – the removal of the direct link between the view and the controller described earlier.
- N2: The nonrisk risk exists because of using asynchronous mechanism as it avoids negative impact on the frame rate that could be caused by the synchronous mechanism
- N3: The nonrisk is the use of API approach which should stay compatible.
- N4: assume that the behaviour engine requires the game state to be replicated to its own working memory
- N5: A one-to-one mapping for objects across both game states is established by having a unique identifier and if that is not possible the objects mapping table handles that.
- N6: Should stay compatible

Applying ATAM – Phase 1 | Step #6

- Sensitivities:
 - S1: Concern over network latency
 - S2: Concern over messages load
 - S3: In the event that a single unique identifier cannot be set this architecture decision becomes very sensitive to any modification as they have to be added manually to the table
 - S4: Using ontologies allows for better game model scalability but it makes the architecture very sensitive to change as the change propagates to behaviour and adapters.

Applying ATAM – Phase 1 | Step #6

- Tradeoffs:
 - T1: Portability (+) and Modifiability (+) vs. Performance (-) – separating the architecture into layers adds an overhead for exchanging information between layers which affects the performance.
 - T2: Portability (+) vs. Performance (-)
 - T3: Portability (+) and Modifiability (+) vs. Performance (-) – mid-game scripting allows better portability and modifiability but runs at 10x slower than pre-compiled code.
 - T4: Portability (+) and Modifiability (+) vs. Performance (-) – using ontologies allows for having a game model that is independent from the game engine's game model however it has adverse effect on performance.
 - T5: Modifiability (-) vs. Performance (+)

Applying ATAM – Phase 1 | Step #6

- Risks:
 - R1: The risk is caused by the tight coupling between the controller and the model which is a known liability of using this pattern.
 - R2: Data integrity
 - R3: The risk is introduced by the tight coupling as a consequence of using pre-determined messages for the messages arriving from the game engine which means changes to them would require deployment of new system every time.
 - R4: The risk raised is a consequence the game engine or the game space which might not have fully exposed their functionality through the scripting.
 - R5: If no unique id can be set this means the mapping table should be done manually.

Applying ATAM – Phase 1 | Step #6

- Non-risks:
 - N1: The non-risk exists because of the decision taken to remove the other liability of using MVC – the removal of the direct link between the view and the controller described earlier.
 - N2: The non-risk risk exists because of using asynchronous mechanism as it avoids negative impact on the frame rate that could be caused by the synchronous mechanism
 - N3: The non-risk is the use of API approach which should stay compatible.
 - N4: assume that the behaviour engine requires the game state to be replicated to its own working memory
 - N5: A one-to-one mapping for objects across both game states is established by having a unique identifier and if that is not possible the objects mapping table handles that.
 - N6: Should stay compatible

Applying ATAM – Phase 1 → Phase 2

- After Phase 1, there is a detailed set of quality requirements with the architectural decision tradeoffs that affect those requirements.
- In Phase 2,
 - The stakeholders help to prioritize the existing scenarios and propose new ones.
 - The architectural tradeoffs are again analyzed, now in consultation with the stakeholders.
- The results of the review are presented to all.
- During Phase II, in addition to brainstorming on quality scenarios, the stakeholders provide the in-depth knowledge of whether the proposed architectural tactics will actually meet their expectations.

Applying ATAM – Phase 1 → Phase 2

- Scenario Examples
 - Use case scenario
 - Remote user requests a database report via the Web during peak period and receives it within 5 seconds.
 - Growth scenario
 - Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week.
 - Exploratory scenario
 - Half of the servers go down during normal operation without affecting overall system availability.
 - Scenarios should be as specific as possible

Applying ATAM – Phase 1 → Phase 2

Type	Scenario
Use case	<ul style="list-style-type: none">- Run two games from the same domain- Run two games from two different domains
Growth	<ul style="list-style-type: none">- Interoperate between two game engines- Run two behaviour engines (internal and external)
Exploratory	<ul style="list-style-type: none">- Increase the load by increasing the number of NPCs the user can interact with- Increase the load by simulating multiple number of simultaneous players- Run the game space on the same machine as the game engine- Run the behaviour engine and the game space on the same machine as the game engine

Applying ATAM – Phase 1 → Phase 2

- Sensitivity & Tradeoffs
 - Stakeholders will join form this step.
 - Sensitivity:
 - A property of a component that is critical to success of system.
 - The number of simultaneous database clients will affect the number of transaction a database can process per second. This assignment is a sensitivity point for the performance
 - Keeping a backup database affects reliability.
 - Power of encryption (Security) sensitive to number of bits of the key
 - Tradeoff point:
 - A property that affects more than one attribute or sensitivity point.
 - In order to achieve the required level of performance in the discrete event generation component, assembly language had to be used thereby reducing the portability of this component.
 - Keeping the backup database affects performance also so it's a trade-off between reliability and performance.

Applying ATAM – Phase 1 → Phase 2

- Risks & Non-Risks
 - Risks
 - The decision to keep backup is a risk if the performance cost is excessive
 - Rules for writing business logic modules in the second tier of your 3-tier style are not clearly articulated. This could result in replication of functionality thereby compromising modifiability of the third tier.
 - Non-Risks
 - The decision to keep backup is a non-risk if the performance cost is not excessive
 - Assuming message arrival rates of once per second, a processing time of less than 30 ms, and the existence of one higher priority process, a 1 second soft deadline seems reasonable Performance.

Applying ATAM – Phase 2 | Step #7

- Brainstorm & Prioritize Scenarios
 - Utility tree shows architects view on the quality attributes
 - Stakeholders generate scenarios using a facilitated brainstorming process.
 - Scenarios at the leaves of the utility tree serve as examples to facilitate the step.
 - The new scenarios are added to the utility tree
 - Here the focus is on the other stakeholders view on the quality attributes and scenarios based on these
 - Which are the most meaningful and important scenarios for users etc.

Applying ATAM – Phase 2 | Step #8

- Analyze Architectural Approaches
 - Highest ranked scenarios from Step #7 are presented to the architect
 - Explain how relevant architectural decisions contribute to realizing each one

Applying ATAM – Phase 2 | Step #9

- Present Results (Output)
 - The architectural approaches documented
 - The set of scenarios and their prioritization from the brainstorming
 - The utility tree
 - The risks discovered
 - The non-risks documented
 - The sensitivity points and tradeoff points found

Applying ATAM – Phase 2

- Benefits and Observations
 - Technical participants in ATAM are typically amazed at how many risks can be found in a short time.
 - Managers appreciate the opportunity to see precisely how technical issues threaten achievement of their business goals.

References

- <https://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm>
- <http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Three+Analyzing+Architectures/Chapter+11.+The+ATAM+A+Comprehensive+Method+for+Architecture+Evaluation/>
- Using ATAM to Evaluate a Game-based Architecture -
<http://www.cs.rug.nl/~paris/ACE2006/papers/BinSubaih.pdf>

Exercise (offline)

- Refer a few existing software systems (e.g. your Group Case Study) and identify what are the important quality attributes for that system
 - Identify about 2-3 Quality Attributes
 - Write Concrete Quality Attribute Scenarios
 - Generate Quality Attribute Utility Tree
 - Identify Sensitivity points, Trade-offs, Risks and Non-Risks



Software Architecture Frameworks

**Software Architecture
3rd Year – Semester 1
Lecture 16**

Why Software Architecture

- Provides standard governing structure
- Provide solutions to known problems
- Helps to make projects successful
 - Addresses: Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk
- Makes products easy to maintain

When do we need to “Architect”

- When the Solution(s) gets bigger
 - Modern Software are more complex than what was before
- When you have to think about the future
 - Software lasts longer, they are no longer “throw-away” items - specially on Data
- Increased Usage and Usage Types
 - Earlier only direct users interacted with Software, now Systems interact with each other

Enterprise Architecture

- A well-defined practice for conducting enterprise analysis, design, planning, and implementation, using a holistic approach at all times, for the successful development and execution of strategy
- Applies architecture principles and practices to guide organizations through the business, information, process, and technology changes necessary to execute their strategies

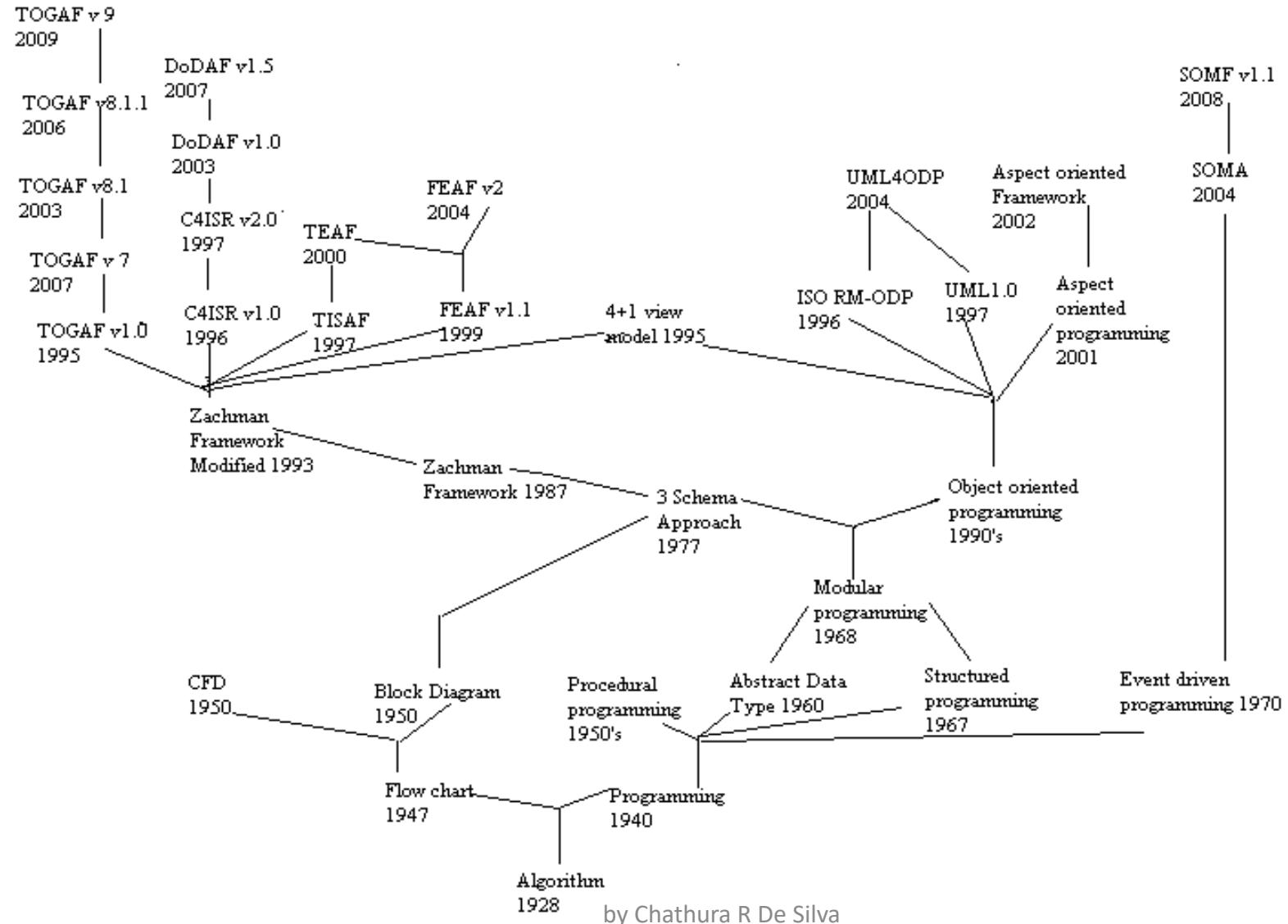
Benefits of Enterprise Architecture

- Business Benefits
 - Helps an Organization achieve its business strategy
 - Faster time to market for new innovations and capabilities
 - More consistent business process and information across business units
 - More reliability and security, less risk
- IT Benefits
 - Better traceability of IT costs
 - Lower IT costs – design, buy, operate, support, change
 - Faster design and development
 - Less complexity
 - Less IT risk

Key Architecture Principles

- Build to change instead of building to last
 - Consider how the application may need to change over time to address new requirements and challenges
 - Build with flexibility to adopt changes
- Model to analyze and reduce risk
 - Use design tools to visualize e.g. UML
 - Capture requirements and architectural and design decisions and to analyze their impact
 - Do not formalize the model to the extent that it suppresses the capability adapt easily
- Communication and Collaboration
 - Use visualizations of the architecture to communicate and share your design efficiently with all the stakeholders, and to enable rapid communication of changes to the design
- Identify key engineering decisions
 - Understand the key engineering decisions and the areas where mistakes are most often made
 - Invest in getting these key decisions right the first time (late changes are always costly)

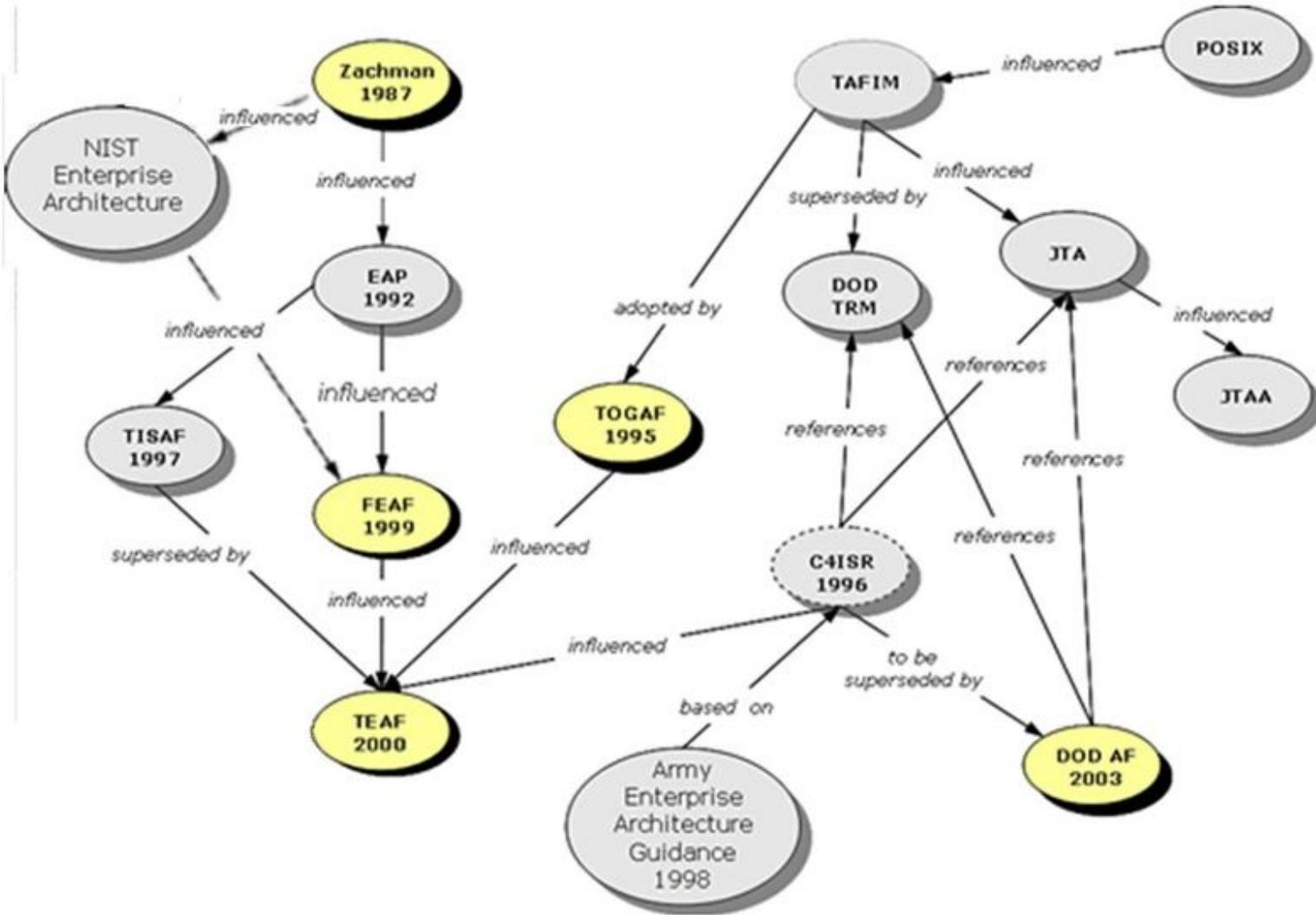
Architecture Frameworks - How it evolved



Enterprise Architecture Frameworks

- Zachman Framework
 - In 1982, when working for IBM and with BSP, John Zachman was perhaps the first to mention Enterprise Architecture in the public domain. In 1987, John Zachman, who was a marketing specialist at IBM, published the paper, A Framework for Information Systems Architecture. The paper provided a classification scheme for artifacts that describe the what, how, where, who, when and why of information systems.
- The Open Group Architecture Framework (TOGAF)
 - In 1994, the Open Group selected TAFIM from the US DoD as a basis for development of The Open Group Architecture Framework (TOGAF), where architecture meant IT architecture.

Architecture Frameworks – Mutual Influence

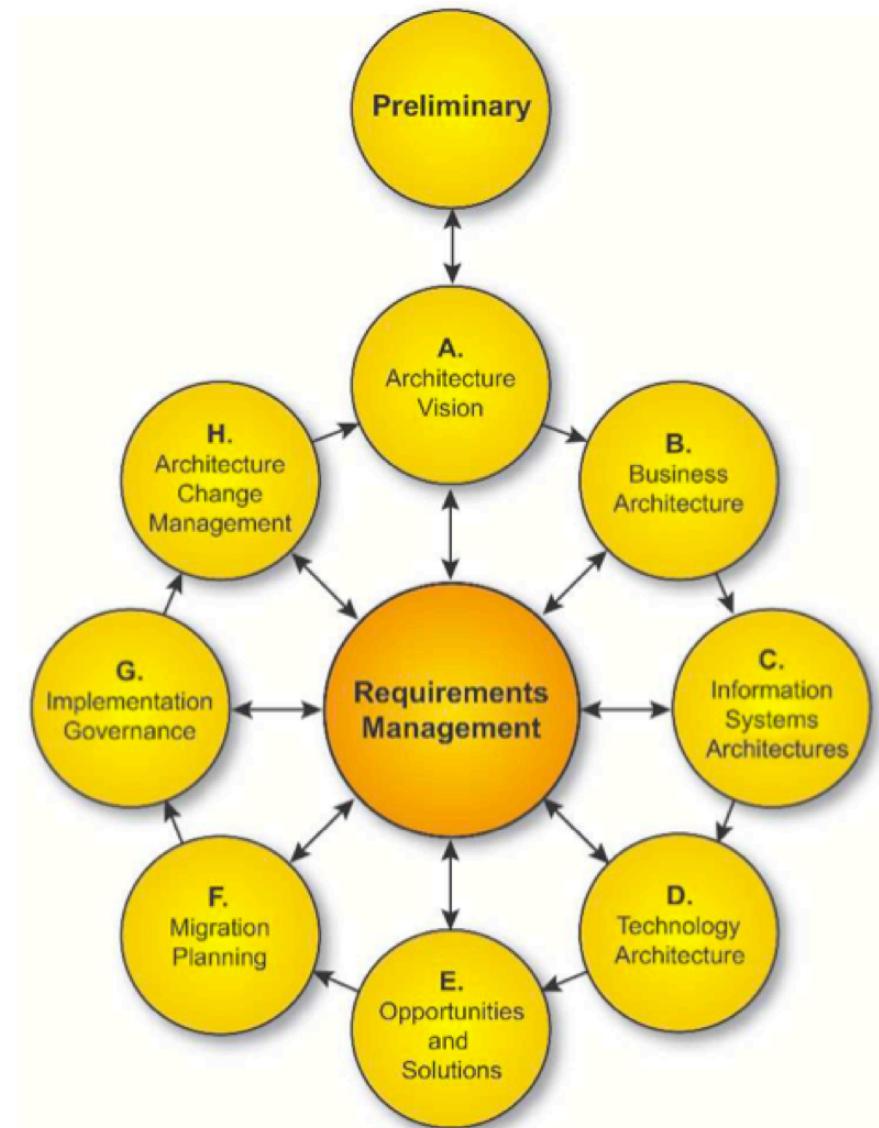


TOGAF

- The Open Group Architecture Framework (TOGAF) is a framework for enterprise architecture that provides an approach for designing, planning, implementing, and governing an enterprise information technology architecture.
- TOGAF is a high level approach to design. It is typically modeled at four levels: Business, Application, Data, and Technology.
- TOGAF Relies heavily on modularization, standardization, and already existing proven technologies and products
- TOGAF Components
 - Architecture Development Method (ADM)
 - Enterprise Continuum
 - Resource Base

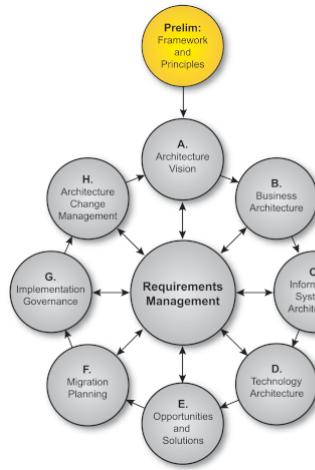
TOGAF – Architecture Development Method (ADM)

- A Step by Step process to Developing or Changing an Architecture
 - Preliminary } Defines the need for Architectural Change
 - Phase A }
 - Phase B }
 - Phase C }
 - Phase D }
 - Phase E How the Vision & Future Architecture is Delivered
 - Phase F Implementation & Migrations Planning
 - Phase G Architectural Oversight to Implementation
 - Phase H Architecture Change Management
 - Requirements Management

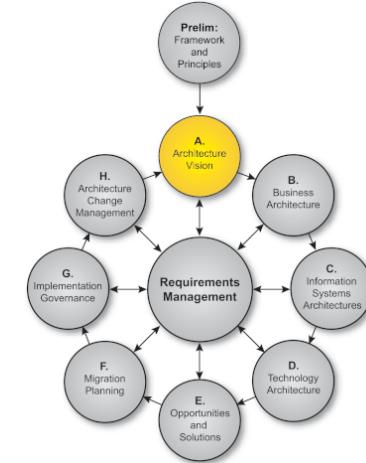


ADM – Preliminary Phase

- Defines what needs to be done, how it will be carried out.
- Establish parameters for a successful iteration of ADM
- Identify and establish architecture Frameworks & Principles
- TOGAF can be tailored to meet the needs
- TOGAF can be integrated with other management frameworks (e.g. PRINCE2)
- Output: Request for Architecture Work
 - Outlines Requirements, Organizational Context, Structures, Tools or Architecture Frameworks



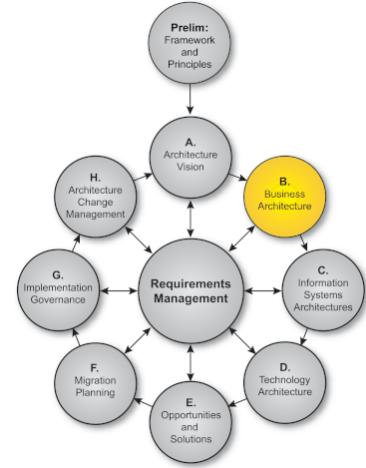
ADM – Phase A: Architecture Vision



- Starts with Request for Architecture Work
- Sells the benefits of the proposed capability to stakeholders and decision-makers
- Outlines Vision for the Architecture
 - High-Level aspiration of capabilities
 - Business values that the Architecture will deliver
- Identifies Concerns and Requirements.
- Confirms business goals, drivers and constraints
- Goal is to make sure that the enterprise is Able, Ready, Willing and Committed to make the necessary Architecture Changes
- Output: Statement of Architecture Work
 - It also provides the Vision of the proposed enterprise architecture. This sense of direction is vital for guiding the work throughout this iteration of the ADM

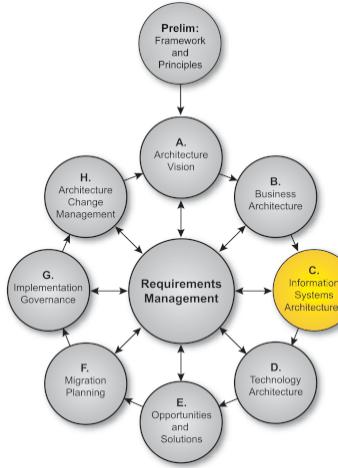
ADM – Phase B: Business Architecture

- Starts with Improving Business Capability
- Key objective is to identify Target Business Architecture that shows how the enterprise can achieve the Architecture Vision
- Business Architecture demonstrate Business Value and Return on Investment (ROI) to the Stakeholders
- Output:
 - Business models
 - Activity or Process models
 - Use Case



ADM – Phase C: Information Systems Architecture

- Takes the Business Perspective from previous Phase as the input
- Information Systems Architecture compose Data Architecture and Applications Architecture
- Data & Application Architecture uses different reference models
 - Data Architecture – Class Diagrams, ER Diagrams
 - Application Architecture – Application Communication, Component Diagram, etc...
- Identify Candidate Architecture Roadmap Components
- Output:
 - Architecture Definition Document



Exercise #1

- Business Case:
 - There are many types of Vehicles; Cars, Vans, Bikes, etc... Different types of vehicles may have specialized attributes but there are a few attributes in common.
 - A Person may own a vehicle for a given time and this ownership is registered at the Department of Motor Vehicles.
- Q1) How would you develop Data Architecture for the Above? State your assumptions.
- Q2) Draw the Data Architecture diagrams.

Exercise #1: Sample Answers...

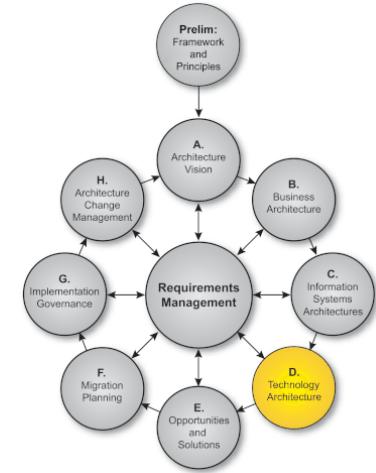
- Developing Data Architecture
 - Requirement Clarification meetings to be held to better understand the requirement. Use Case Diagrams to be created
 - A Gap Analysis on the requirement & business entities needs to be created.
 - Business and Logical Data Models to be identified. Create Diagrams i.e. Class Diagrams / Entity Relationship Diagrams to be created.
- Assumptions
 - A vehicle cannot be a Car or a Van at the same time [Total Participation]
 - Vehicle will be owned by only 1 person at a given time [Person : Vehicle = 1 : M]
 - A vehicle ownership can be transferred from one person to another [Association Vs. Composition]
 - Vehicle & Person attributes can represented in Text, Number and Date formats
- Diagrams
 - ER Diagram
 - Class Diagram
 - Table Structures

Exercise #2:

- Business Case:
 - A CCTV Camera system captures a set of videos and send them to a central Server. The server needs to store the video and meta information for later retrieval.
- Q1) Create the Data Architecture
 - Data Architecture diagrams.
- Q2) Create the Application Architecture
 - Application Communication
 - Component Diagram

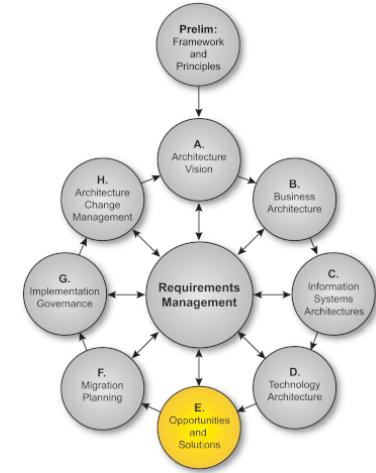
ADM – Phase D: Technology Architecture

- Technology Architecture is a description of...
 - Structure and interaction of the platform services
 - Logical and Physical technology components.
- Develop Baseline Technology Architecture
 - Creates Technology Reference Models & Criteria for Measurement
 - Develop Target Technology Architecture - requirements traceability, criteria for selection of service portfolio
- Output:
 - Baseline Technology Architecture
 - Networked Computing/Hardware view
 - Communications view
 - Processing view
 - Technology Architecture Report (summarizing the key findings)



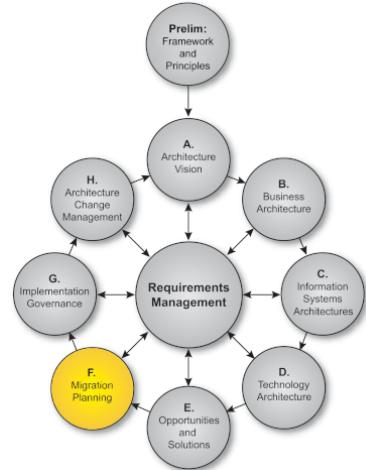
ADM – Phase E: Opportunities and Solutions

- About finding Opportunities for...
 - Delivering the Target Architecture by implementing specific Solutions.
 - Concentrated on How to Deliver the Architecture
 - When the change is large, this Phase provides an Incremental Approach to convert from Baseline to Target Architecture
- Generates the first complete version of Architecture Roadmap by combining the analysis and suggestions from the Architecture Development phases
- Output:
 - High-level Implementation Plan
 - High-level Migration Plan & Impact Analysis



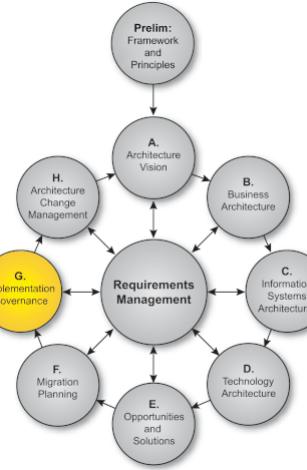
ADM – Phase F: Migration Plan

- Finalizes a detailed Implementation and Migration Plan
 - Also finalizes the Architecture Roadmap
- Plan is coordinated with...
 - Change management approach used within the enterprise
 - Business Planning
 - Enterprise Architecture
 - Portfolio and Project Management
 - Operations Management
- Goal is to ensures that key stakeholders fully understand
 - Business value
 - Cost of work packages
 - Transition and Future Architectures



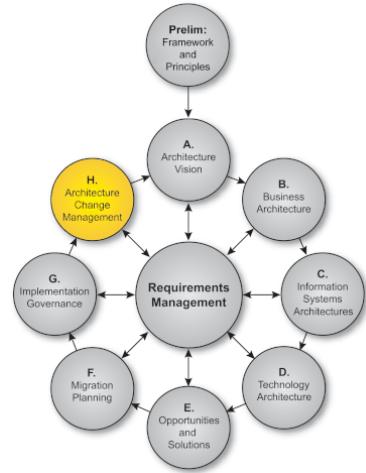
ADM – Phase G: Implementation Governance

- Provides an Architectural Oversight to Implementation
 - Ensures Project Implementation conforms the Target Architecture
- Formulate Project Recommendations
- Manages Implementation Driven Architecture Changes
- Review Ongoing Implementation Governance and Architecture Compliance
 - Confirms the scope and priorities for deployment
 - Guiding development and solutions deployment
 - Performs compliance reviews
- Output:
 - Architecture Contract Document (drives any Architecture Changes)

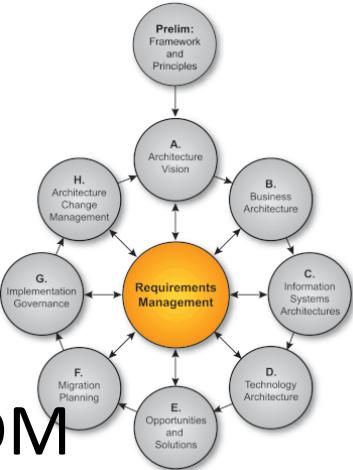


ADM – Phase H: Architecture Change Management

- Change management process to manage changes to the Architecture
 - Process for managing changes
 - Ensure Architecture achieves its intended Business Value
- Requires continues monitoring
 - Governance Requests
 - New Technologies
 - Changes in the Business Environment
 - Strategic Changes (e.g. Cost)
- Judge whether a Change Request warrants a simple architecture update or whether it requires to Re-Architect with ADM
- Output:
 - Architecture updates
 - Changes to architecture framework and principles
 - New Request for Architecture Work



ADM – Requirement Management



- It's a Continuing Ongoing Process and sits in the center of the ADM
 - Requirements are Produced, Analyzed and Reviewed in each ADM Phase
- Ensured Changes to Requirement are well governed and Reflected in all other Phases
- Describes a Process for Requirements Management and how they are lined to the other Phases
- Output:
 - Changed requirements
 - Requirements Impact Statement

Zachman Framework

- Zachman Framework is an enterprise ontology and is a fundamental structure for Enterprise Architecture which provides a formal and structured way of viewing and defining an enterprise.
- The ontology is a two dimensional classification schema that reflects the intersection between two historical classifications.
 - Dimension #1: What, How, Where, Who, When and Why
 - Dimension #2: Contextual, Conceptual, Logical, Physical, As Built and Functioning Enterprise

Zachman Framework – Rows

Row 1 – Scope

External Requirements and Drivers
Business Function Modeling

Row 2 – Enterprise Model

Business Process Models

Row 3 – System Model

Logical Models
Requirements Definition

Row 4 – Technology Model

Physical Models
Solution Definition and Development

Row 5 – As Built

As Built
Deployment

Row 6 – Functioning Enterprise

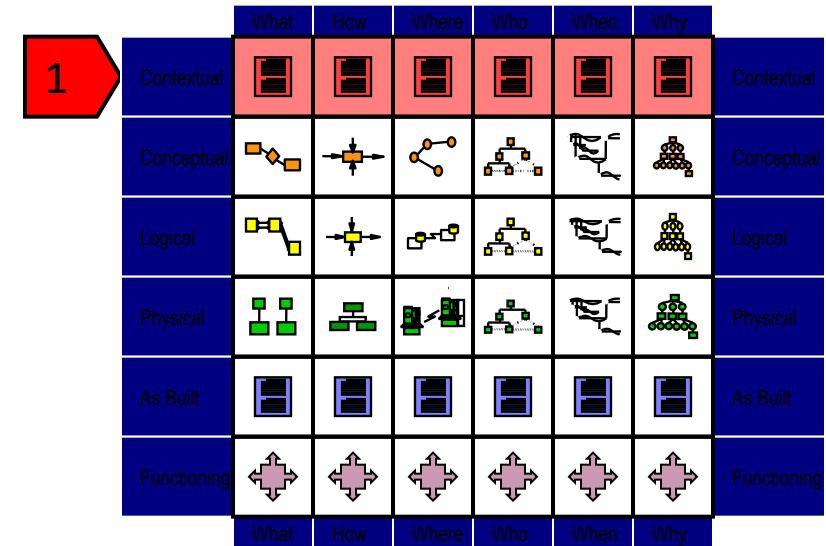
Evaluation



	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning

Zachman – Row 1: Scope (Planner's View)

- Motivation/Why
 - Business goals, objectives and performance measures
- Function/How
 - High-level business functions
- Data/What
 - High-level data classes related to each function
- People/Who
 - Stakeholders related to each function
- Network/Where
 - Locations related to each function
- Time/When
 - Cycles and events related to each function



Zachman – Row 2: Enterprise Model (Designer's View)

- Motivation/Why
 - Policies, procedures and standards for each process
- Function/How
 - Business processes
- Data/What
 - Business data
- People/Who
 - Roles and responsibilities in each process
- Network/Where
 - Locations related to each process
- Time/When
 - Events for each process
 - Sequencing of integration
 - Process improvements



	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning
	What	How	Where	Who	When	Why	

Zachman – Row 3: System Model (Designer's View)

- Motivation/Why
 - Policies, standards and procedures associated with a business rule model
- Function/How
 - Logical representation of information systems and their relationships
- Data/What
 - Logical data models of data
 - Data relationships
- People/Who
 - Logical representation of access privileges
- Network/Where
 - Logical representation of the distributed architecture
- Time/When
 - Logical events and their triggered responses



	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning
	What	How	Where	Who	When	Why	

Zachman – Row 4: Technology Model (Builder's View)

- Motivation/Why
 - Business rules constrained by information systems standards
- Function/How
 - Specifications of applications that operate on particular technology platforms
- Data/What
 - Database management system
 - Logical data models
- People/Who
 - Access privileges to technologies
- Network/Where
 - Network devices and their relationships
- Time/When
 - Specification of triggers to respond to system



	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning
	What	How	Where	Who	When	Why	

Zachman – Row 5: As Built (Integrator's View)

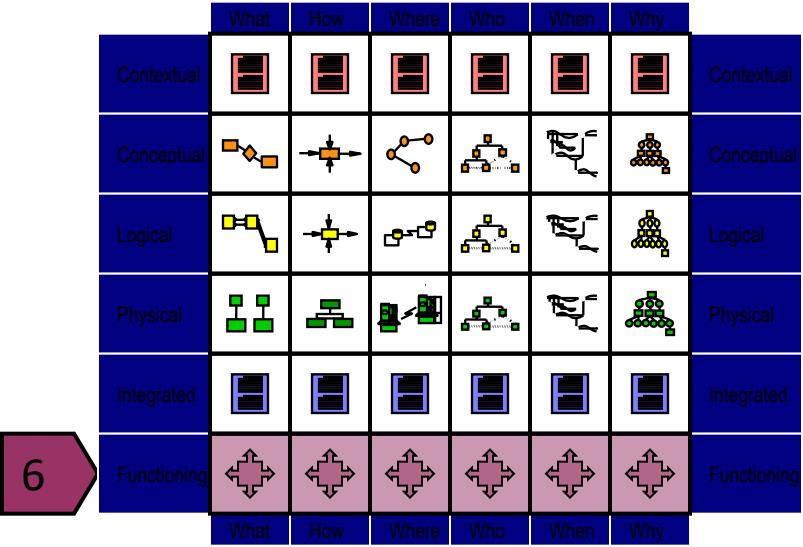
- Motivation/Why
 - Business rules constrained by specific technology standards
- Function/How
 - Programs coded to operate on specific technology platforms
- Data/What
 - Data definitions by physical data models
- People/Who
 - Access privileges to control access
- Network/Where
 - Devices configured to conform to node specifications
- Time/When
 - Timing definitions coded to sequence activities



	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning
	What	How	Where	Who	When	Why	

Zachman – Row 6: Functioning Enterprise (User's View)

- Motivation/Why
 - Operating characteristics of specific technologies constrained by standards
- Function/How
 - Functioning computer instructions
- Data/What
 - Data values stored in actual databases
- People/Who
 - Personnel and key stakeholders / roles
- Network/Where
 - Sending and receiving messages
- Time/When
 - Timing definitions operating to sequence activities



Zachman Framework – Outputs

ENTERPRISE	What	Where	Why	How	Who	When
Conceptual	Entity Relations	Node Relations	Goal Relations	Process Relations	Persona Relations	Event Relations
Contextual	Entity Associations	Node Associations	Goal Associations	Process Associations	Persona Associations	Event Associations
Logical	Entity Attributes	Node Attributes	Goal Attributes	Process Attributes	Persona Attributes	Event Attributes
Physical	Entity Domains	Node Domains	Goal Domains	Process Domains	Persona Domains	Event Domains
Mechanical	Entity Definitions	Node Definitions	Goal Definitions	Process Definitions	Persona Definitions	Event Definitions
Instantial	Entities	Nodes	Goals	Processes	Personas	Events

References

- <http://pubs.opengroup.org/architecture/togaf8-doc/arch/toc.html>
- <https://www.orbussoftware.com/enterprise-architecture/togaf/what-is-the-adm/>
- <https://www.zachman.com/>