Basic Design Guidelines for Java Application Developers

SDN Community Contribution

(This is not an official SAP document.)

Disclaimer & Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

Basic Design Guidelines for Java Application Developers

Applies To:

Applications developed in Java for SAP Enterprise Portal (SAP EP) and/or SAP Web Application Server (SAP Web AS).

Summary

When developing Java web applications intended to run on SAP EP and/or SAP Web AS, it is important to design them to be flexible and open for change, well-tested, and with reusable components from other applications developed within your organization. In the following article I share some design guidelines that I find most valuable.

By: Nitzan Levi

Company: SAP

Date: 02 Jun 2005

Table of Contents

Development Guided by an Application Framework

Overview

Among the many challenges that exist in software development are:

There are many challenges in software development today. Among the are:

- Projects must be developed quickly, yet with high-quality.
- Development must result in better code maintainability and application stability.
- Development should employ useful design patterns.
- Development should seamlessly integrate with existing legacy systems while providing a path to more advanced technology, without downtime or expense.
- You must be able to decouple the application implementation from specific products/vendors (like DB, EAI, etc.)

In the face of these challenges, the software development community has turned to application frameworks to guide development. A "framework" is a defined support structure in which another software project can be organized and developed. Typically, a framework may include support programs ,code libraries, and a scripting language amongst other software to help develop and glue together the different components of your project. Application frameworks allow consistency in software design and provide low-level services that developers can use (and reuse) to speed development. The right application framework can dramatically reduce software development and maintenance costs.

Here are some guidelines for using application frameworks.

When to Use an Application Framework

1. Use frameworks for hiding complex coding that is repeating itself.
2. Use frameworks for handling expensive or sensitive resources and operations.
3. Use frameworks for interaction with third-party products.

What to Use

When it comes to using frameworks, one typically struggles with questions such as, "What framework should I use?", "Is there an appropriate framework for me?", etc. Here are some basic guidelines to help support your decisions.

1. **Standard frameworks** – It is generally recommended to use standard frameworks. Today, the software development community is becoming more standardized and you can see that the major software vendors are cooperating in order to define those standards. The framework itself might be implemented by varies vendors, but it complies with a specific standard. Using those frameworks ensures that your code is using a mainstream framework that better integrates with other implementations (vendor decoupling), and that it is proven and tested. In Java, you can find several standard frameworks, such as the J2EE framework standards.
2. **Open source frameworks** – If no standard framework is available, then you can use frameworks developed as open source projects (you have to make sure that you can use it from a licensing standpoint). Today, you can find many of your development needs in the open source community. The advantage in these frameworks is that they are based on scenarios exactly the same as yours, and they are developed and evolve according to end-user needs. The problem with these frameworks is that you have no commercial commitment for support and future releases; thus, you may find yourself in a position where you have to maintain the code yourself. When choosing an open source framework, try to determine whether this open source will last (there are several open source initiatives that became the de facto standard, such as those coming from APACH – struts, Log4J, and more).
3. **Vendor-specific frameworks –** If you didn't find an open source framework, or you don't want to use one, you can use vendor-specific frameworks if available. One of the pro's of using vendor-specific frameworks is the support; one of the cons is the coupling to a specific vendor (like SAP Web Dynpro).
4. **Homemade frameworks** – If none of the options above suit you, then you can always develop your own framework and tailor it to your specific needs. The disadvantage of this approach is that the development can be long and complicated, and the fact that you control what would be in the scope of the framework increases the temptation to keep on developing and adding new features.

How to Implement

1. Design it according to the KISS concept (Keep It Stupid Simple) as much as possible.
2. The frameworks should expose only your application needs - not theoretical needs.
3. Make sure the framework is less complex for the client application to use then the original tool/operation (the framework should help you).
4. If the framework uses/hides third-party products, do not use third-party terminology or concepts when implementing it; keep it as generic as you can (think about the possibility of replacing it with other products).
5. Make sure your framework does not create new problems (performance, conceptual, etc.).

Use the MVC (Model View Controller) Pattern

MVC was first widely defined in the book "Design Patterns" by Erich Gamma et al. It defines a separation of concerns in a program where the model defines the internal data structures of the program, the view defines how the model is rendered to the user, and the controller performs the actual actions in the program that affect the model. All user-interactive applications should stick to the MVC pattern (for more details on MVC pattern, just perform a simple search on the Internet). What it means is that there should be separation between the following types of logic:

- Data presentation and user interaction logic
- Business logic
- Data access logic

Each level of logic should be kept separated in order to ease changes, reusability, and consistency of the code.

How to Implement MVC in a Nutshell

For portal applications, the MVC pattern should be implemented in the following way (this is general without going into details of frameworks):

1. The data presentation and user interaction code will be written inside the Web Dynpro components.
2. Business Logic components should be implemented in one of the following ways:
   2.1. In a different layer built from regular Java classes that will be part of the WebDynpro components.
   2.2. In a different layer built from J2EE classes (session beans) that will run separately in the J2EE server.
3. The data access logic should be on a separate layer implemented on top of the J2EE engine.

Note: When designing and implementing code (on any layer), always try to decouple it from other layers so it is more flexible and less aware of deployment issues.

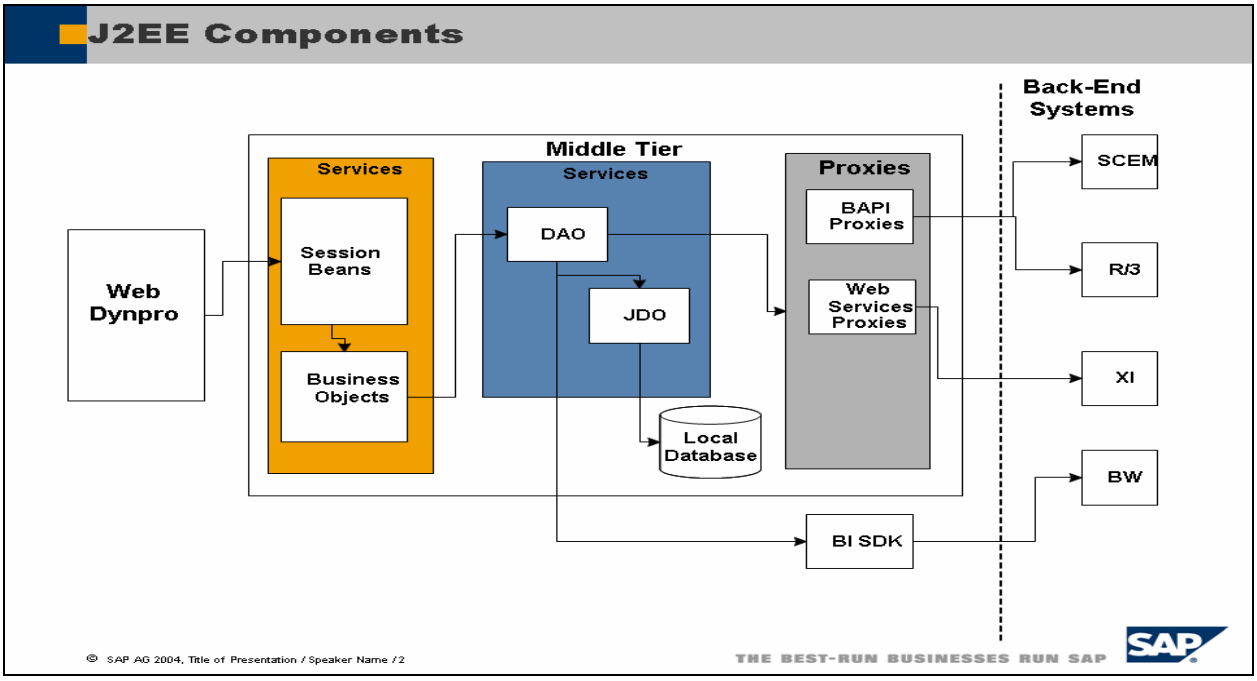

Figure 1: This figure shows an example of MVC implementation in SAP environment.

(DAO – Data Access Object, JDO- Java Data Object)

Interactions Between Applications

Interaction between applications sometimes can be complicated and likely to change (for example changing middleware products, changing applications, etc.). In order to reduce integration costs, follow these guidelines:

1. Application integration code should be implemented in a dedicated layer.
2. The application integration layer should be coded according to J2EE standards and run on the SAP Web AS J2EE Engine.
3. Application connectors – When connecting to other systems, the following rules must be kept:
   3.1. Do not keep/use connectivity parameters inside the code (should be external).
   3.2. Where possible, use standards for connectivity (i.e. J2EE JCA standard).
   3.3. Where possible, use infrastructure vendor frameworks (i.e. SAP connector framework, SAP SLD concept, etc.) both for out-of-the-box connectors and custom-implemented connectors.
4. If using EAI tools for application integration (like SAP Exchange Infrastructure or other connectors), the specific tool API should not be used directly, but:
   4.1. Used with standard framework wrappers (like JMS in case of messaging tools).
   4.2. Homemade specific wrapper – build dedicated frameworks for abstracting the service from the specific tool implementation.
5. Do not use asynchronous tools for synchronous communication.

Database Access

1. DB access code should be implemented as a separate layer (see DAO design pattern).
2. Access to DB should be done according to industry standards (JDBC).
3. Access code to DB should be kept standard/generic and not DB vendor-specific (ANSI SQL).
4. Any use of specific DB vendor functionality should be approved and wrapped in a way that this code would be able to be replaced if needed with minimum changes of code.
5. DB Connections creation should be done by SAP J2EE DataSources and not directly by the code for the following reasons:
   - More effective/optimized code for creation.
   - Ability for external configuration (connection parameters, pool config, etc.).
   - Advanced resource management (connection pooling, etc.).

Author Bio

Nitzan Levi is a Senior NetWeaver Solution Architect in the NWSO with a great deal of experience in design and implementation of large and medium-scale enterprise applications. Nitzan has a high level of J2EE expertise and experience with the SAP NetWeaver platform, as well as other vendors (i.e. the IBM WebSphere family).