

# Java Architecture using Oracle 10G Application Server

James Yao and Andrew Okimi  
Apr 30, 2006

# Table of Contents

<b>1</b>	<b>PURPOSE</b>	<b>2</b>
<b>2</b>	<b>ARCHITECTURAL VIEW</b>	<b>2</b>
2.1	Architectural Layers	2
2.1.1	User Interface Layer (UI Layer)	3
2.1.2	Business Layer	3
2.1.3	Service Layer	4
2.2	High Level Architectural Design	5
2.3	Deployment View	6
2.3.1	Production	6
2.3.2	Development	7
<b>3</b>	<b>ENVIRONMENT</b>	<b>7</b>
<b>4</b>	<b>ARCHITECTURE STYLE</b>	<b>8</b>
4.1	Application Server	8
4.1.1	What is an Application Server? What does it provide?	8
4.1.2	The benefit of an application server	8
4.1.3	Use the application server smartly	10
4.2	Interface Driven Design	13
4.2.1	What's interface driven design?	13
4.2.2	The benefit of interface driven design	13
4.2.3	Recommended pattern: Inversion of Control	13
4.2.4	Recommended pattern: Factory / Abstract Factory	14
4.3	Spring Framework	14
4.3.1	What's Spring Framework?	14
4.3.2	The benefit from its features	14
4.3.3	Guideline of Spring Framework	15
4.3.4	Spring Framework Examples	15
<b>5</b>	<b>GUIDELINES AND REFERENCE</b>	<b>16</b>

# 1 Purpose

This document describes a Java architecture based on Oracle 10G Application Server. It also provides the architecture styles and the design guidelines to follow.

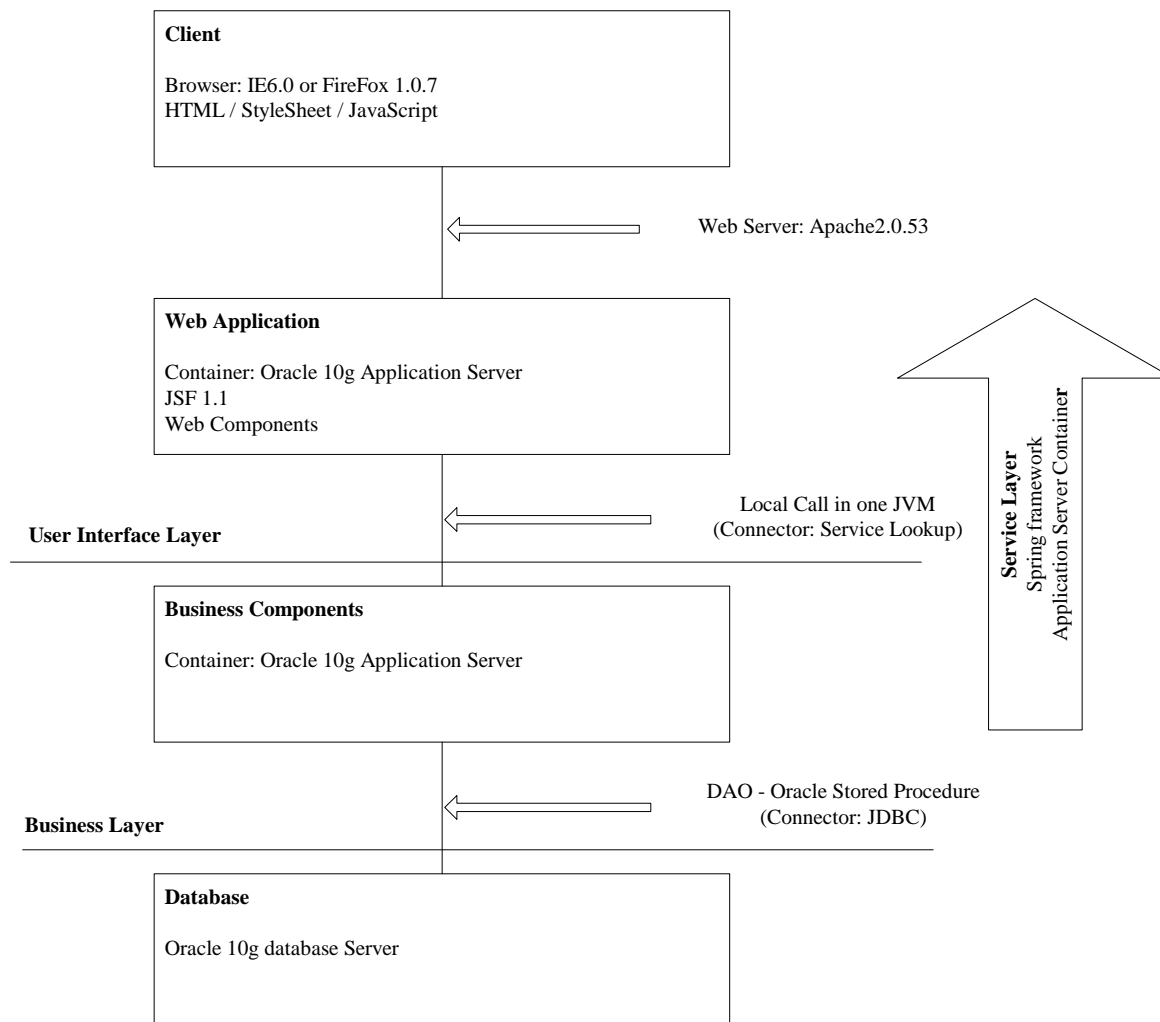
## 2 Architectural View

### 2.1 Architectural Layers

The system can be divided into three layers.

- The user interface layer will handle web operations. The user uses the browser to interact with the web application.
- The business layer will handle business logic and data / models.
- The service layer is the infrastructure that provides services and the application framework.

The relationship of these layers can be illustrated as follows:



### 2.1.1 User Interface Layer (UI Layer)

Client	Browser	IE 6.0 and Firefox 1.0.7 should be supported.
	Style Sheet	Style sheet is mandatory for each web page.
	AJAX	AJAX is an option – AJAX and JavaScript will be used where it makes business sense to do so.
Web Server	Apache 2.0.46	Provided by Oracle AS 10.1.3
	Web Container	Oracle 10g Application Server
Web Component	JSF 1.1	JSF is mandatory for the development of dynamic web pages.
	Struts 1.2	Struts 1.2 is an option as a controller if necessary
	Controller	Flow controller and action controller are independent components to develop. They should be decoupled from JSF and Struts.
Design Strategies	Look and Feel	The look and feel (including layout and color schema) of the web page will follow the prototype.

### 2.1.2 Business Layer

Model	Pure Value (Data) Object	Except accessor methods, no logic should be put into value objects. If the logic has to be embedded, the value bean should be designed as a component. For example it can be an instance of two interfaces: one interface spec defines the accessor and other interface spec defines the logic.
Patterns	Service locator	The service-locator component provides any service/component from any other layer.
Data Access	DAO	Data access object component will use Oracle JDBC to access stored procedures.
	Stored Procedure with JDBC	Stored procedure will behave as backend data access point.
	Hibernate	Using Hibernate to simplify coding and mapping is recommended when applied. However, the backend query should be Oracle Stored Procedures.
Component Container	Spring Framework	Spring Framework is mandatory even when an application server is utilized.
	Application Server Container	Application server container provides support services, e.g. cache, JMS, JMX, MDB, stateless SB, etc. Stateful SB is typically not allowed, unless very specific circumstances warrant their use.

### 2.1.3 Service Layer

Logging	A log component will be provided in the component container. Direct usage of Log4j is allowed but not recommended.
Data Source	A DS will be provided in the application server container. If OAS is not utilized, the component container will provide one.
Security	A security context will be provided in the component container and the application server container.
Transaction	We encourage one operation with one read-committed transaction. It is recommended to work with the stored procedures (data access components). If a user transaction is required, a JTA support is the solution provided by application server container.
Other J2EE Service	Except Entity Bean and Stateful Session Bean, most of J2EE services are allowed to use.
Free Components	Any open-source components provided by Spring, JBoss and Jakarta are allowed to use. But if it is not interface based, a proxy interface spec is required to wrap such component before usage.

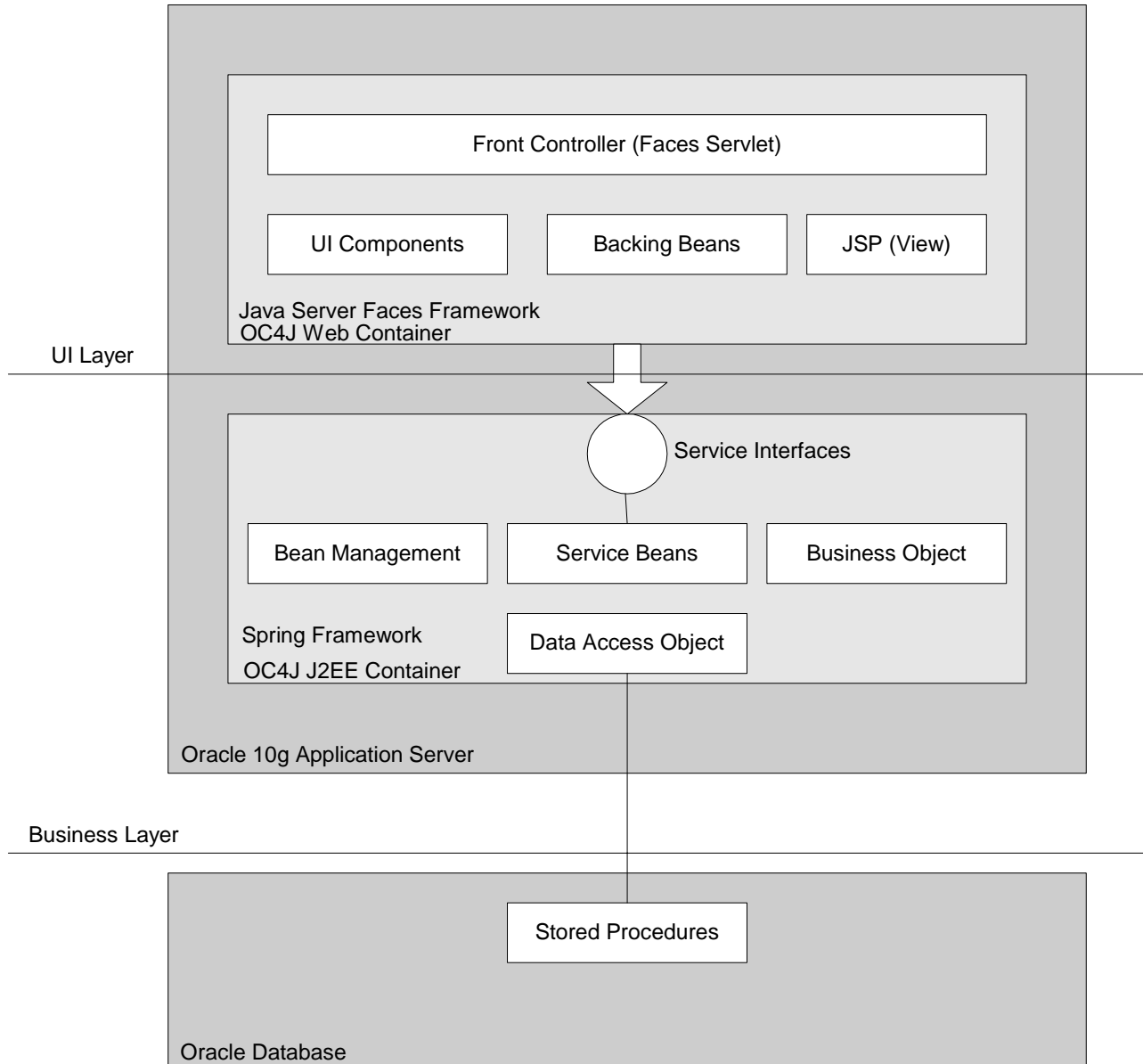
## 2.2 High Level Architectural Design

At UI Layer, JSF framework is applied.

At Business Layer, Spring Framework is applied to work with UI Layer.

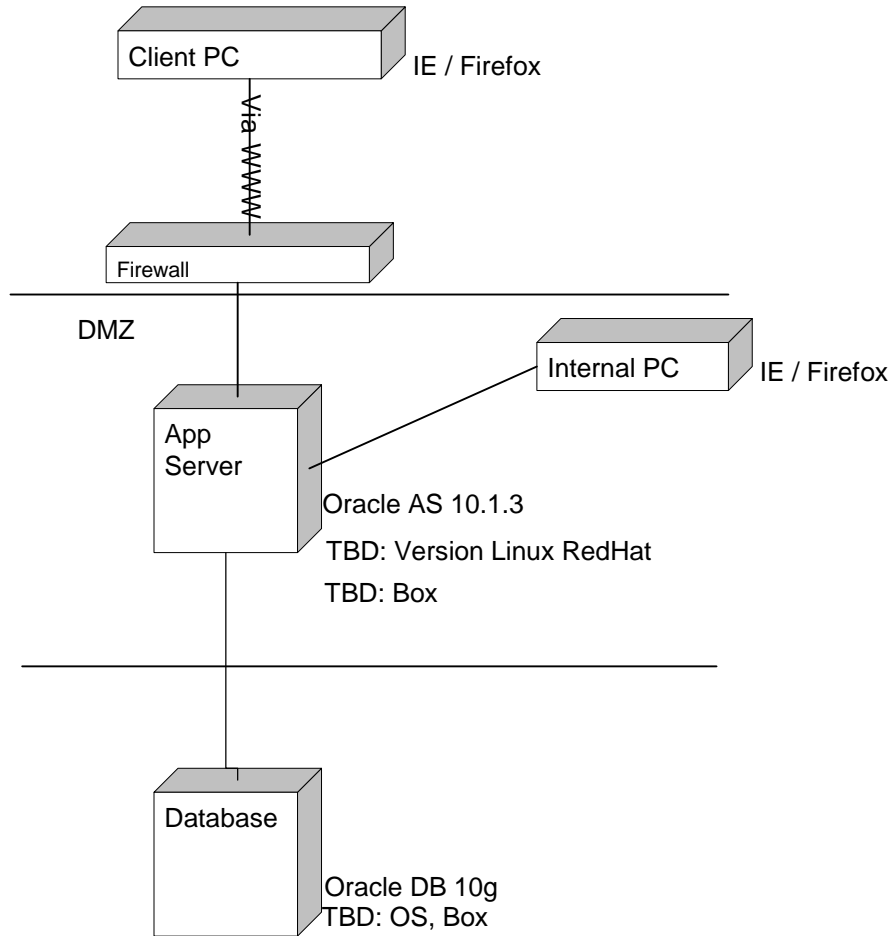
At Integration of Business Layer, DAO is applied to work with Stored Procedures at the RDBMS.

The technical architectural design can be illustrated as the following:

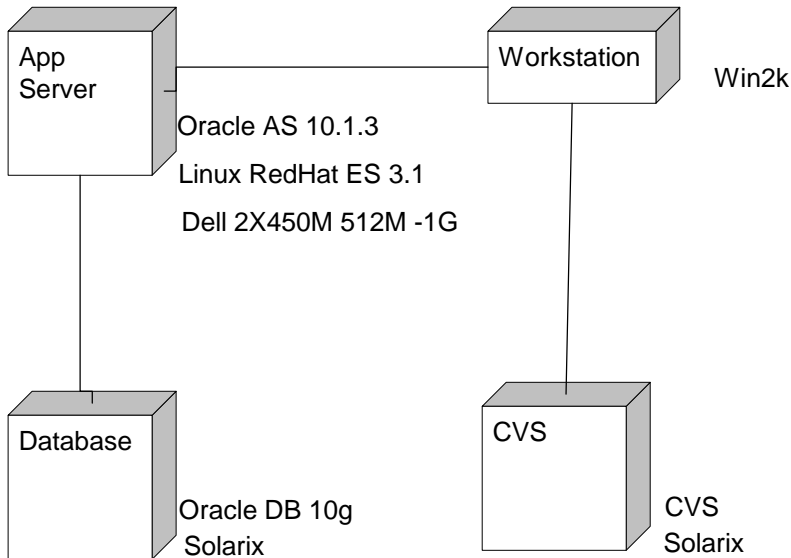


## 2.3 Deployment View

### 2.3.1 Production



### 2.3.2 Development



## 3 Environment

Language	Java1.5.0	
	J2EE 1.4	JSF1.1; Servlet 2.4/JSP2.0; JDBC2.0
IDE	JDeveloper 10.1.3	It is recommended to developers.
Web Designer	JDeveloper 10.1.3	
Testing	Junit4	
	Jmock1.0	
	Jmeter2.1	
Configuration Management	CVS	
Build	Ant 1.6.5	
Tracking	Perfect Track	
DB	Oracle DB 10g	
Application Server and Web Container	Oracle AS 10.1.3	

## 4 Architecture Style

### 4.1 Application Server

#### 4.1.1 What is an Application Server? What does it provide?

A J2EE application server is the server container following the J2EE specification. Application server provides many of the basic services required by any application.

- J2EE Services provided by an application server include:
  - Web container: JSP, Servlet, JSF and more
  - DataSource: A configurable-pooled data source
  - JMS: Java messaging and event notification service
  - JTA: Java transaction
  - Jmail: Java mail service
  - MDB/Stateless session bean: J2EE Components
  - Others: JMX, JSP/Servlet, Web Service, XML API services, etc.
  
- Other services provided by an application server
  - Administration: Start/Stop/Monitor/Configure application
  - Logging Service
  - Performance Tuning

#### 4.1.2 The benefit of an application server

- Example: Asynchronous vs. JMS

	Self-Developed Event-Notification	J2EE JMS implementation
Development	It requires the design and the development.	No development is required.
Usage	The developer has to understand the design and read the document to use it. It may vary from project to project.	It requires J2EE JMS specification knowledge. All J2EE application servers will follow this spec. It means: Study once, apply it every project!
Quality	The homemade or project-oriented asynchronous framework component is usually not fully tested and powerful enough.	It is certificated and fully tested as a product. E.g. Oracle Application Server use AQ as the implementation, which provided assumed atomic messaging transaction.
Benefit: <ol style="list-style-type: none"> <li>1. Less development</li> <li>2. Less learning curve.</li> <li>3. Better quality.</li> </ol>		

- Example: Data connection pool vs. DataSource

	Self-Developed Connection Pool	J2EE DataSource
Configuration	Use configuration file in the application. A rebuild is required.	Via admin web page.
Development	Require development. The quality may not be good.	J2EE certificated.
Supportive functions	Usually only support basic JDBC connection.	Server usually provides multiple connection types, e.g. XADatasource, OCI driver ...
<b>Benefit:</b> <ol style="list-style-type: none"> <li>1. It is more robust, comparing to homemade connection pool.</li> <li>2. It is easy to configure.</li> <li>3. It supports more types and transactions.</li> <li>4. It can be applied within the JTA transaction boundary.</li> </ol>		

- Example: Log4J vs. Application Logging

	Log4J	Oracle Application Server Logging
Config files	To change config files, you have to edit it in the code and do a rebuild and redeployment.	Via admin web page.
View	First of all, you have to know where Log4J config file location is; Read the config files to find the file locations; Log to server to view the file.	Via admin web page.
<b>Benefit:</b> The production support and developer can easily admin the log with no requirement on log4j knowledge. The access via Web page is easier and friendly to filter the logging data.		

### 4.1.3 Use the application server smartly

Rule: Do not couple to any specific application server.

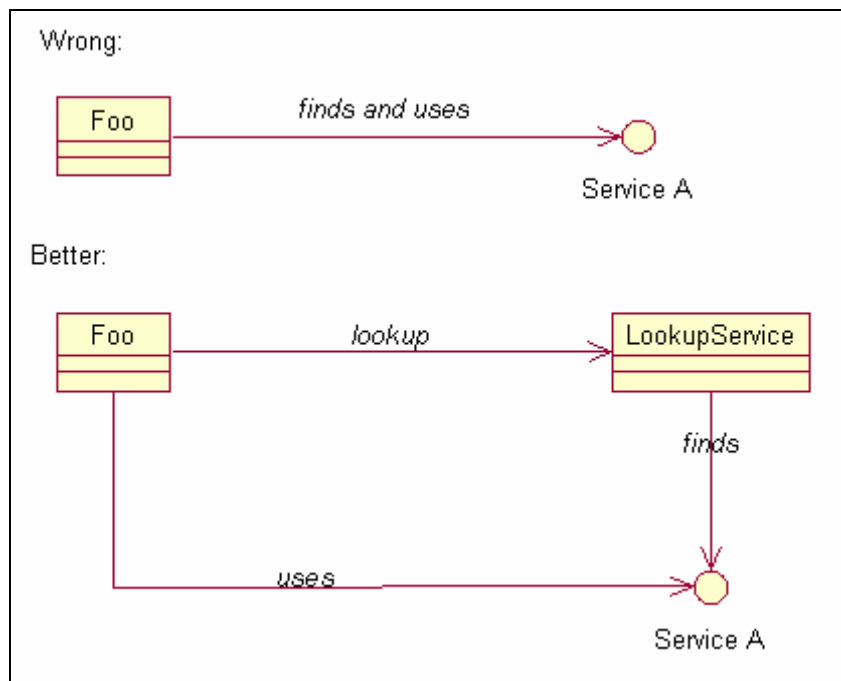
- Purpose
  1. The application must be portable to any other J2EE application server.
  2. Application server is powerful but we should not couple it to the application.
- Recommend Pattern: Service locator

Pattern overview:

Locator (lookup) service provides the layer to isolate the coupling between applications with application server.

As a result, when the underlying application server is changed, the application will not be affected. The only change will be the configuration and the new factory.

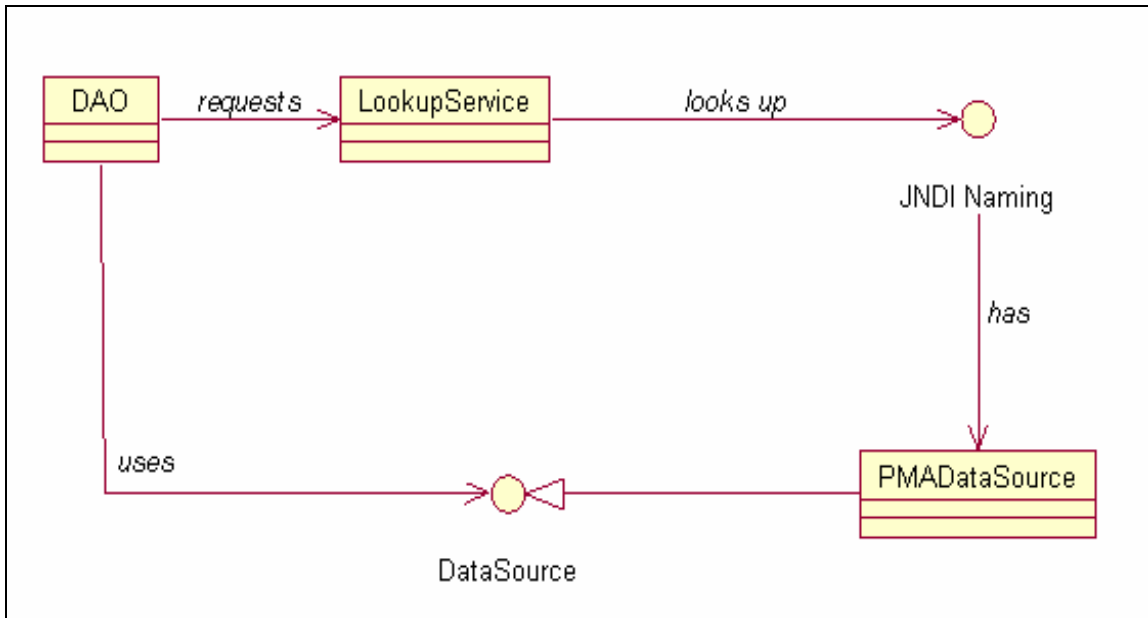
A figure to illustrate the locator pattern is shown as below:



### Example: J2EE Service (DataSource: javax.sql.DataSource)

When using J2EE service in the application server container, a service locator can encapsulate the JNDI naming and AS's JNDI looking specification from the application.

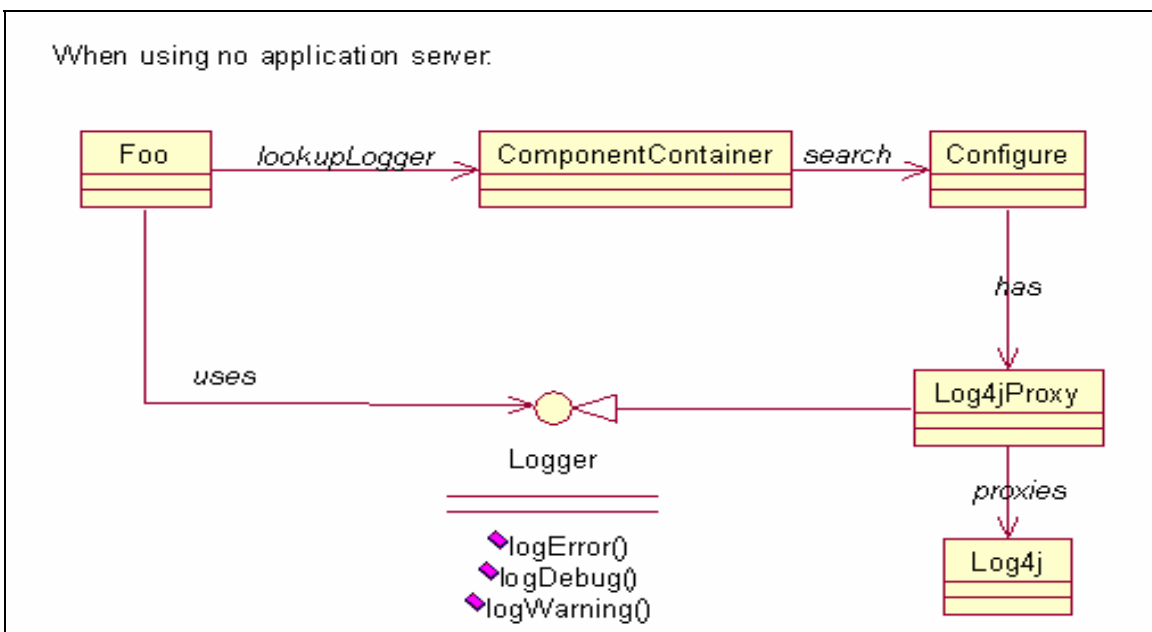
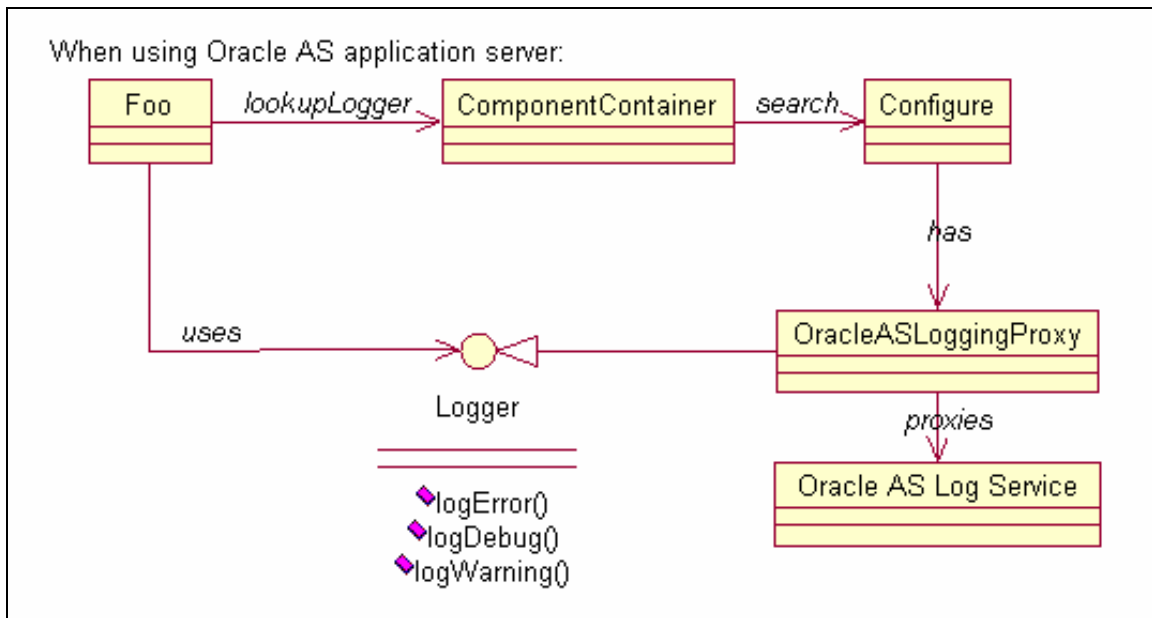
A figure is given at the below to show a DAO component use the lookup service to get the DataSource. From such design all codes in the application will get a data source regardless the detail of the infrastructure's operation. Because DataSource is a J2EE service, all these can be done in J2EE configuration via admin web page. There is no involvement of the Procase framework team development.



### Example: non-J2EE Service (Logging)

When a non-J2EE service is required by the application, a component container behaves as the service lookup. The following figure shows two cases. One shows when the underlying logging service is provided by Oracle Application Server. An Oracle Application Server Logging proxy is introduced to wrap all Oracle AS specific components. Another shows when the underlying logging service is provided by Log4j. A Log4j proxy is introduced.

From such design, all codes in the application will use Logger service. If we change the instance of Logger to Log4jProxy, such a change is transparent to the application. Because Logger is not a J2EE service, Logger is a class created by the Proc case framework team.



## 4.2 Interface Driven Design

### 4.2.1 What's interface driven design?

1. Simply speaking, any class should contact interfaces instead of other class directly.
2. The interface is the specification of a class, a component.
3. The modification of the class and the component will not affect the interface user as long as the specification of this interface is not changed.
4. The user of a service will force on high-level abstractions rather than low-level implementation details.

### 4.2.2 The benefit of interface driven design

Decoupling.

It reflects the fact that what A needs to know is a (part) function specification that B provides. If so, when B is modified as long as it meets the specification, A will not be affected.

Testability

A is testable regardless if B is ready or function correctly.

Some common patterns are introduced here as the best practice of interface driven design.

### 4.2.3 Recommended pattern: Inversion of Control

#### 1. Introduction

Assume Class A has a relationship with Class B: it wants to use the services of Class B. The usual way to establish this relationship is to instantiate Class B inside Class A. Though this approach works, it creates tight coupling between the classes.

Class A

```
{  
  ...  
  B b = new B();  
  b.service1();  
}
```

You can't easily change Class B without modifying Class A.

Class A

```
{  
  void setB(Binterface b)  
  {  
    b.service1();  
  }  
}
```

To eliminate the coupling, you can have a configuration inject the instance of Interface Binterface (Object "b") to the instance of Class A (Object "a"). If you want to change the implementation of B later on, you simply change the configuration object. *So, the control of how Object "a" gets the reference of Object "b" is inverted.*

#### 2. Benefit

Object "a" is not responsible for getting the reference to Object "b". Instead, the Configurator is responsible for it. This is the basis for the IoC design pattern.

#### 4.2.4 Recommended pattern: Factory / Abstract Factory

Factory pattern and abstract factory pattern are recommended as the creational pattern. By applying them, the code is made highly reusable and extensible. The addition of new components (classes) will not impact the old components (or system).

### 4.3 Spring Framework

#### 4.3.1 What's Spring Framework?

Spring is an open-source framework to address the complexity of enterprise application development.

Spring makes it possible to use plain-vanilla JavaBeans to achieve things that were previously only possible with EJBs. Any Java application can benefit from Spring in terms of simplicity, testability, and loose coupling.

Spring makes developing enterprise applications easier.

#### 4.3.2 The benefit from its features

- *Lightweight*—Spring is lightweight in terms of both size and overhead. The entire Spring framework can be distributed in a single JAR file that weighs in at just over 1 MB. What's more, Spring is nonintrusive: objects in a Spring-enabled application typically have no dependencies on Spring-specific classes.
- *Inversion of control*—Spring promotes loose coupling through a technique known as inversion of control (IoC). When IoC is applied, objects are passively given their dependencies instead of creating or looking for dependent objects for themselves. The spring container gives the dependencies to the object at instantiation without waiting to be asked.
- *Aspect-oriented*—Spring comes with rich support for aspect-oriented programming that enables cohesive development by separating application business logic from system services (such as auditing and transaction management). Application objects do what they're supposed to do—perform business logic—and nothing more. They are not responsible for (or even aware of) other system concerns, such as logging or transactional support.
- *Container*—Spring is a container in the sense that it contains and manages the life cycle and configuration of application objects. You can configure how your each of your beans should be created—either create one single instance of your bean or produce a new instance every time one is needed based on a configurable prototype—and how they should be associated with each other. Spring should not, however, be confused with traditionally heavyweight EJB containers, which are often large and cumbersome to work with.
- *Framework*—Spring makes it possible to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file.

### 4.3.3 Guideline of Spring Framework

Because Procase Java Team always utilizes the component-based development. We encourage using Spring containers and other useful services.

### 4.3.4 Spring Framework Examples

Usage:

The major use of the Spring framework is the container. (Package: context)  
We will also reuse some nice components spring provides:  
Cache and JDBC-wrapper. JDBC Wrapper is a good model to write neat JDBC in DAO.

Scenario:

A DAO need java.sql.Connection; without Spring, we will code the following  
Class DAO1 {  
void doSelect(String lastName)  
{  
    javax.sql.DataSource ds = JNDIContext.lookup("DataSource"); ← Coupling to J2EE  
    Container, e.g. Oracle AS ...  
    java.sql.Connection conn = ds.getConnection();  
    ... do queries ...  
}}

With Spring:

```
Class DAO_SPRING {  
private DataSourceProvider m_dsProvider;  
void doSelect(String lastName)  
{  
    java.sql.Connection conn = m_dsProvider.getDataSource().getConnection();  
    ... do sqls ...  
}  
void setDataSourceProvider(DataSourceProvider dsProvider)  
{  
    m_dsProvider = dsProvider;  
}}
```

```
interface DataSourceProvider  
{  
    javax.sql.DataSource getDataSource();  
}
```

In the Spring config:

```
<bean name='OracleDatasourcrProvider' class='DataSourceJNDILookup'/>  
<bean name='DAO_Spring' class='DAO_SPRING'>  
    <set property='dataSourceProvider' bean='OracleDatasourcrProvider' />  
</bean>
```

Then Spring framework use this config to create a class "DataSourceJNDILookup" which knows how to do JNDI lookup to find a data source, it will assign this instance to instance of DAO\_SPRING so that DAO\_SPRING can function with a datasource.

Result:

So, DAO1 will have to do JNDI context lookup and so on while DAO\_SPRING requires "DataSourceProvider" to finish its function with no knowledge of JNDI functions.

Advantage:

1. One day, we dump J2EE container and the DataSource provider is changed to Procace-developed datasource. If so, we have to go the DAO class to change the JNDI logic. But for DAO\_Spring, a change of configuration is sufficient. No code change is required.  

```
<bean name='ProcaceConnectionPool' class='procase.sql.ConnectionPoolProvider'/>
<bean name='DAO_Spring' class='DAO_SPRING'>
  <set property='dataSourceProvider' bean='ProcaceConnectionPool' />
</bean>
```
2. DAO\_SPRING class focuses on SQL logic instead of overhead logic of getting a connection ...
3. DAO\_SPRING class is pure Java and no JNDI technical coupling.

## 5 Guidelines and Reference

Design	Patterns	Procace OO Design guideline
	Strategies	Procace OO Design guideline
Java Coding	Document	Procace Java Coding Standard
	Standard	Procace Java Coding Standard
	Review	Procace Java Coding Standard
Other coding	Page Fragments	We recommend 300 lines as maximum of each JSF component. We recommend 10% of comment. (A 30 second reading rule is recommended. It means a reader should be able to understand a code page in 30 seconds.) This standard will be evolved during the review.