SAPinsider

This article appeared in the Oct ▪ Nov ▪ Dec 2003 issue of *SAP Insider*. It appears here with the permission of WIS, publisher of *SAP Insider*. Visit **www.WISpubs.com** for additional resources.

SAP Java Development Infrastructure Supports Developers from Project Start to Finish

by Wolf Hengevoss, SAP AG

Any application development project includes tasks that go far beyond just writing application code. Inevitably across the life cycle of your application — from setting up the development environment, to managing software delivery issues, to maintaining the application after deployment — the larger the project, the more developers need an IDE that can support them in those "housekeeping" tasks.

With SAP Web Application Server 6.30, SAP created a state-of-the-art integrated development environment (IDE), the SAP NetWeaver Developer Studio, which provides tool support for developing Java applications in the SAP environment.[1] What's more, this IDE can be connected to an infrastructure that takes care of most of those non-development tasks: the **Java Development Infrastructure (JDI)**. The JDI is a key differentiator between SAP NetWeaver Developer Studio and other approaches to a typical Java IDE. It's the difference between a comfortable environment for a few standalone developers, and an IDE that can also scale up to an industrial-strength environment for even hundreds of developers, complete with all the bells and whistles of full-scale support for every phase of development.

SAP's creation of the NetWeaver Developer Studio is based on experience gained with the ABAP Workbench, a highly productive tool built for hundreds of developers in one system. SAP built the ABAP Workbench to provide a consistent development environment for each member of your project team and to make this consistency as easy to achieve as possible. When SAP made the move into the Java world, though, it soon became clear that the Java IDEs on the market didn't meet the needs of development teams that large. So it became SAP's goal to create an equally reliable and efficient system for the Java world. In a short time, this goal has already been achieved — and even exceeded — for many functions of the SAP NetWeaver Developer Studio. The Java Development Infrastructure is an ongoing part of this vision. It is currently rolled out for use in SAP's own development efforts and will be made available aligned with Release 6.40.

This article introduces you to the JDI and some of the functions — change management, version control, and automatic deployment, among others — that, with SAP Web Application Server and NetWeaver Developer Studio tools, will yield further reductions in development time and costs, easier error correction, and greater reliability for processes throughout the application life cycle.

The Main Pillars of the JDI

The key elements of the JDI are:

- *Design Time Repository* (DTR): A repository for designtime development objects (sources) that provides highly efficient storage and distributed versioning of J2EE application components.
- *Component Build Service* (CBS): Stores Java archives and provides immediate and incremental builds upon request from the IDE.
- *Software Logistics* (SL): Consists mainly of *Change Management Service* (CMS), which handles the flow of the software, and *Software Delivery Manager* (SDM), which is responsible for deployment of a Java application across the entire development system.

With all these pieces, you may expect installation and configuration to take a lot of effort for the development team. But the good news is that each developer gains access to the JDI simply by importing a single *Development Configuration* file. After the Development Configuration file is imported, the elements of the JDI work together as one, and all the

[1] See Karl Kessler's article for a fuller introduction to the SAP NetWeaver Developer Studio in this issue of *SAP Insider* (www.SAPinsider.com).

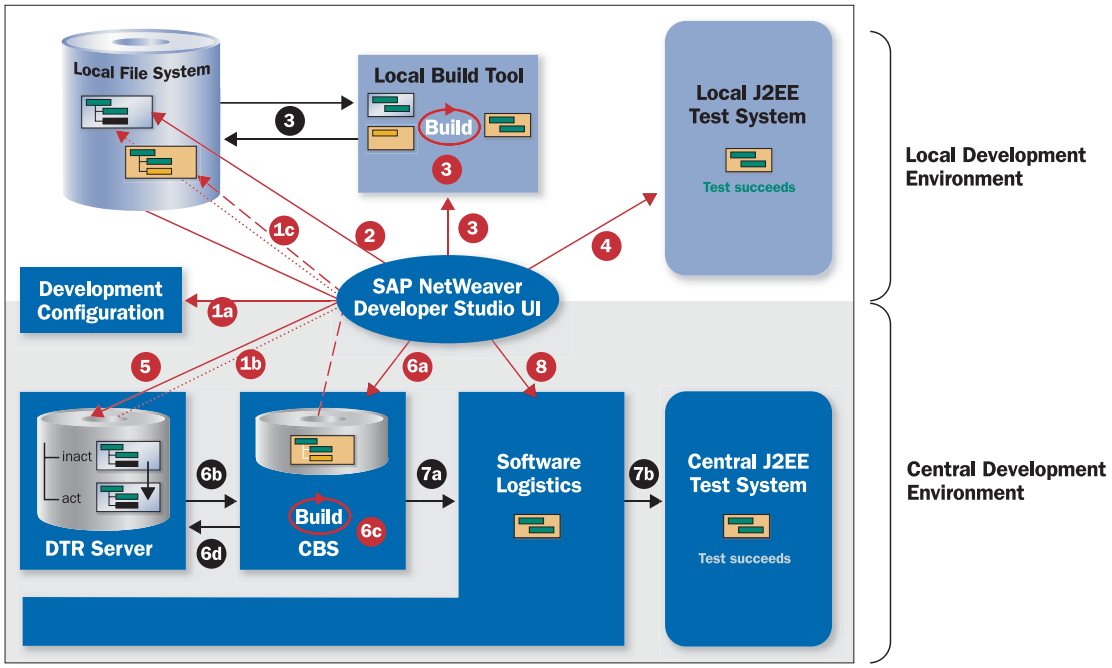

Figure 1 Development Process Using SAP NetWeaver Developer Studio and the JDI as Your Development Environment

sources and libraries needed are available directly to the developer.

A Refined Approach to Synchronized Java Development

The JDI is built to be seamlessly integrated with the Eclipse-based SAP NetWeaver Developer Studio to give you the best of both worlds — you can still develop in Java on the local environment, but your team's work is also synchronized via a central development environment (see **Figure 1**).

In the setup in Figure 1, the developer manages all tasks from SAP NetWeaver Developer Studio. Most tasks between JDI components are handled automatically; you will only have to take care of the direct connections to the Developer Studio (shown in red in Figure 1). The sidebar "Java Development with the JDI In Place" offers a quick overview of this process.

A Closer Look at Component-Based Development

All development using SAP NetWeaver Developer Studio and JDI is fully component based and can be broken down into one of these three elements:

- *Development objects* (tables, Java classes, Web Dynpro project files, etc.), which are stored as versioned files.
- *Development components (DCs)*, the units of development and build. They group development objects such as Web Dynpro projects.
- *Software components (SCs)*, delivery and installation units (e.g. SAP HR) containing DCs.

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.[2] In other words, to use a 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 encapsulation of functionality that leads to a fine-tuned build process in the CBS.

Synchronize and Store Source Files in the Design Time Repository

The Design Time Repository (DTR) handles file versioning to ensure that all developers are working from the same set of code. The DTR server stores all kinds of files used at designtime, and developers can view this versioned file system[3] via SAP's Eclipse-based IDE, which hosts the DTR client. From this client-side DTR interface, developers can access the local file system to check in and check out files, compare versions, and so on, in a very user-friendly way.

[2] This is also called "component interfacing."

[3] This file system is physically stored in a relational database using WebDAV/DeltaV protocol.

During development, you can synchronize repository and local file systems whenever you wish, or even interrupt the connection between the two — thus combining the flexibility of a local file system with the reliability and scalability of a relational database.

From the DTR you can also:

- Manage different versions of a development object in the same repository.
- Maintain multiple states of a software component.
- Manage multiple users making modifications on the same development object.
- Manage the development of the same object in different workspaces[4] — whether they are in the same repository or spread across many repositories in different locations.

The DTR also provides new 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 must always be transported along with the files for *global version history* — the history must always be available with the version, even after transports to other installations of the DTR. As a result, versions created in parallel are detected automatically across repository boundaries.

Then there are times when, during 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. The DTR supports this, since changes are always transported by 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 new approach to distributed development, the DTR enhances productivity and reduces costs of development throughout the whole product life cycle.

Access Up-to-Date Archives in the Component Build Service

Like the DTR, the Component Build Service (CBS) is a J2EE application that uses a database. But instead of storing version and change management information, it hosts all Java archives

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[5] file is imported into the SAP NetWeaver Developer Studio 1a, you will 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.[6]

7 Automated deployment occurs, via Software Logistics 7a, into a central test environment 7b.

8. Release changes for further processing.

[4] A workspace is the DTR term for a line of code.

[5] For those of you with ABAP experience, think of the Development Configuration file like the ABAP Repository — in that case, once you logon to an SAP system, the environment is there for you. With the Development Configuration file, you are importing a file that fully describes the technical environment. Then, in the JDI, development configurations are centrally maintained.

[6] See Karl Kessler's article "Your 'Easy Way In' to Web Dynpro Development" in the April-June 2003 issue of *SAP Insider* (www.SAPinsider.com).

needed or produced during software development.

For each software component, a *Buildspace* is set up to contain these archives. You can trigger a central build in the CBS anytime. Central builds are limited only to changed files, along with any files that have dependencies with the changed archives. This "incremental build" approach allows you to correct errors in smaller chunks, dramatically reducing bug-fix cycle times. The CBS also provides:

- J2EE cluster support for high performance.
- Automated build scripts for Java development.
- Synchronization of build tasks — if used archives are changed during a build, a rebuild is started with the new version.
- Automatic rebuild of dependent development components after changes of 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.

Look to Software Logistics for Transport and Maintenance Functions

Once you're finished coding the application's functions, there's still some work to be done. You need to deploy the application in different systems and make it available during various stages of the product life cycle — the start of the correction phase, the setup of the delivery to customers, and product maintenance cycles, for example. For big development teams and large numbers of end-users, this can only be handled efficiently if there is some kind of central management.

In SAP's development infrastructure, Software Logistics (SL) offers three main benefits for developers:

- A centralized development system is readily available. Access to the Java Development Infrastructure is provided through simple import of the Development Configuration file.
- As you move from the development to the correction phases, transports are managed centrally.
- After the build, you can avoid any transport issues. From the CBS, all runtime objects are transported *automatically* into, for example, a central test system.

Summary

To highlight some of the most important facts about the JDI:

- The development infrastructure is set up centrally. With a simple file import, all the latest sources and archives needed for development are centrally available for your specific development project. You can download all the objects you need to your local file system, so you have the flexibility of development on your local PC, combined with the advantages of a central development environment.
- All sources are stored centrally in the DTR and retrieved for the central build process, and all archives are stored in the CBS. You are assured of automatically using the latest versions. This process allows for comprehensive tests in early project stages, sparing you from many surprises when putting everything together in the last big step! The CBS's support for incremental builds also leads to shorter bug-fixing cycles.
- You're assured a greater degree of safety when using archives. Changes to archives automatically lead to rebuilding dependent components. Only when you've reached a successful central build are your sources activated.
- Clearly defining dependencies and encapsulating functions eases reuse and maintenance.
- Distributed versioning and concurrency control allow you to manage large development projects taking place at different locations. Due to the DTR's versioning capabilities, modifications are not overwritten during updates, but they can be integrated into the new version.
- You will find a centrally managed system landscape — there's no need for each developer to know precisely where to deploy an archive.

Most of this may sound familiar to those of you used to developing in the ABAP Workbench. But it sure means a big step forward for development in Java!

For more information on the Java Development Infrastructure for SAP NetWeaver Developer Studio, see **www.sap.com/netweaver**. ■

Wolf Hengevoss studied natural sciences at the University of Kaiserlautern. He joined SAP in 1999 as a member of the product management group of the ABAP Development Workbench department. He can be reached at wolf.hengevoss@sap.com.