

Taking the Pain out of Large-Scale Java Development Projects

Managing development environments

by Wolf Hengevoss
and Christopher Hearn

You know how to write good Java code and deployment to a server is no mystery either. But have you ever had to work in large development teams, maybe geographically dispersed (off-shoring...)? Ever had to address the pain of application software updates?

So often, when evaluating Java development tools, the focus is on productivity in writing code. While this is clearly important, it is essential not to underestimate the importance of managing the entire life cycle of an application – from setting up the development environment, to managing software delivery and maintaining the applications over the life cycle. The larger the project, the more critical this is, both from an efficiency and a financial point of view.

Over the past 30 years SAP has gathered in-depth experience in developing applications with large, geographically dispersed development teams. The delivery of software to the customer and the management of software upgrades and synchronizing these updates with customized code at the customer site are all issues that SAP has mastered. Now SAP is bringing the same high-quality approach to the Java world.



Wolf Hengevoss is a member of the SAP NetWeaver product management team.

wolf.hengevoss@sap.com

SAP Web Application Server

SAP Web Application Server is a key component of SAP's integration and application platform. SAP NetWeaver provides tool support for productive, model-driven and service-oriented Java development, but also, and this is key, an infrastructure that provides full-scale support for all phases of the life cycle – the Java Development Infrastructure (JDI). This infrastructure – tightly integrated with the SAP NetWeaver Developer Studio, delivered with the application server – reduces overall development TCO and provides for reliability and flexibility in the deployment and change management process.

Synchronized Team Development Made Easy

The JDI gives you the best of both worlds – you can still develop in Java in

your local environment, but your team's work is also synchronized via a central development environment.

Within the JDI, the Design Time Repository (DTR) handles file versioning to ensure that all developers are working from the same set of code. Developers access the central service via the SAP NetWeaver Developer Studio, check out files, produce new versions in the local file system, and check them back in after successful local testing. In the DTR perspective, you can compare versions, check version or revision history, and so on. During development, you can synchronize repository and local file systems whenever you wish, or even interrupt the connection between the two and reconnect later.

From the DTR you can manage:

- Different versions of a development object in the same repository
- Multiple states of a software component (development and consolidation of several releases)
- Multiple users making modifications to the same development object (with conflict detection)

Each state of software component development is represented in one workspace. The information about the state of a workspace can be propagated to other workspaces so you can synchronize the work of development teams using various instances of the DTR.

The DTR provides features for change management in a distributed, multi-user development environment. As you replace older versions of files with newer ones, the DTR handles this process centrally and keeps the version history. For more complex projects, though, you'll need more than that. Suppose modifications are occurring directly in end-users' systems, so various versions are being developed in parallel and multiple DTRs are in place. The version history is always deployed along with the files for global version history of the DTR. As a result, versions created in parallel are detected automatically across repository boundar-

ies. Then there are times when, during code modifications, you may not want to have an earlier version automatically overwritten by updates – instead, the DTR supports the merging of two colliding versions, allowing you to combine the advantages of the newer version with your modifications. What's more, to reduce the maintenance effort as much as possible, you would want to integrate bug fixes from older releases into newer ones from the maintenance cycle. The DTR supports this, since changes are always deployed as a whole set of versions, instead of individually. This approach to change management means that the results are unaffected by the sequence in which the changes are applied. With its innovative approach to distributed development, the DTR enhances productivity and reduces costs of development throughout the whole product life cycle.

Component-Based Development for Efficient Projects

The JDI takes a component-based approach to development. A component hierarchy distinguishes multiple levels of granularity:

- **Development objects – the finest level of granularity:** Tables, Java classes, and project files that are stored as versioned files.
- **Development components (DCs):** The units of the development and build process – they group development objects into larger development components, whereby one DC can contain multiple DCs.
- **Software components:** The deployment and installation units that are made up of DCs represent larger building blocks of products.
- **Product:** The complete solution.

DCs are at the heart of application development and follow a simple principle: only the parts declared to be public are visible to other DCs. In other words, to use one development component from another DC, you need to explicitly declare

this usage. This explicit declaration of dependencies between DCs and precise relationships between objects allows the encapsulation of functionality that leads to a fine-tuned, highly efficient build process in the Component Build Service.

Component Build Service

Speed up the build process and avoid errors with the Component Build Service. Ever experience night-builds of complete applications that fail – followed by another 24-hour wait for the next build? With the JDI, these days are over.

Like the DTR, the Component Build Service (CBS) is a J2EE application that uses a database. It hosts all Java archives needed or produced during software development. For each software component state, a buildspace is set up to contain these archives. You can trigger a central build in the CBS at any time. Central builds apply to modified DCs only, along with any DCs that have dependencies with the changed archives.

This DC build approach allows you to correct errors in smaller chunks, reducing bug-fix cycle times. A failed build process will not affect the build process of any other DC.

The CBS also provides:

- J2EE cluster support for high performance
- Automated build scripts for Java development
- Automatic rebuild of dependent development components after changes to objects

After a successful build, the CBS automatically makes the sources and archives available for use by other developers. Because all archives are centrally stored and up-to-date in the CBS, it is an ideal source for retrieving or updating used libraries in your local file system. Faster build cycles and a current build environment significantly reduce development costs, time, and errors, especially in large projects.

Managing the Development Environment

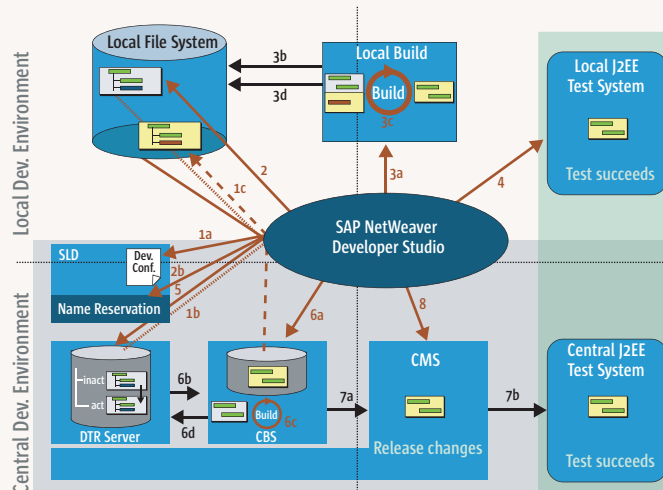
The development process starts with the definition of a product in the Software Landscape Directory. Here you define which software components are used in the product and the dependencies between them.

This information is imported into the Change Management Service (CMS) where you define the environment for the developers. You create workspaces and buildspaces by defining a track that's used

Java Development with the JDI in Place

Figure 1 outlines the basic steps involved in synchronizing local and central file systems and then, after development, using the JDI's central build and deployment features.

1. Once the development configuration file is imported into the SAP NetWeaver Developer Studio (1a), you synchronize your local file system with the sources in the Design Time Repository (1b), and with the archives in the Component Build Service (1c). Then develop your Java application as you normally would on any local development environment.
2. Edit sources (check out, change, create, or add files).
3. Build archives locally.
4. Test your build results in a local test environment.



From here, the JDI automatically takes care of synchronization and even deployment into the test environment.

5. After successful testing, update (check in) sources in the DTR.
6. Build archives centrally. First, trigger the build (6a). The sources and archives are loaded automatically into the CBS (6b), then the build starts according to the provided build scripts (6c). After a successful build, sources are activated automatically (6d).
7. Deployment to the central test environment is automatic.
8. Release changes for further processing.

to describe the development environment for one software component state. Fill the buildspaces with all the libraries required for a development configuration. This in turn is imported as an XML file into the SAP NetWeaver Developer Studio, defining the access to those objects the developer needs for his/her task.

After development, the developer releases his/her work to the CMS again. Here the QM triggers the import into the consolidation system, after central releases approves and assembles a software component release. For the next release, it's not necessary to physically set up a new system; simply define a new track.

This gives you complete control of the product life cycle from product definition to application patches.

Key Takeaways

Let's summarize the advantages of this approach:

- The development infrastructure is set up and managed centrally.
- All sources are stored centrally in the

DTR and retrieved for the central build process, and all archives are stored in the CBS, ensuring that you use the latest version of sources. You're assured a greater degree of safety when using archives.

- Clearly defining dependencies and encapsulating functions facilitates reuse and maintenance of development components.
- Distributed versioning and concurrency control allow you to manage large development projects taking place in different locations. Due to the DTR's versioning capabilities, modifications are not overwritten during updates, but can be integrated into the new version.
- You'll find a centrally managed system landscape – there's no need for each developer to know precisely where to deploy an archive.

For more information on the SAP Web Application Server and the Java Development Infrastructure or to download a trial copy check out www.sdn.sap.com.