NANYANG TECHNOLOGICAL UNIVERSITY

CZ3002 - Advanced Software Engineering

Software Maintenance:

Designing For Maintainability

Faculty : Dr Althea Liang

School : School of Computer Science and Engineering

Email : qhliang@ntu.edu.sg

Office : N4-02c-107

Introduction to this Lesson

At the end of the lesson, you should be able to:

- Outline **design for maintainability** or **design ideas for change (why and how)**
- Understand the design idea for change **in form of design patterns**
  - Definition and purposes of **design patterns**
  - Levels and kinds of design patterns: Architectural patterns and mid-level design patterns
  - Two design patterns: **Model-View-Controller (MVC)** and **service-oriented design**.

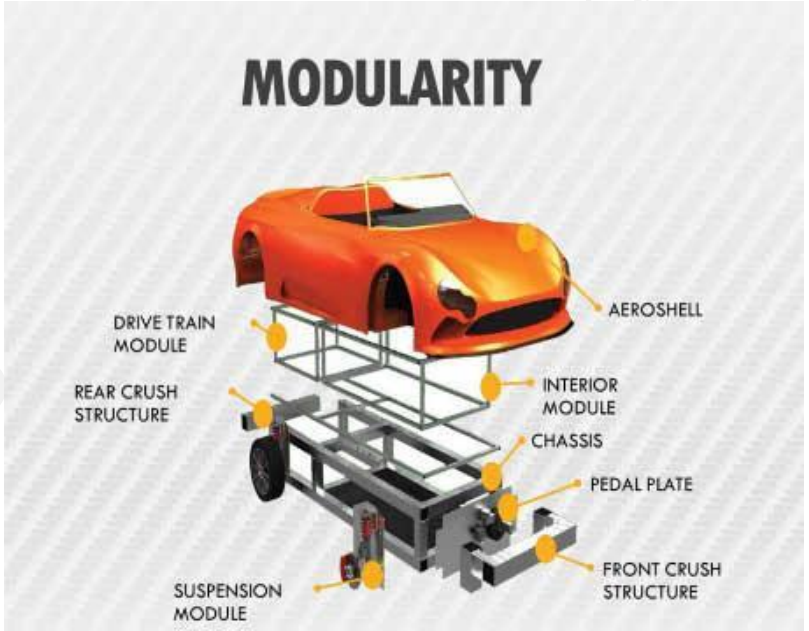
Introduction to this Lesson: Examples Preview

Example	Slides
Str	Struts MVC Framework Struts 2 MVC Framework
WC	Wiring-by-Configuration
JR	Java Reflection – Elegance!

Design Ideas for Change

Applying law of maintenance to overall goal of design, then to ideas of design

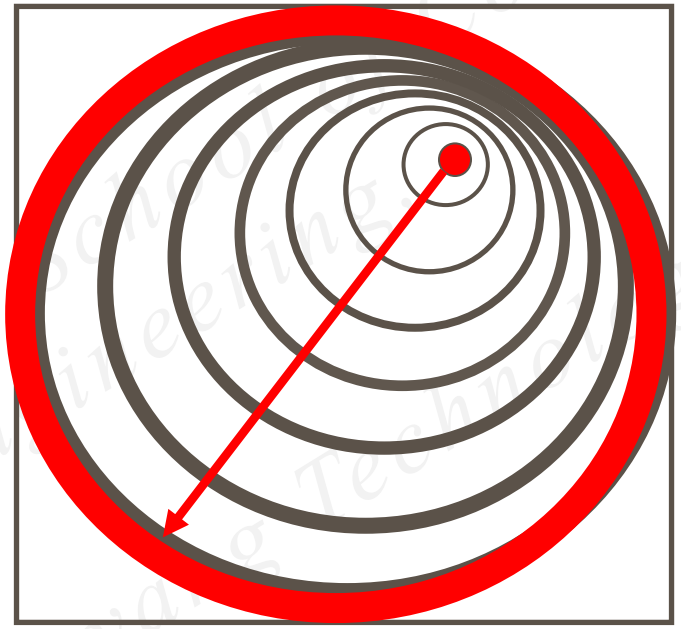
- Software industry laws: Second law of **Lehman's laws**
  - Change->complexity->takes resources (more details to be discussed in next lection)
- Idea 1: To create separate modules and with a **good modularity**
  - Well-defined
  - Conceptually simple
  - Independent upon partitioning
    - low **coupling**
    - high **cohesion**
  - With well-defined interfaces



More Design Ideas for Change

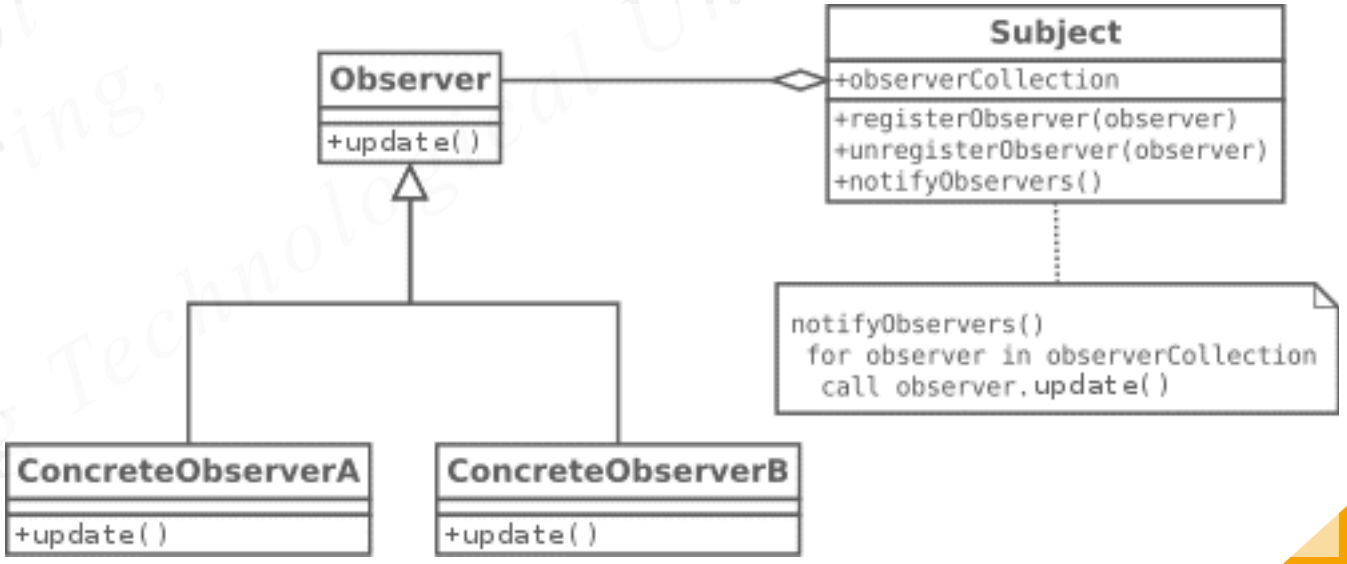
Applying law of maintenance to overall goal of design, then to ideas of design

- Idea 2: Via decisions ensuring encapsulation
- Idea 3: Patterns
- Indications of bad Ideas?



Patterns in Software !

- A pattern is a **model** proposed for **imitation** for solving a **software design problem**
- Used at different **levels** of abstraction of **design problems**
  - Architectural Patterns for entire systems
  - Design Patterns for collaborations between several classes
  - Data Structures and Algorithms
  - Programming Idioms



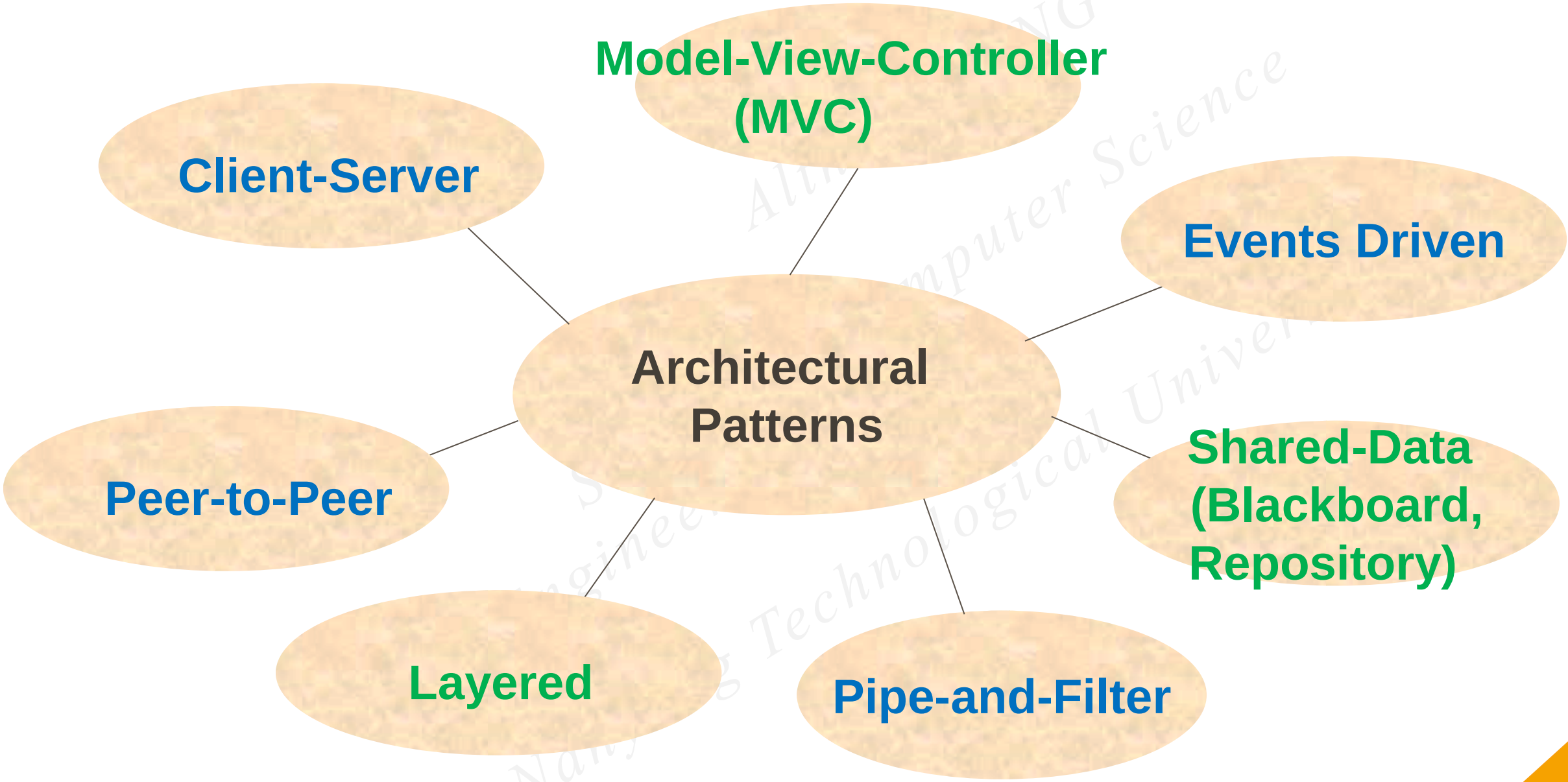
Defining Design Patterns

- A design pattern is a way of reusing abstract knowledge about a problem and its solution.
- A pattern is a **description of the problem** and the **essence of its solution**.
- It should be sufficiently abstract to be reused in different settings.
- Patterns often rely on object characteristics such as inheritance and polymorphism.

Pattern Elements

- Name
  - A meaningful pattern identifier
- Problem description
- Solution description
  - Not a concrete design but a template for a design solution that can be instantiated in different ways
- Consequences
  - The results and trade-offs of applying the pattern

Architectural patterns and two types!



Mid-Level Design Patterns

Broker Design Patterns

- Façade, Mediator
- Adaptor
- Proxy

Generator Design Patterns

- Factory
- Singleton
- Prototype

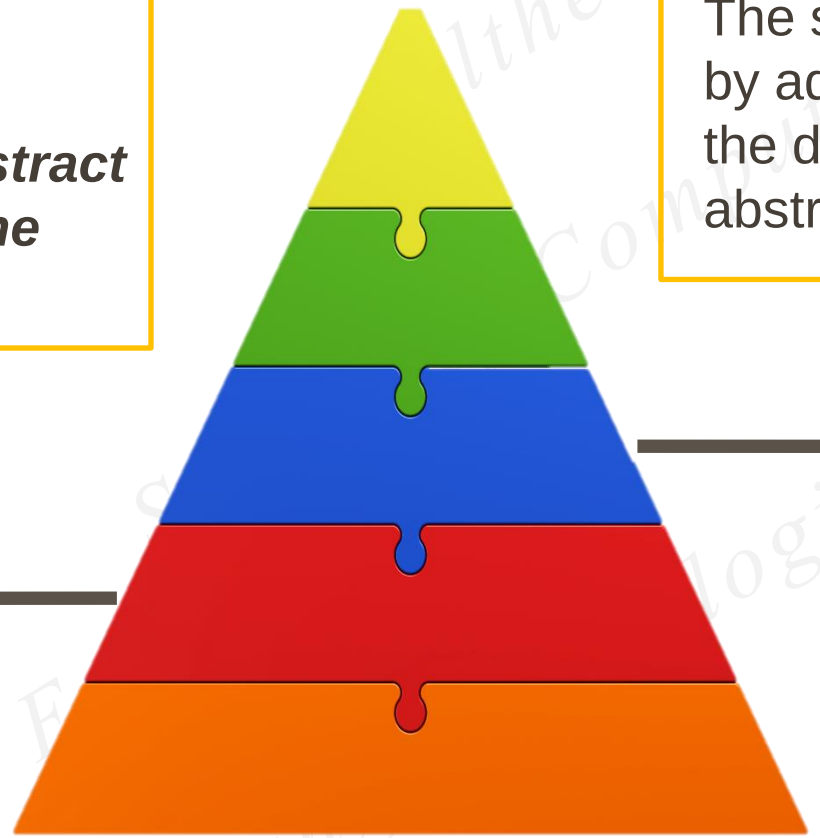
Reactor

- Command
- Observer

Application Frameworks

Frameworks are a **sub-system *design*** with implementation specifics

Made up of a collection of ***abstract and concrete classes and the interfaces*** between them.



The sub-system can be implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework.

Model-View-Controller (MVC)

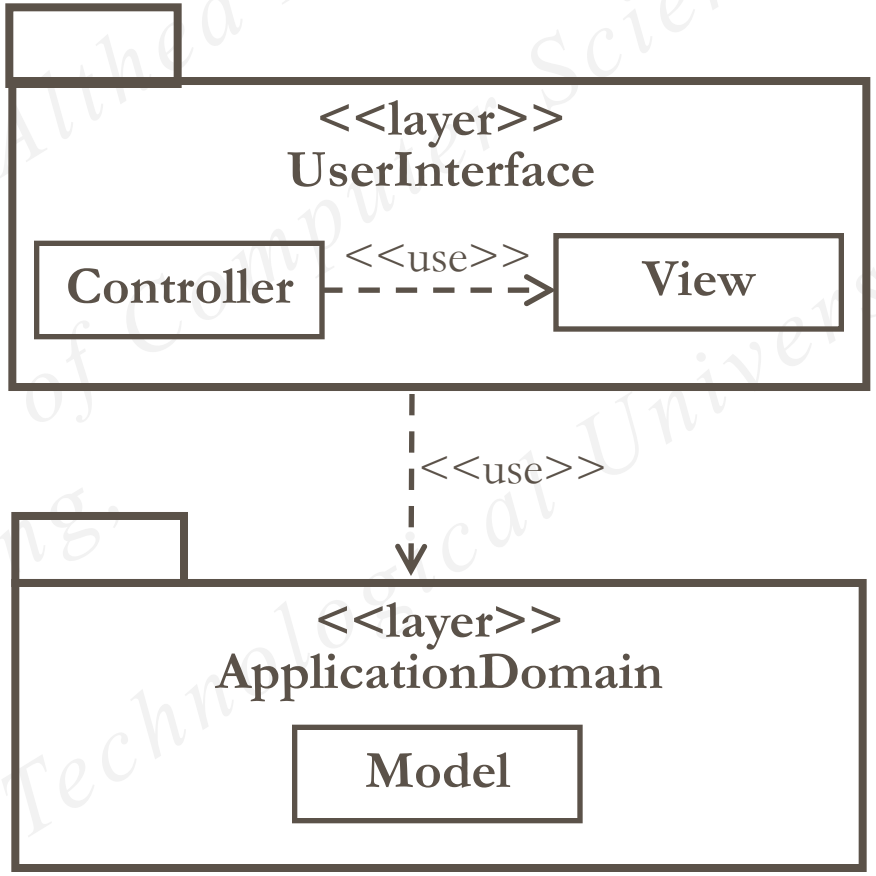
- System infrastructure framework for Graphic User Interface (GUI) design.
- MVC frameworks involves the instantiation of a number of patterns (as discussed earlier under concept reuse).
- Allows for multiple presentations of an object and separate interactions with these presentations.

MVC Static Structure

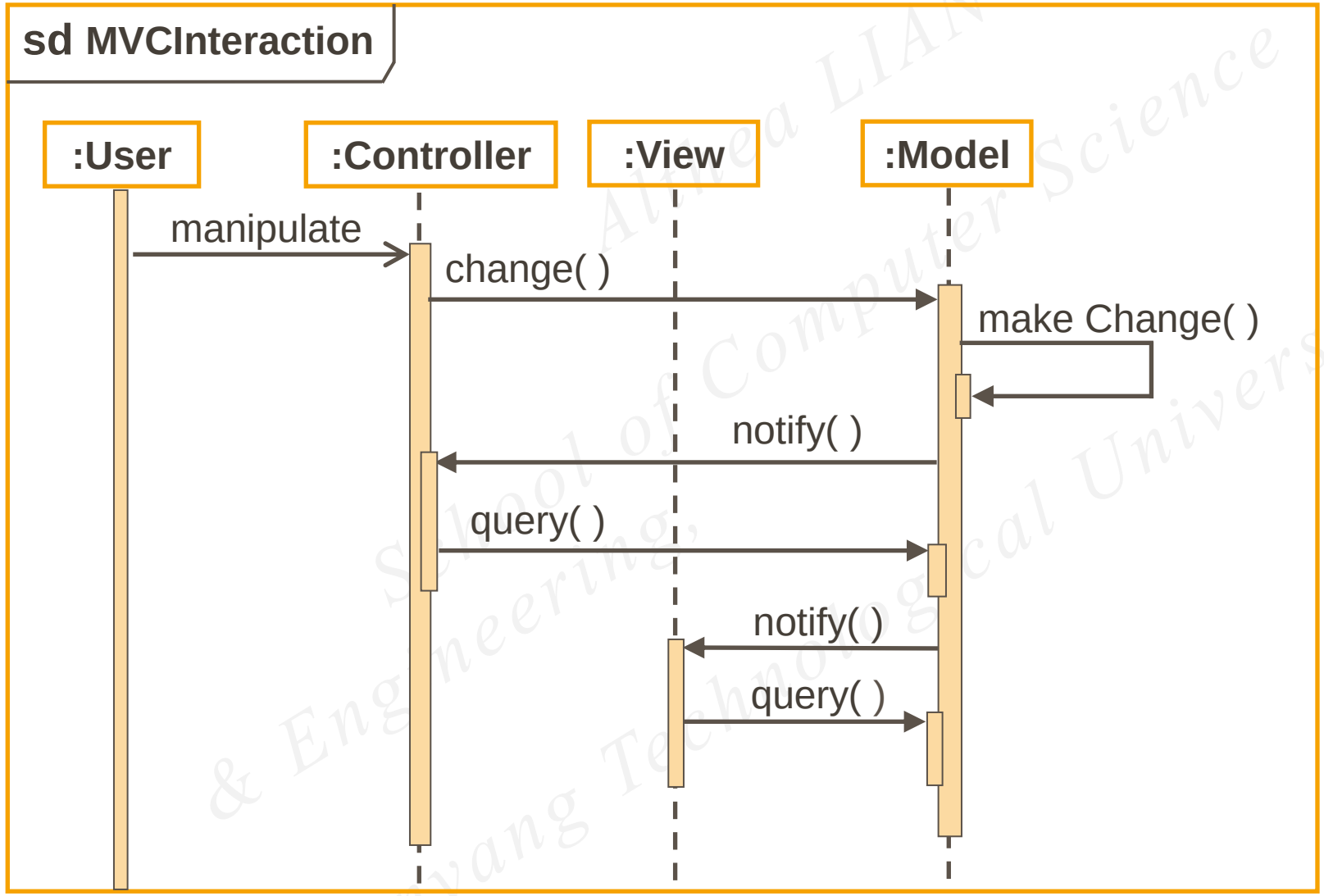
Model – holds data and operations for achieving computation goals of application

View – components that display data from Model to users

Controller – components that receive and carry out commands from users, to alter the View or Model



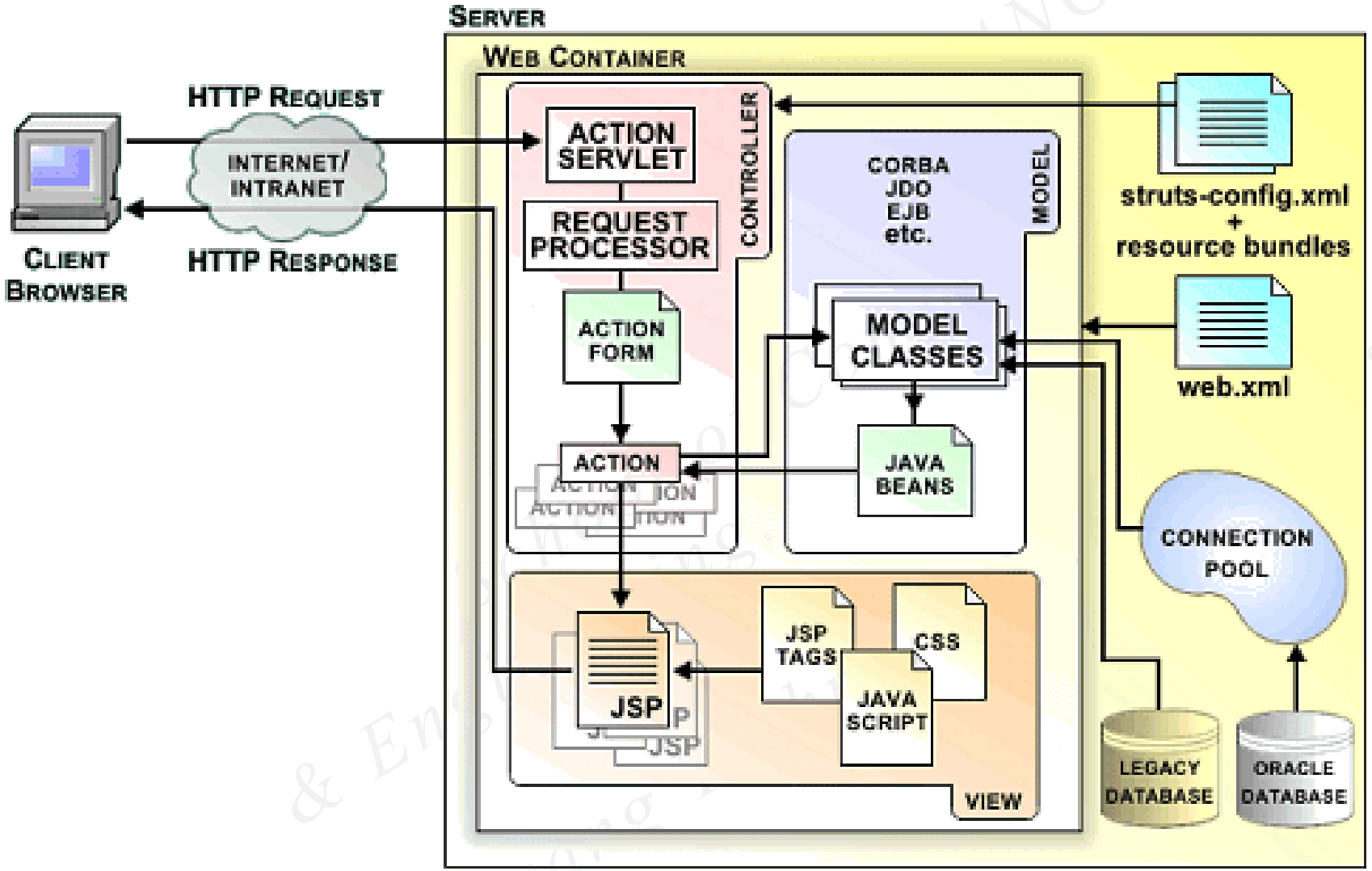
MVC Dynamic Behavior



Using Frameworks by Extending them

- Frameworks are considered generic and are extended to create a more specific application or sub-system.
- Extending the framework involves:
  - Adding concrete classes that inherit operations from abstract classes in the framework.
  - Adding methods that are called in response to events that are recognised by the framework.
- Problem with frameworks is their complexity which means that it takes a long time to use them effectively.

Struts MVC Framework – Strutrs 1 (Example Str)

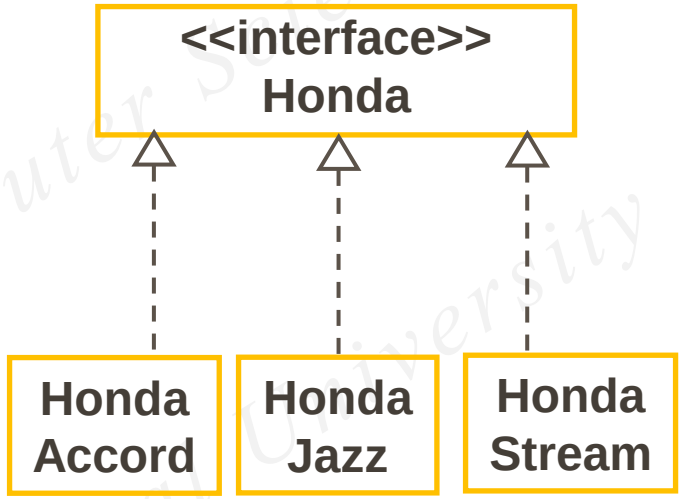


Wiring-by-Configuration (Example WC) by Java Reflection

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
   <!DOCTYPE struts-config PUBLIC
         "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
         "http://struts.apache.org/dtds/struts-config_1_3.dtd">
   <struts-config>
      <form-beans>
         <form-bean name="logonForm" type="app.LogonForm"/>
      </form-beans>
      <action-mappings>
         <action path="/Welcome" forward="/pages/Welcome.jsp"/>
         <action path="/Logon" forward="/pages/Logon.jsp"/>
         <action path="/LogonSubmit" type="app.LogonAction" name="logonForm"
            scope="request" validate="true" input="/pages/Logon.jsp">
            <forward name="success" path="/pages/Welcome.jsp"/>
            <forward name="failure" path="/pages/Logon.jsp"/>
         </action>
         <action path="/Logoff" type="app.LogoffAction">
            <forward name="success" path="/pages/Logoff.jsp"/>
         </action>
      </action-mappings>
      <message-resources parameter="resources.application"/>
   </struts-config>
```

Java Reflection – Elegance! (Example JR)

```
if (param.equals("HondaAccord"))
    return new HondaAccord();
else if
    (param.equals("HondaStream"))
    return new HondaStream();
else if (param.equals("HondaCRV"))
    return new HondaCRV();
else
    return new HondaJazz();
```




```
return (Honda)Class.forName(param).newInstance()
```

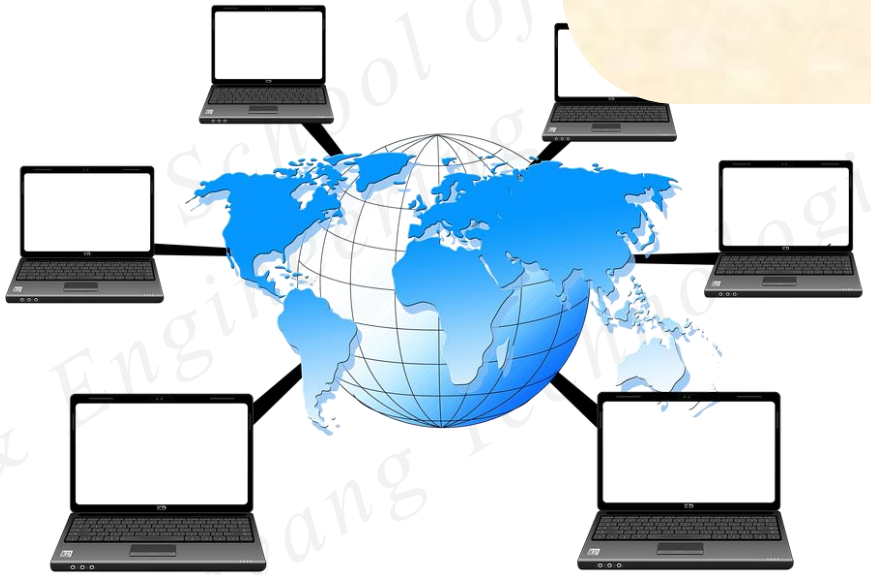
Software Services and Web Services

Service

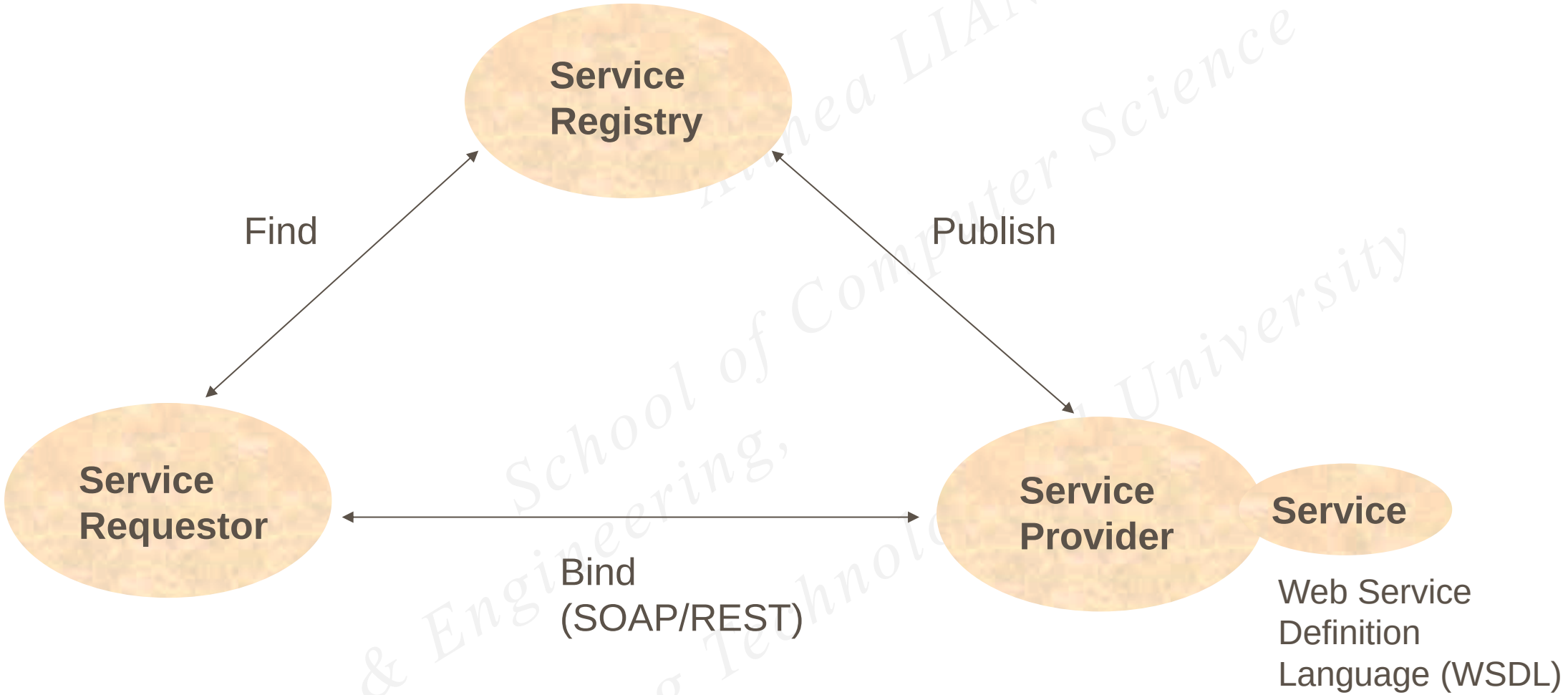
- A **loosely-coupled**, **reusable** software component that encapsulates discrete functionality.

Web Service

- A service that is **distributed** and **programmatically accessed**.
- A service that is accessed using standard Internet and **XML-based** protocols.



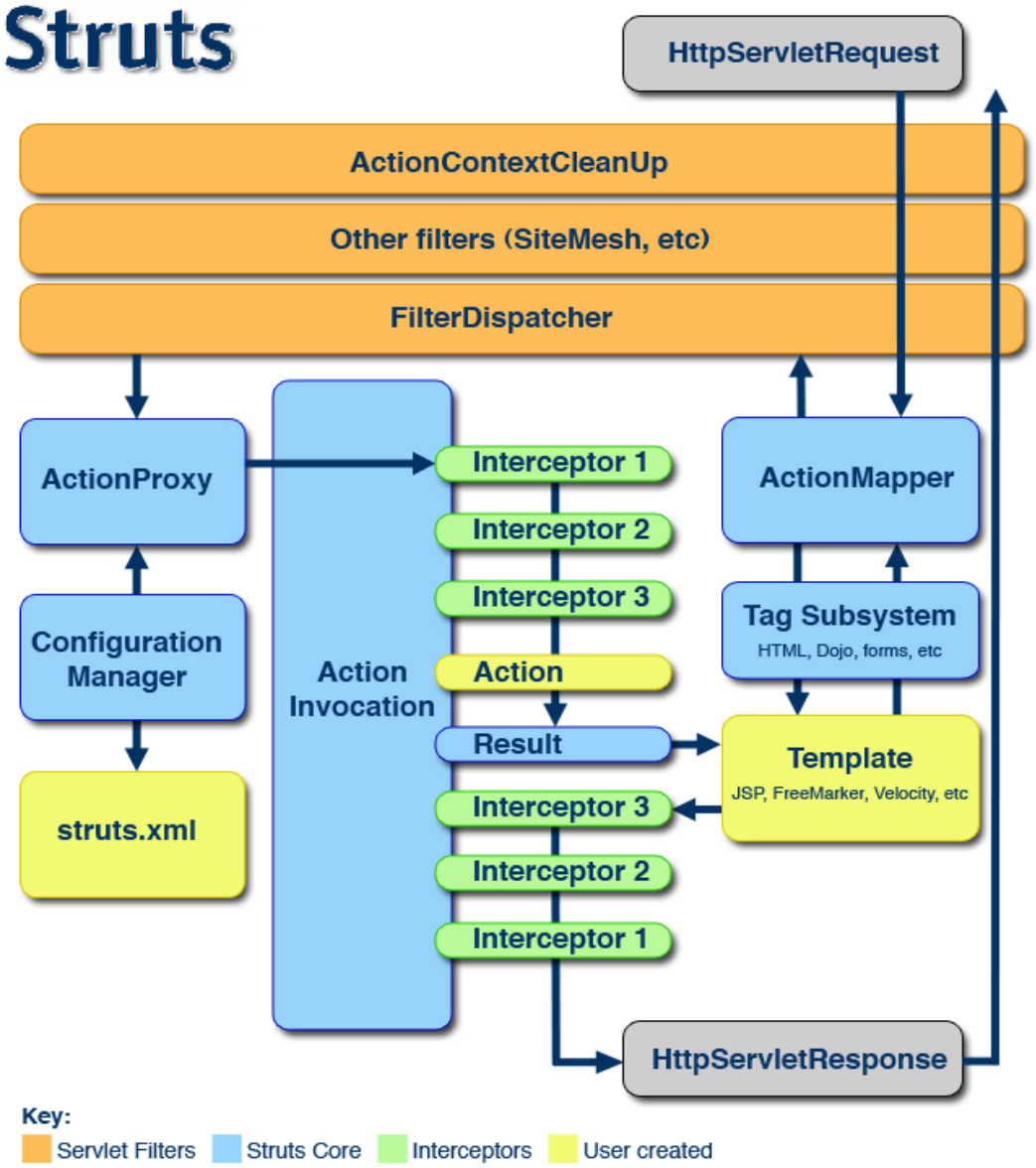
Web Services Implementing Service Oriented Architecture (SOA)



Service-Orientation Design Principles

Well-defined interfaces	A well specified description of the service that permits consumers to invoke it in a standard way
Loose coupling	Service consumer is independent of the implementation specifics of service provider
Logical and physical separation of business logic from presentation logic	Service functionality is independent of user interface aspects
Highly reusable services	Services are designed in such a way that they are consumable by multiple applications
Coarse-grained granularity	Services are business-centric, i.e., reflect a meaningful business service not implementation internals

Struts 2 MVC Framework (Example Str)



https://struts.apache.org/core-developers/big-picture

Summary of Technologies:

- Java
- Filters, JSP, and Tag Libraries
- JavaBeans
- HTML and HTTP
- Web Containers (such as Tomcat)
- XML

Summary

Now, you should understand:

- Design ideas for change
- Purposes of design patterns
- Different architectural patterns and mid-level design patterns
- Model-View-Controller (MVC) Application Framework
- Service-oriented design and their benefits