A guided tour of Java software development lifecycle management with SAP NetWeaver Development Infrastructure (NWDI)

Part 2 — The development process

by Wolf Hengevoss

Wolf Hengevoss
Senior Product Specialist, SAP NetWeaver Life-Cycle Management, SAP AG

Wolf Hengevoss studied natural sciences at the University of Kaiserlautern. He joined SAP in 1999 as a member of product management. He has worked in the Basis group, focusing on topics such as the Computer-Aided Test Tool and Business Address Services. Since the early stages of SAP Exchange Infrastructure (now known as Process Integration), he has worked in the Java environment. Currently, his focus is on the rollout of SAP NetWeaver Development Infrastructure. You may reach him at wolf.hengevoss@sap.com.

SAP NetWeaver Development Infrastructure (NWDI) — introduced under the name Java Development Infrastructure (JDI) with SAP NetWeaver 2004, and now delivered as usage type DI with SAP NetWeaver 7.0 (2004s) — is an enterprise-scale Java development platform, designed for potentially hundreds of developers working in parallel on thousands of interrelated development components. It brings many of the change management capabilities long enjoyed by ABAP developers to the Java world by providing developers with a well-organized, sophisticated approach and toolset to address the complex challenges faced by real-world Java development. While Java developers previously struggled with locating source code versions, finding and tracking foreign libraries, handling discrepancies between local and central environments, and managing conflicts when multiple developers work on the same object at the same time, NWDI changes all of this. It offers enhanced source version control and searching, sophisticated archive management, incremental build capabilities, and parallel development capabilities.

In the previous installment of this two-part article series,[1] I introduced the key players in the NWDI architecture and essential NWDI concepts, including an overall view of the NWDI-enabled Java development process. In this article, I show you how to apply what you have learned by walking you through a practical development example that follows this process, so you can see NWDI in action.

Since the concepts from the last article are a prerequisite for understanding the demo, I'll quickly review them before diving into the example; though to get the most out of this article, I recommend that you read the previous article first.

[1] "A guided tour of Java software development lifecycle management with SAP NetWeaver Development Infrastructure (NWDI): Part 1 — Fundamental concepts" (*SAP Professional Journal*, July/August 2007).

Note!

Although the example is based on SAP NetWeaver 7.0, this article also applies to SAP NetWeaver 2004. NWDI as discussed here with SPS 13 also fully supports development in SAP NetWeaver Composition Environment (CE) 7.1. You can apply what you learn here to NWDI development within SAP NetWeaver CE as well.

Architectural overview and essential NWDI concepts

Figure 1 summarizes the key players within the NWDI architecture:

1. **NetWeaver Developer Studio (NWDS)** is the Eclipse-based Java development environment within which developers build custom Java applications.[2] NWDI commands are integrated directly into the NWDS user interface.
2. The **Design Time Repository (DTR)** stores the complete version history of all source code for all Java development objects managed by NWDI in "workspaces." Developers access the DTR workspaces to check in and check out objects via NWDS, enabling parallel development and conflict detection.
3. The **Component Build Service (CBS)** builds deployable Java archives in a "buildspace" when a developer activates new or modified objects. The CBS ensures that only one version of an object is active at any given time, and that the active version compiles without errors. Like the DTR, developers use NWDS to access the CBS buildspace to activate objects.
4. The **Change Management Service (CMS)** manages the setup and content (i.e., the required sources and archives) of the development landscape along with the transport of objects during the development cycle. It centrally administers almost all of the necessary actions in the DTR and CBS. Administrators use the CMS's Web-based administration tools (mainly the Landscape Configurator and the Transport Studio) to prepare for development, and NWDS accesses the DTR workspaces and CBS buildspace via the CMS when developers want to check in, check out, and activate objects.
5. The **System Landscape Directory (SLD)** serves as a central registry of all systems and software in your landscape. It also includes a name service for generating unique names for development objects.[3]
6. The **User Management Engine (UME)** manages Web users, roles, and permissions.[4] UME data can reside in an LDAP directory, a local J2EE database, or an ABAP system.

As you can see in the figure, the NWDI server should be installed on a separate server from the Java development, consolidation, test, and production runtime systems you have in use. This is very important because you will use your single NWDI system to manage more than one Java landscape, and because you want to isolate your NWDI system from the instability, overall transient nature, and maintenance/upgrade schedule of your development system.

[2] NWDI has two main user interfaces: NWDS for developers, and a set of Web-based administration applications that provide administrators with direct access to the various NWDI components. You'll see both of these user interfaces in action when I take you through the development example. There are also some highly specialized command-line tools, including clients for manually accessing all of the NWDI components (e.g., for manually creating or deleting workspaces in the DTR), and a special tool for dealing with development components in a local build or for "refactoring" them (e.g., renaming them or moving them into another software component), but these are rarely used. These tools are described in the NWDI documentation.

[3] The SLD is not technically part of NWDI — it is a central part of SAP NetWeaver that is used by various SAP applications, such as SAP Solution Manager, Supplier Relationship Management (SRM), Auto-ID Infrastructure (AII), and Process Integration (PI), which was formerly known as Exchange Infrastructure (XI). For more on the SLD, see the article "A system administrator's practical guide to SAP System Landscape Directory (SLD)" (*SAP Professional Journal*, November/December 2006).

[4] The UME is analogous to the ABAP user management/security functionality on ABAP systems.

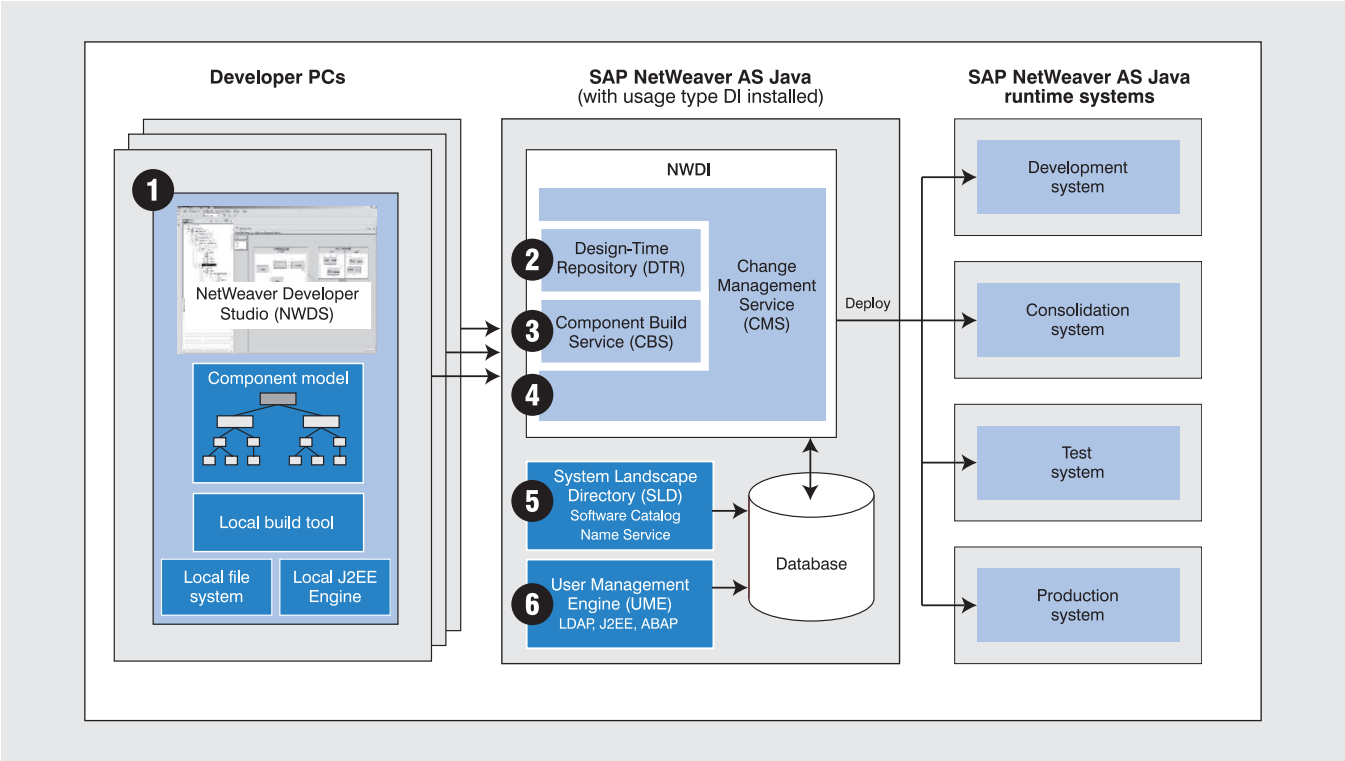

Figure 1 Key components within NWDI (DTR, CBS, and CMS)

Now let's briefly review the fundamental concepts you need to be aware of when using NWDI (again, for a detailed explanation, please refer to the previous installment of this article series). The most important is the programming model that defines the structure of software developed with NWDI — the SAP component model, which is a three-tier hierarchy consisting of the following objects:

- **Products** represent a business solution and are assigned a specific version (e.g., "we'll be working on version 1.0 of a custom tax calculation application").
- **Software components (SCs)** constitute a software product, and bundle together the various technical parts of a solution (e.g., a set of objects related to tax calculation).
- **Development components (DCs)** are the technical objects that constitute a software component, and group together actual Java development objects, such as Java classes and Java Server Pages (JSPs), into Eclipse-based development "projects" — e.g., a Java project, an Enterprise Java Beans (EJB) project, or an Enterprise Application project, depending on the purposes of the development work.

The remaining administrative concepts you need to know are tracks, development configurations, workspaces, and buildspaces:

- **Tracks** define the development environment, including the software components (both those to be developed and any existing ones to be used in development), the DTR workspaces and CBS buildspaces (forming "logical systems"), and the series of runtime systems or "phases" (development, consolidation, test, and production) that an application will pass through during its development lifecycle.[5] Different tracks are created for

[5] Objects are transported as "activities" (essentially change requests) in development and consolidation systems and as "software component archives" (SCAs) in test and production systems.

different development purposes (e.g., initial development, maintenance, etc.).

Note!

Do not confuse logical systems with the physical runtime systems (shown in **Figure 1**) that are set up in your SAP landscape. Logical systems are not physically installed — they are organizational units for handling different software states in a single NWDI instance. Logical systems are automatically generated for each track in the NWDI database, and consist of the DTR workspaces and CBS buildspaces that manage the source files and archives of a particular product state. To separate the development and consolidation states of a product's source files and archives, NWDI always generates both a development and a consolidation logical system for each track (logical systems are not generated for test and production, since software is never directly changed in these types of systems). Using NWDI, you can then assign the physical runtime systems in your landscape to your development work for all phases (development, consolidation, test, and production) for centralized testing and automated transport of development objects. Depending on your needs, you may assign certain phases but not others. If you do not use some phases, you can simply transport the objects to the next phase instead.

- **Development configurations** are XML-based descriptions of systems in a track that permit code changes (development and consolidation).[6] Developers must import development configurations into NWDS to begin their work.

Development configurations tell NWDS the location of the DTR and CBS in which workspaces and buildspaces will be created, the software components to be used for development, the dependencies between them that must be obeyed during the build process, the runtime systems to be used for the deployment, etc.

Note!

Since NWDI always generates logical development and consolidation systems, it also always generates corresponding development configurations for these systems.

- **Workspaces** are where developers do their development work. Workspaces contain the file versions for a software component in a specific track.[7] NWDI automatically defines an *inactive* workspace (where objects are developed) and an *active* workspace (where successfully built objects are stored) for your development project. Workspaces are stored in the DTR, and are displayed in NWDS in a hierarchical file structure.
- **Buildspaces** are areas in the database where the CBS stores imported archives and the build results of all activated objects. When a developer activates changes, the CBS loads the components needed for the build into a temporary folder and executes the build. If the build is successful, the results are written to the buildspace in the database and the object is made available in the active DTR workspace.

Since most of these concepts are new, let's take a brief look at how they work together:

- First, an NWDI administrator creates a track (using the CMS Landscape Configurator) to define

[6] Deployment-only systems (test and production) simply store the information needed for deployment into their runtime systems. Software is never directly changed in these systems, so these systems do not have development configurations.

[7] Technically, workspaces contain references to the files in the database, so the files can be used in more than one workspace without copying them and creating unnecessary duplicates.

the development environment and the specific set of "phases" (development, consolidation, test, and production) through which the application will pass during its development lifecycle.

- Next, an NWDI developer imports the development configuration information into NWDS so that NWDS can access the DTR, CBS, and required files during development and consolidation. When a developer works on objects in the inactive workspace of the DTR using NWDS, the new object versions are "stored" in a DTR activity, which is the unit of transport in development and consolidation systems.
- The developer then initiates a local build of the objects within NWDS. A local build tool, similar to that of the CBS, loads the objects to be built — as well as the latest (and activated, if available) versions of any components upon which these objects depend — and builds local versions of the objects for testing.
- Once satisfied with the test result, the developer checks in the activity containing the changes and attempts to activate it using menu commands within NWDS. The central (server-based) CBS collects the necessary sources from the DTR workspace and the active archives from its buildspace, places them in a temporary folder, and performs the build.
- If the build is successful, the CBS writes the results to the central buildspace in the database and the objects are flagged as active and added to the active workspace in the DTR.[8] The developer can then release the activity containing the objects, making it eligible for transport from the development system to the consolidation system (using the CMS Transport Studio).
- Once testing is complete in the consolidation system, the objects are assembled as software component archives (SCAs) for deployment (again, using the CMS Transport Studio) into the test and production systems.

I will go into more detail on these tasks when I walk you through the development process; here, I just wanted to give you an overall picture of how the concepts fit together.

With the necessary theory out of the way, it's time to get our hands dirty with a practical, but fictitious, development example — building a Web-based tax calculation application.

The step-by-step development process

Figure 2 on the next page shows the step-by-step process we'll follow to implement the example application, which I introduced in the previous installment of this two-part series. As you can see, the process steps are neatly divided across five discrete roles: software architect, NWDI administrator, developers, transport manager,[9] and quality manager. This demonstrates a key advantage of NWDI: clear segregation of duties with defined hand-off points between roles, each supported by an easy-to-use toolset — developer tasks are all done within NWDS, while almost all other activities are done within the centralized CMS Web administration toolset.

> ***Note!***
>
> From a permissions perspective, there are only two roles in NWDI — NWDI developer and NWDI administrator — so setting up team permissions in the system is very simple.

Over the remainder of this article, I explain the most important parts of each of these steps as we develop the example Tax calculation application using NWDI (the full details of these steps are available

[8] From this point forward, new development or changes must compile against this latest version in order to be activated.

[9] Depending on the size and needs of your organization, the NWDI administrator may also serve as the transport manager.

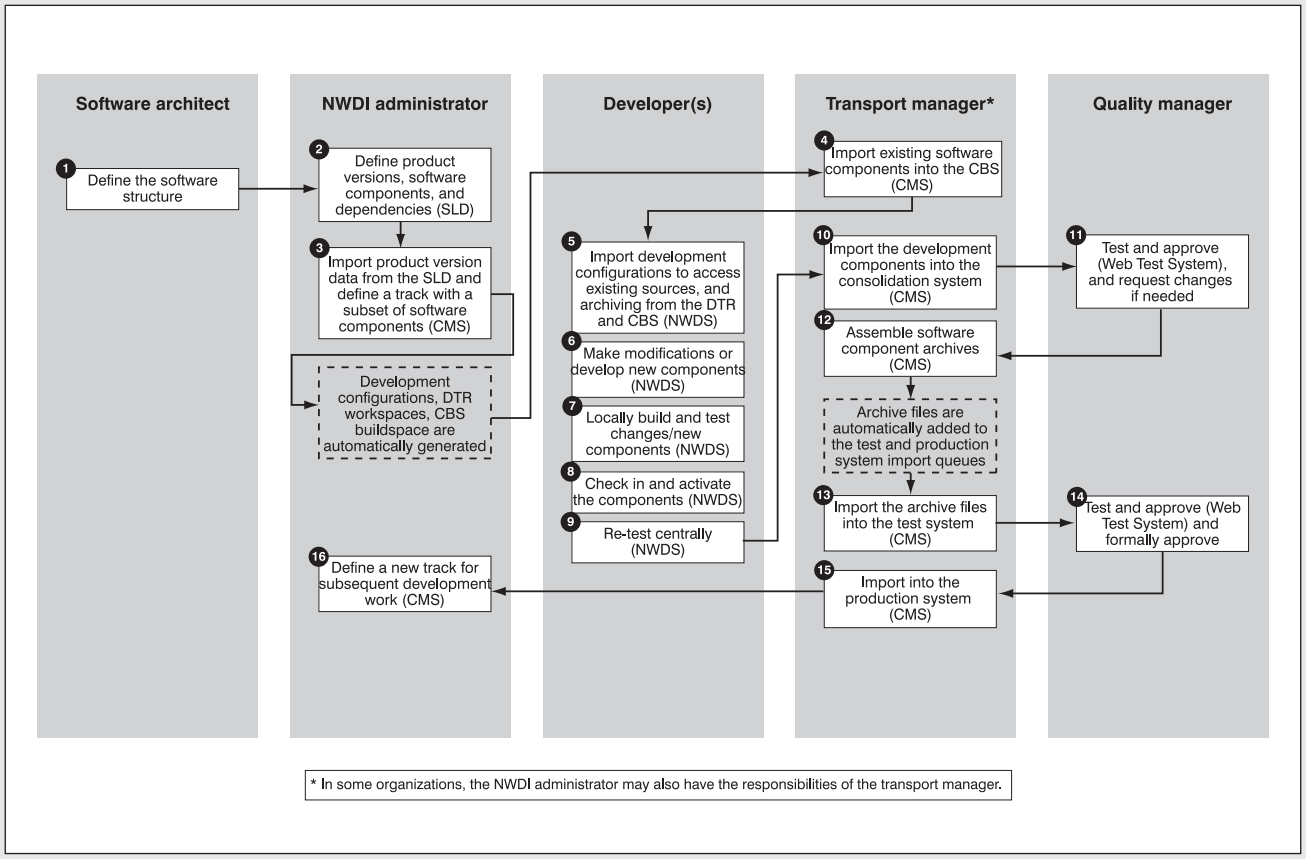

Figure 2 Complete NWDI-enabled Java development process

Note!

To follow along with the example, you need an NWDI system (i.e., an SAP NetWeaver 2004 or 7.0 system with the Java runtime installed), along with NWDS and the basic set of software components described in the next section (SAP_JTECHS, SAP_BUILDT, and SAP-JEE), which are delivered with NWDI. The necessary UME permissions will be configured automatically when you install the 7.0 version (you will need to configure them manually for SAP NetWeaver 2004 using the installation guide available at http://service.sap.com/instguides). You also need to configure your SLD, where both the technical and business landscape of a company is described.[10]

[10] For details on how to configure the SLD, see the *SAP Professional Journal* article "A system administrator's practical guide to SAP System Landscape Directory (SLD)" in the November/December 2006 issue, as well as the documentation available at http://help.sap.com and http://service.sap.com/sld.

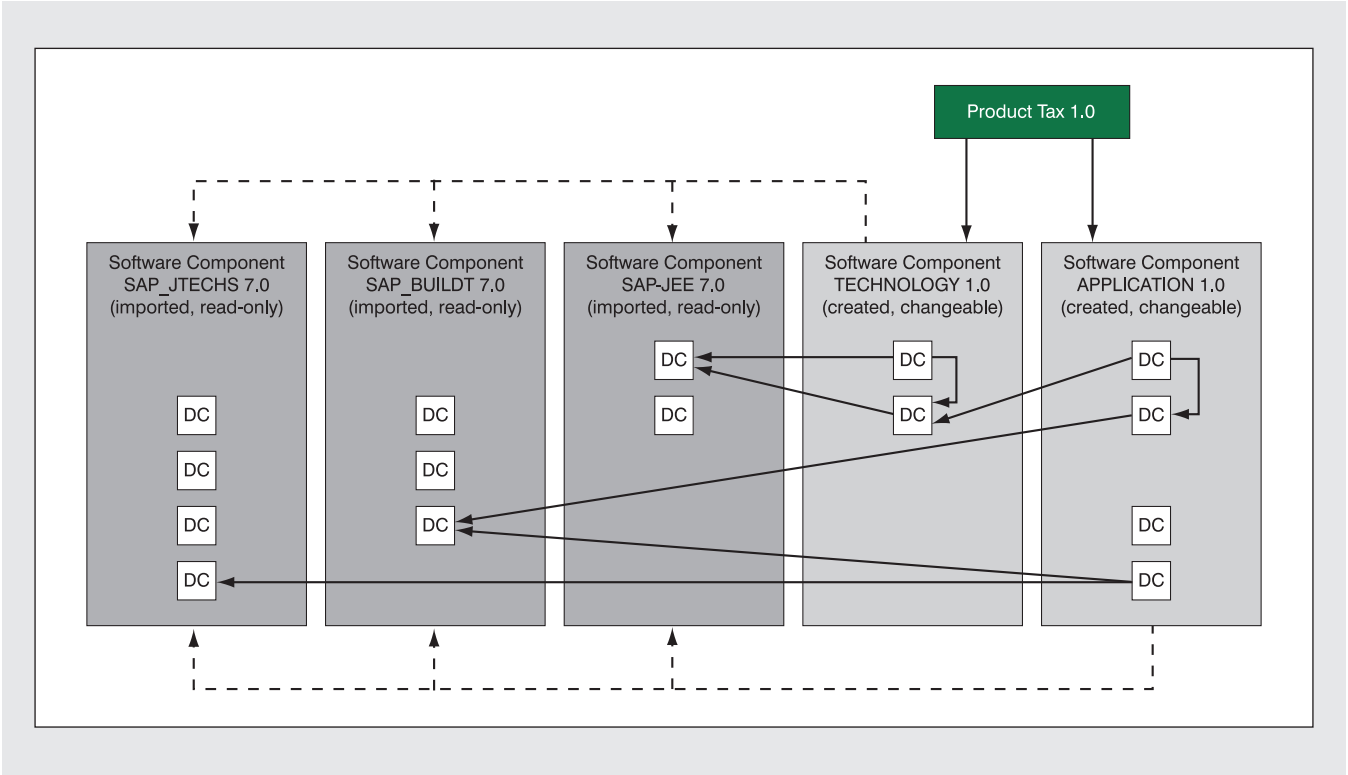

Figure 3 The structure of the example Tax application

from the SAP Help Portal at http://help.sap.com). There's a lot to cover, so let's get started!

Step 1: Define the software structure

First, the software architect must define the structure of the product — i.e., the business solution — including any new software components to be developed, the target platform release, and any dependencies to existing software components that developers will (or might) need to use as a part of the development process.

Figure 3 shows the structure defined for the example application. As you can see, we'll be developing the first version (version 1.0) of the product Tax, which is a Web-based tax calculation application that consists of two new software components and three existing software components.[11] The three software components in dark gray (SAP_JTECHS, SAP_BUILDT, and SAP-JEE[12]) are existing software components provided with SAP NetWeaver 7.0 that we will import for development use (they are needed for all NWDI Java-based development in SAP NetWeaver 2004 and 7.0). All the functionality in these three components will be available for use in the development of the two new software components

[11] Note that I won't go into detail on the application's screens or functionality, which are beyond the scope of this article. Here, I focus on the development of just a few parts of the application to give you an idea of how NWDI works. All components of the application are developed, tested, and deployed similarly with NWDI.

[12] These reusable software components are similar to the automatically available functions in an ABAP development system (e.g., the ability to create database tables or set up connections to other systems). SAP_JTECHS contains libraries of Java functionality (such as Web services prerequisites and JDO functionality), SAP_BUILDT contains parts needed by the build infrastructure of the CBS, and SAP-JEE contains parts of the SAP J2EE engine's API (such as the Open SQL functions needed to address various databases).

shown in light gray, APPLICATION and TECHNOLOGY; these components will house the tax calculation functionality and the user interface of the Tax 1.0 product, respectively.[13] APPLICATION will use TECHNOLOGY to provide the tax calculation functionality to users through the user interface, which means that the APPLICATION software component has a "dependency" on the TECHNOLOGY software component. Because these two components will be developed, they are changeable, meaning that developers can modify them; the existing SAP-provided components used by APPLICATION and TECHNOLOGY are read-only, meaning that developers cannot modify them.

> ***Note!***
>
> Software component dependencies are defined in the SLD, prior to the first concrete steps taken in the NWDI environment. Development component dependencies, on the other hand, are defined in NWDI.

As you can also see in **Figure 3**, each software component contains a number of development components that provide the software component's functionality (e.g., the tax calculation functionality is provided via Java classes in TECHNOLOGY and the user interface is provided via JSPs in APPLICATION). Development components can also be used by those in other software components — e.g., APPLICATION uses the tech/tax/calc development component in TECHNOLOGY to present the calculated values in the application's user interface, and TECHNOLOGY uses a Java Connector (JCo) development component in SAP-JEE for data exchange with an ABAP back end.

[13] The separation of back-end, reusable logic from user interface logic is strategic, especially for software that is being developed for sale to other companies. One reason is that the inclusion of source code in the compiled Java archive (JAR) file is specified at the software component level, and thus you may want customers to be able to modify/enhance the user interface but not the back-end logic. To do this, you'd tell NWDI to include sources in the user interface archive but not the logic archive.

Step 2: Define the product, software components, and dependencies

Once the software structure is planned, the NWDI administrator (in consultation with the software architect) must define a product for the application in the SLD, which essentially represents an open development project for the application. This product (more specifically, the product version) will serve as the root of all development work. The NWDI administrator must also define the software components that will be part of the project (or that will be created as part of the project), and any build dependencies between the software components. If any existing software components other than the basic three (SAP_JTECHS, SAP_BUILDT, and SAP-JEE) are required, installation-time dependencies can also be added to ensure that the components are installed on the server. However, the system does not check the installation-time dependencies, so while a missing build-time dependency will break the build, a missing installation-time dependency will not.

> ***Note!***
>
> Only software components that have been identified in the SLD in this step will be available to developers in NWDS, so this step is important! Information on software components delivered by SAP are automatically provided as SLD content; software components that you define you must add yourself.

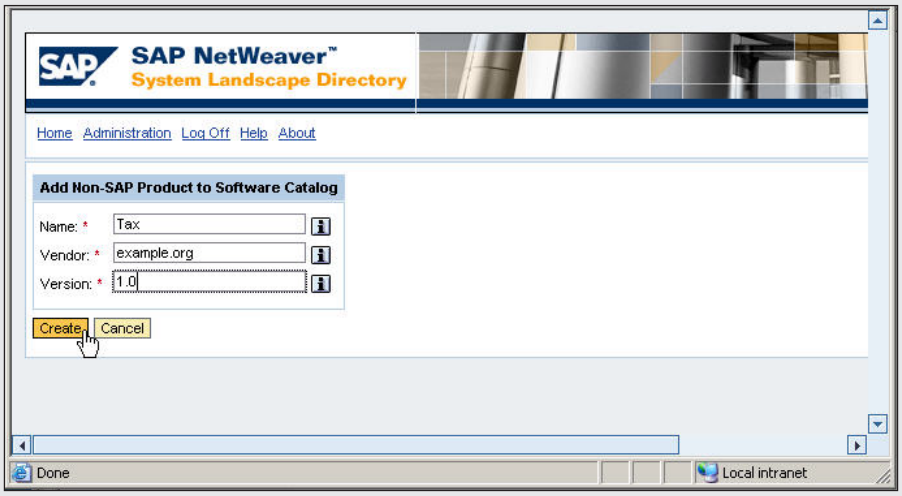

Figure 4 Creating a new product in the SLD

Define the product version

To define the product version, log onto the SLD as an NWDI administrator[14] and open the Software Catalog within the SLD's Web-based administration tool. Enter the following information, as shown in **Figure 4**:

- **Name:** The product name (e.g., "Tax")
- **Vendor:** Your company's Internet domain (e.g., "example.org")
- **Version:** The product version (e.g., "1.0")

Create the software components for the product

We need to create the two software components to be developed for the Tax product: TECHNOLOGY (which will contain the tax calculation functionality) and APPLICATION (which will contain the user interface elements). **Figure 5** (on the next page) shows the settings for the TECHNOLOGY software component. Choose the product and the software unit to which the new software component will belong (units help to group software components to simplify the organization of large products such as SAP NetWeaver), add your company's Internet domain to indicate the owner of the software component, and then add the name and the version of the software component (note that the production state is not currently relevant to NWDI). Create the APPLICATION component in the same way, only this time add a build-time dependency on TECHNOLOGY.

Define dependencies for the software components

Now, we need to specify dependencies for the software components — i.e., any additional software components on which the TECHNOLOGY and APPLICATION software components depend, and any build dependencies between the two software components.

In the SLD Software Catalog, select the Software Components tab, choose the software component you want to allow to use other software components (e.g., the TECHNOLOGY software component we just created), and go to the Dependencies tab. Select "BuildTime" for the context, click on Define

[14] This role is provided in the UME and will have the necessary permissions after the installation and configuration of NWDI.

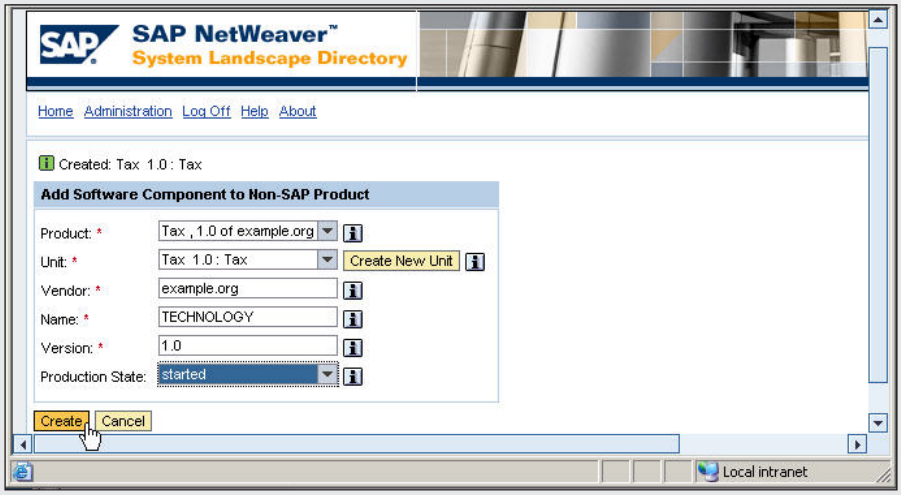

Figure 5 The TECHNOLOGY software component created for the Tax product

Prerequisite Software Components, and select the required software components from the list that appears (remember that the SLD stores the information for all software components in the system, enabling you to simply choose the software components from a prepopulated list).

> ***Note!***
>
> At a minimum, in SAP NetWeaver 2004 and 7.0, you must allow your software components to use the three required SAP-provided components (SAP_JTECHS, SAP_BUILDT, and SAP-JEE) by adding them as dependencies in the SLD. Otherwise, you will not be able to create a development component because the required components — such as the build infrastructure development components of SAP_BUILDT — are checked at build time. In an SAP Netweaver CE track, the required components are SAP_BUILDT, ENGFACADE WD-RUNTIME, and FRAMEWORK.

Figure 6 shows the definitions for the TECHNOLOGY software component. As you can see in the example, we've selected DI BUILD TOOL, SAP J2EE ENGINE, and SAP JAVA TECH SERVICES as prerequisite components of TECHNOLOGY, so that we can use the functionality in these components when we develop TECHNOLOGY. (These prerequisite components are the SAP-supplied SAP_BUILDT, SAP-JEE, and SAP_JTECHS software components, respectively, that are needed for all Java-based development in SAP NetWeaver 2004 and 7.0.)

As you can see in the example, we have also specified the APPLICATION component as a dependent software component, because APPLICATION will use TECHNOLOGY to provide the tax calculation functionality through the user interface.

Next, create the APPLICATION software component using the same settings, only in this case, add TECHNOLOGY as a prerequisite software component in addition to the three basic software components.

With the software components defined in the SLD, so that they will be available as development

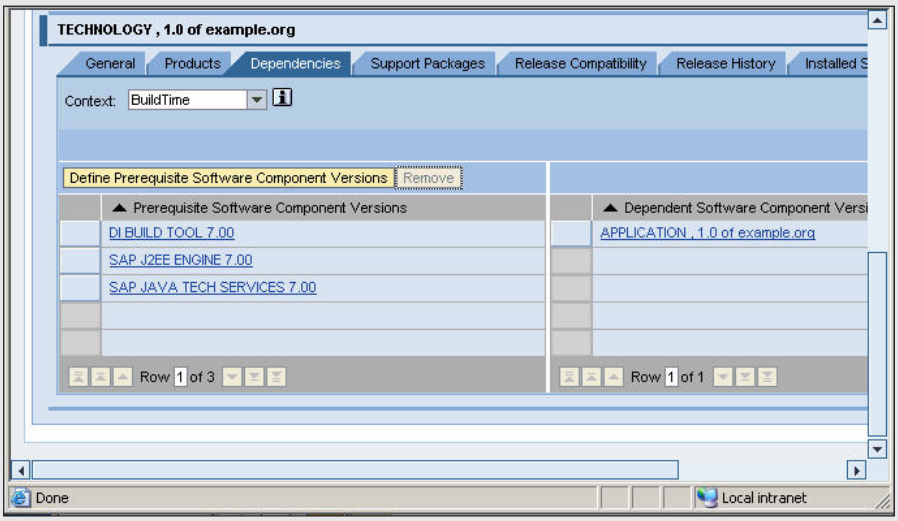

Figure 6 Defining the prerequisite and dependent software components for TECHNOLOGY

objects in NWDI, the next step is for the NWDI administrator to import the new data from the SLD into NWDI and define a track for managing the development objects using the CMS Landscape Configurator.

Step 3: Import the product data from the SLD and create a track

In this step, the NWDI administrator uses the CMS Landscape Configurator tool to define a track, which provides the development landscape for the complete Tax product, and to support all the development tasks (e.g., development, testing, etc.) required for the development objects. A track identifies the software components that can be modified or that are used as part of the development project, as well as the location of the CBS and the DTR, as mentioned earlier. A track shepherds a development object through the various "states" or "phases" of its lifecycle (development, consolidation, test, and production).

> ***Note!***
>
> The CBS location is where the buildspaces for your track are created, and where the build is performed, so it is the most CPU-consuming service of NWDI. If your team's requirements start to grow beyond the power of your first NWDI server, simply add another server with another CBS and refer to that server in your next track. As you can see, NWDI easily scales to support any number of developers.

Download data from the SLD into the CMS

First, we need to update the CMS with the information in the SLD, so that the software components we just defined are available for development work. On the NWDI initial screen, click on Change Management Service and log in at the prompt as

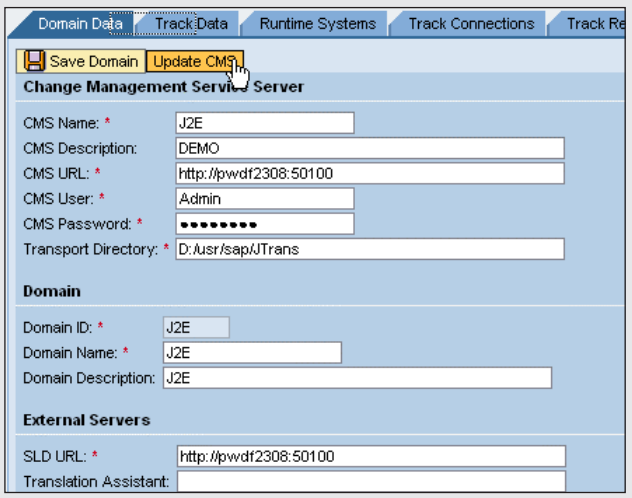

Figure 7 Updating the CMS

an NWDI administrator. On the Landscape Configurator screen, shown in **Figure 7**, go to the Domain Data tab and click on Update CMS in order to download the data from the SLD.

Note!

The Domain ID is set only once. All the tracks you create are based on this information. For example, the ID (J2E in the example) appears as part of each software component's name when it is built as an SCA (e.g., example.org~APPLICATION~J2E_APPL_C~20050215182030.sca). The remaining information — the CMS user data, the transport directory, and the SLD URL — can be changed if the CMS system moves or a different SLD should be used, for example. You will also frequently use the Update CMS function to update the SLD data — for example, when you want to create new software components to be used in a new track.

Create a track

Now we can create a track for our Tax 1.0 product. Select the product on the left side of the screen, as shown in **Figure 8**, and then go to the Track Data tab and define the following fields:

- Repository Type — The options are "DTR" for Java-based development (which we select for the example) and "XI" for transporting Exchange Infrastructure (now known as Process Integration, or usage type PI) design and configuration objects.
- Track ID, name, and description — For the example, we enter "TAX10" for the track ID, "TAX1.0" for the track name, and "Product TAX in version 1.0" for the description.
- Design Time Repository and Component Build Service URLs — When the track is saved, the DTR workspaces and CBS buildspaces are automatically created at the locations specified here to store the sources and archives used in the software development project.

The development configurations (one for development and another for consolidation), which default to the same name and description as the track, will be automatically generated when the track definition is complete. As mentioned earlier, the development configurations contain the "administrative information" defined in the track — the location of the DTR workspaces and CBS buildspaces, which we just defined, along with the software components to be used for development, dependencies between them, and the runtime systems to be used, each of which we will define momentarily. Developers must import the development configurations into NWDS to do their work.

After entering the information, click on Save. The the corresponding track ID, name, and DTR and CBS locations are defined.

Add software components to the track and define the packaging

Next, we need to add the software components that we want to be available for development work.

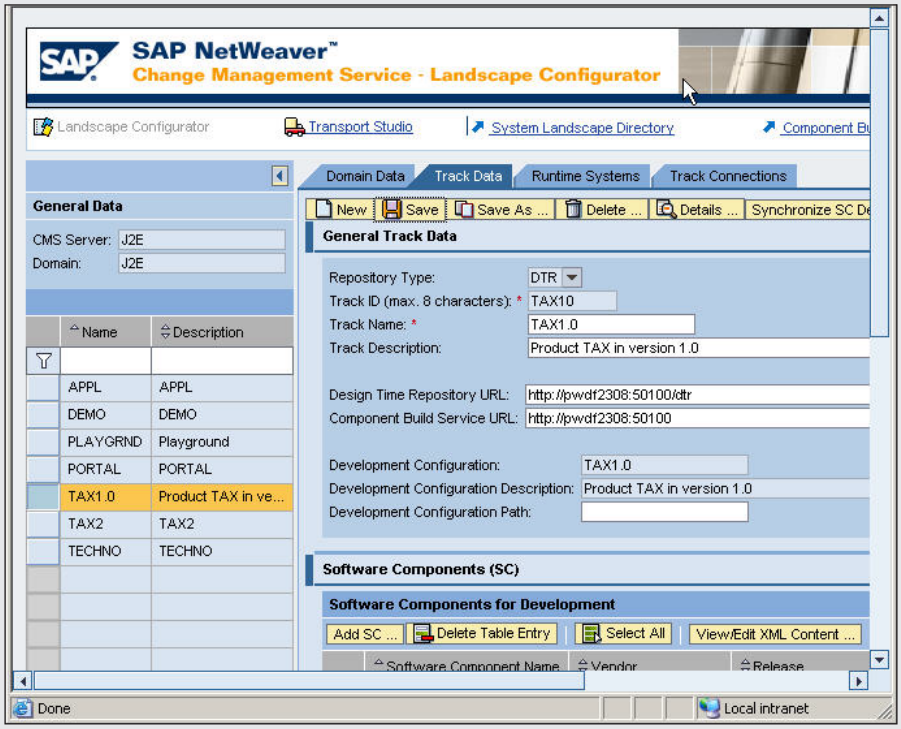

Figure 8 Creating a track for the example Tax product

Click on the Add SC button in the Software Components for Development section, select a software component from the list that appears (which is based on the SLD content we just downloaded), and click on the Add button. For the example, select the APPLICATION software component, as shown in **Figure 9** (on the next page), and then select the TECHNOLOGY software component as well. Remember that only the components from the latest update from the SLD are available for selection and therefore for development work, so be sure that you have the latest information from the SLD available.

Next, we need to define how each software component should be packaged (with source files, with archive files, or with both) when it is assembled into an SCA for use in tracks or in deployment to a test or production server.

> ***Note!***
>
> You should only place sources in an SCA when you want to allow modification of a piece of software, such as the development of the next version of a software component within your company.

Figure 10 on the next page shows the package definition for the TECHNOLOGY software component, which we'll package with source and archive files to allow for later modifications of the software. APPLICATION is packaged the same way.[15]

[15] APPLICATION could just as easily have been developed in a track of its own with TECHNOLOGY imported as a required software component — reuse is really that easy with NWDI.

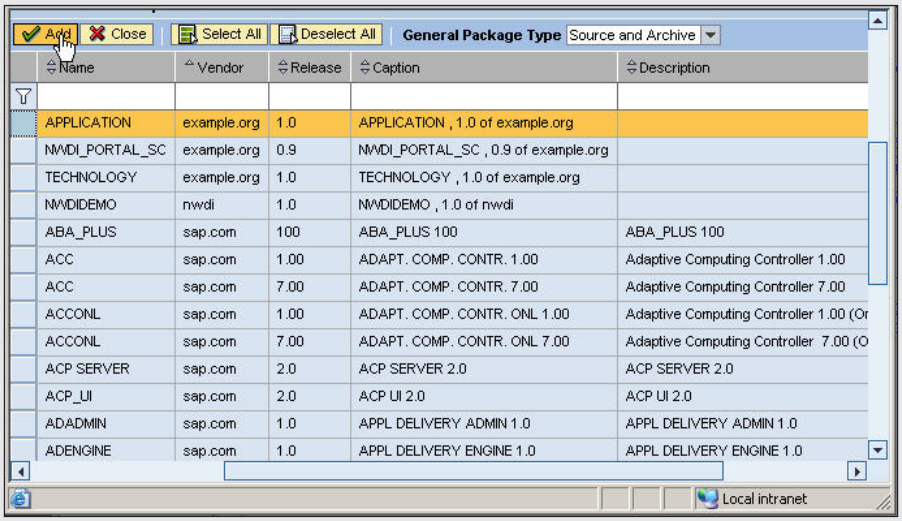

Figure 9 Adding software components to the track

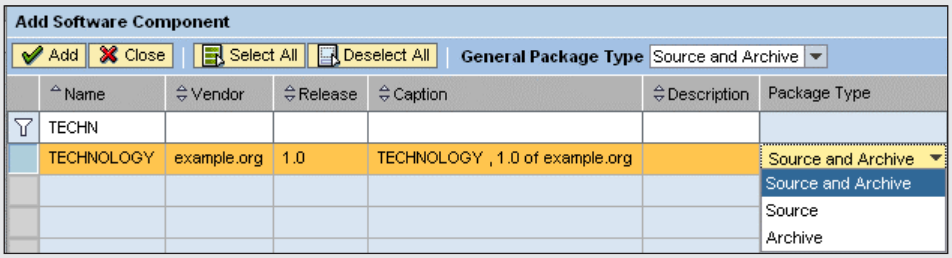

Figure 10 Defining the software component packaging

Note!

Since APPLICATION and TECHNOLOGY use the same software components (SAP-JEE, SAP_BUILDT, and SAP_JTECHS), they are shown only once under Required Software Components. The TECHNOLOGY component, which is a required component of APPLICATION, is not displayed under Required Software Components because it is already listed as an SCA for development and, in contrast to the required (only) software components, it will also be represented with a workspace (i.e., it will contain sources in this track).

As a result of the dependencies we defined in the SLD (refer back to **Figure 6**), the additionally required software components specified for TECHNOLOGY and APPLICATION (the three SAP-supplied SAP_BUILDT, SAP-JEE, and SAP_JTECHS software components for TECHNOLOGY, and these three plus TECHNOLOGY for APPLICATION) are retrieved automatically and displayed in the Required Software Components section (see **Figure 11**).

Define the runtime systems

Next, we need to define runtime systems for the phases of our development project. A runtime system is an instance on the server that runs the "states" or "phases" the object will go through during the development lifecycle.

> ***Note!***
>
> Remember that NWDI automatically creates "logical" development and consolidation systems consisting of DTR workspaces and a CBS buildspace to enable development. Tracks allow you the option to assign runtime systems to these logical systems to allow developers and the quality management team to centrally test new functions in the early stages of development.

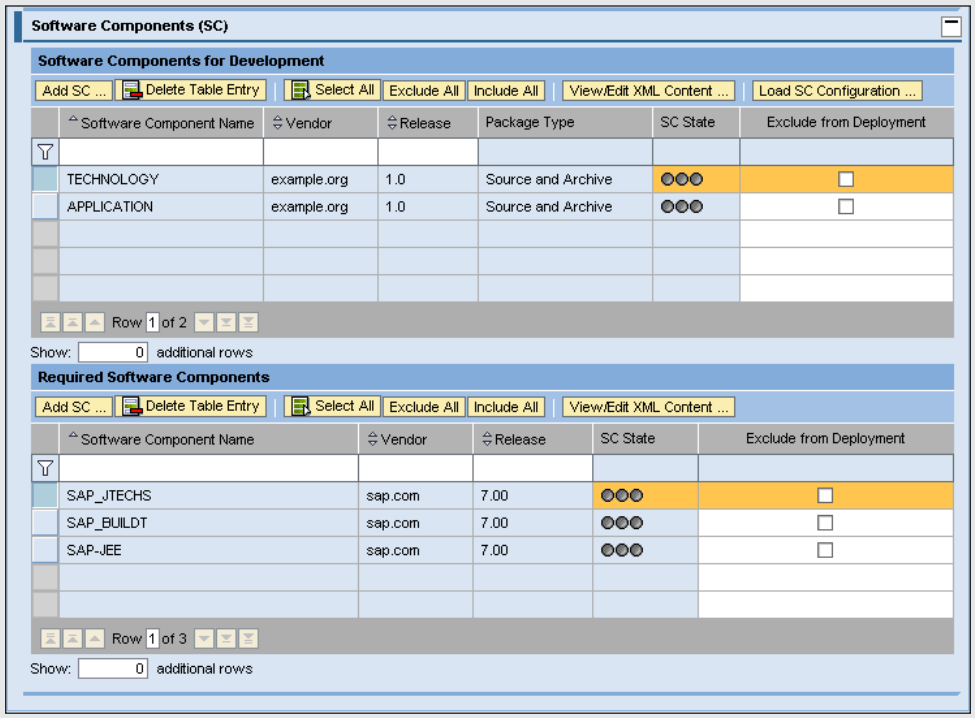

Figure 11 The required software components are added automatically

There are four types of runtime systems to choose from depending on the project and your particular needs:[16]

- **Development:** The development runtime system is a central test instance that all developers can use. If developers have SAP NetWeaver Developer Workplace (i.e., NWDS and a local J2EE engine), this runtime system is not absolutely necessary, because development testing can be done locally. It *is* necessary when developers have only NWDS without a local engine. Keep in mind, however, that centralized debugging limits the number of developers that can be in the development system. In addition, with a development runtime system all activated changes are automatically deployed once the developer triggers the central build of a development component, which means very early integration tests for all development components. While this is useful for seeing the application work as a whole in its early stages, it is not stable enough for the quality management team. The state of the runtime system also changes constantly because of the deployments triggered by each developer.

- **Consolidation:** In the consolidation runtime system, deployment occurs only if the administrator starts it. This type of runtime system is often used to have a system that is independent from the development runtime system, so that unchanging software states are guaranteed for the quality management team for a defined period of time. Automated deployment happens after the import and build of sources that have been released by developers in the development phase.[17] Since the administrator controls the import of activated changes, the state of this runtime system remains stable as long as needed. A consolidation runtime system is recommended over a development runtime system.

- **Test:** The test runtime system is where components that have been assembled into SCAs are deployed exactly the way they are deployed in a production runtime system. That's why the test runtime system is the most valid test case for your production system. Any stability issues would occur here just as they would in the production system and can be resolved before deployment to production.

- **Production:** The production runtime system is where the software is in actual production, so software vendors focused only on development may omit this type of runtime system.

While the runtime systems you choose to use are entirely up to you, it is generally a good idea to provide all developers with SAP NetWeaver Developer Workplace installations and, much like an ABAP development environment, have a three-system setup of consolidation, test, and production runtime systems.

To define the runtime systems, go to the Runtime Systems tab, and then select the runtime systems for the phases you want to use for your project. For simplicity, select Development and Consolidation for the example (instead of the recommended consolidation, test, and production setup), as shown in **Figure 12**. We then need to define for each selected system the Software Deployment Manager (SDM) host name, port number, and password, so that the automated deployment process via the CMS can trigger the SDM automatically. We also need to define the URL, user name, and password for the J2EE engine on which NWDI is installed; these settings are used by the Java Support Package Manager to integrate with NWDI deployments. Deployment of activated newly built archives into the consolidation runtime system will happen automatically when the administrator triggers the import of changes from the developers.

[16] For example, if developers are using SAP NetWeaver Developer Workplace (NWDS and a local J2EE engine), you might omit the development runtime system to save server resources, because development testing can be done locally. If you're a software vendor and don't run your own software productively, you can skip the production runtime system (note that in this case, the CMS Transport Studio will not include a tab for importing software components into a production runtime system; the same is true if a test runtime system is not used).

[17] Remember that a development runtime system is not a prerequisite for this step.

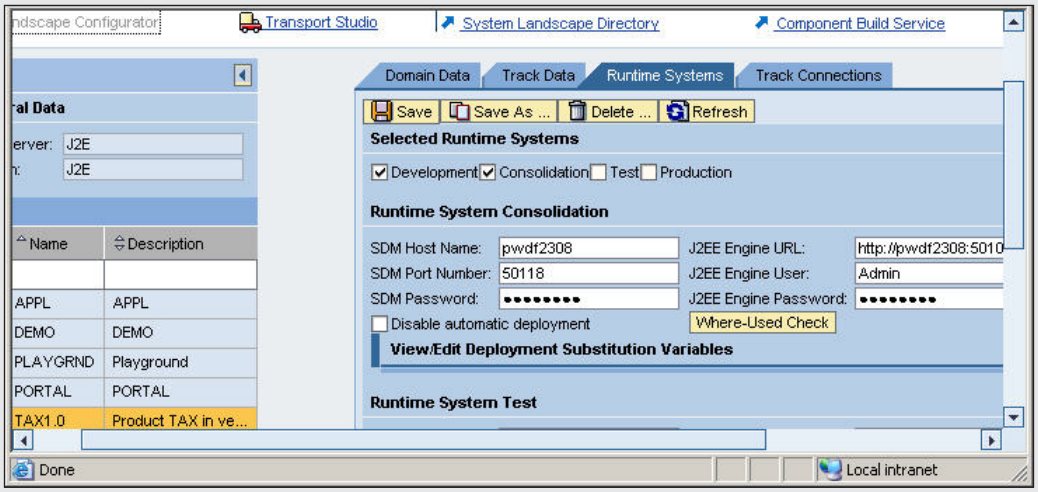

Figure 12 Runtime systems defined for the development and consolidation phases

Define any required track connections

If you are enhancing or modifying a product that is already being used in production, you might need more than one production runtime system with automated deployment, so that the production system's state is available for emergency corrections to the development and consolidation systems, for example. This can be configured on the Track Connections tab. Two types of connections are possible: Transport, which puts SCAs assembled in a development track into the import queue of the subsequent track (e.g., for development of the next version), and Repair, which puts bug fixes (activities) of a maintenance track into the import queue of the development system of a development track.

Let's say that we are creating the next version of our example tax product, and we have defined a track for managing its development called TAX2. We now can connect the two tracks with a Transport connection, so that the output (the SCAs of TECHNOLOGY and APPLICATION) is automatically available for development of the next version in a new track. In **Figure 13**, we have created a track connection of type Transport between TAX1.0 to TAX2, where TAX1.0 is the source track that provides the SCAs we want to transport and TAX2 is the target track to which we want to transport the SCAs. This means that if a

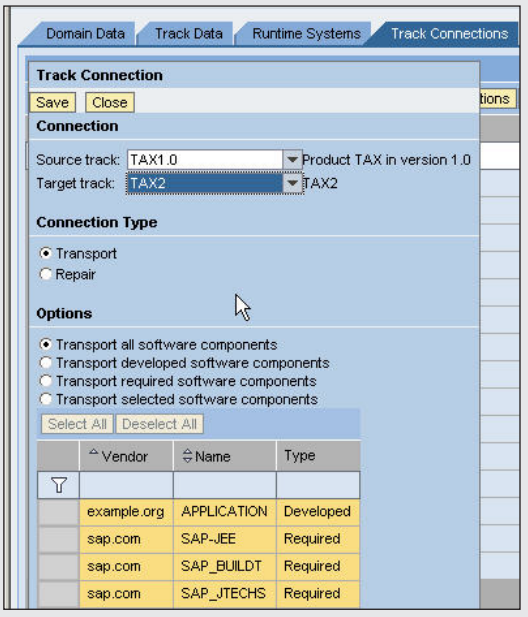

Figure 13 Defining track connections

version of Tax 1.0 is assembled, the sources will automatically become available in TAX2. If we had created a track connection of type Repair, fixes made to TAX2 could be automatically transported down to TAX1.0. The options on the lower part of the screen allow you to customize which software components are transported.

Tip!

If the same software version is running in multiple production systems, you can use track connections to automatically supply all of them with deployments ready for import.

Once the track has been defined, development configurations are automatically generated for the systems that permit code changes (development and consolidation), along with the corresponding DTR workspaces and CBS buildspaces needed by developers.

Note!

Communication in NWDI is based on standards such as HTTP (plus WebDAV and DeltaV for the DTR) and XML. The development configurations containing the information all developers need to access the DTR workspaces and CBS buildspaces are automatically written as XML files to be imported into the developer's IDE (NWDS, which is based on the open standard Eclipse framework).

At this stage, what will be developed and what can be used in the development process are now clearly defined (and with that the paths for all transports needed in that process). The next step is to import any files associated with the software components defined in the track.

Step 4: Import the software component files into the CBS

The last step before development can begin is for the transport manager to check in and import any existing software component files that are stored as SCAs into the CBS buildspace, so that the files are available for development work. Since we have not yet developed the APPLICATION and TECHNOLOGY software components, and therefore there are no files associated with them yet, we physically import only the archives[18] for the predefined SAP-JEE, SAP_BUILDT, and SAP_JTECHS software components.

To do this, go to the CMS Transport Studio, and select the track we created containing the software components for the development project. Go to the Check-In Tab (see **Figure 14**), which displays the SCAs contained in the track. Since it's the first version of the product, the required SCAs are the only ones available, so simply click on Select All and then Check-In.[19] (Remember that even though it has no associated files yet, the TECHNOLOGY component is included here because it is a required component of APPLICATION.) The available releases, support package (SP) numbers, and patch names for the software components are automatically provided for you based on the information from the SLD. You need to be sure that you select the correct SP version of the SCA. While **Figure 14** shows only a single SP version of each SCA, in most cases, there will be several versions available, and it is crucial that you choose the correct version — the version of the required software components must match both the NWDS version (which "expects" certain functions in those components), and the version of the target runtime systems.

[18] You can import both sources and archives for custom-developed components, but you can only import archives for any predefined components to be used in development.

[19] If there were multiple versions of the product, all of the SCAs associated with each of the product versions would be displayed, and you would need to individually select the relevant ones.

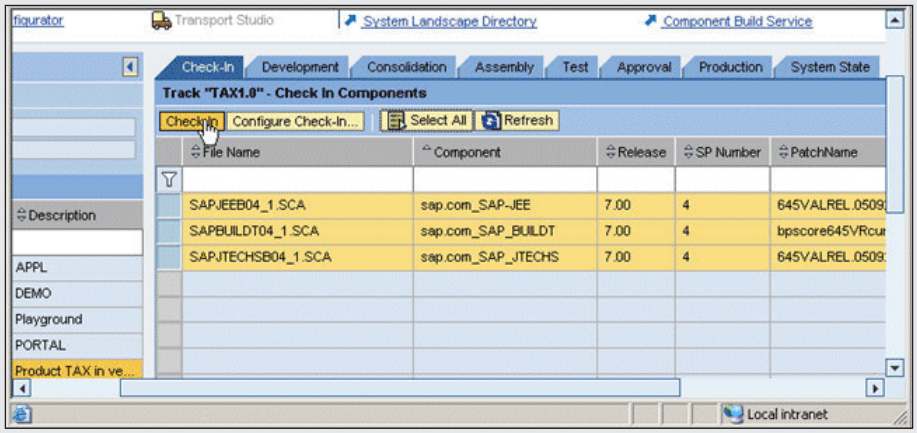

Figure 14 Checking in the components defined in the track

Note!

The view in **Figure 14** has been configured by the transport manager (by clicking on Configure Check-In) such that only the highest version of SCAs is shown; if you are basing your development on the latest version of the required software components, this setting helps make them easier to find. It is important that the versions of the required software components, the version of the target runtime system, and the version of NWDS all be the same to ensure compatibility. This is especially important in Web Dynpro development because frequent changes have been made to Web Dynpro-specific functions in NWDS, resulting in several corresponding version numbers.

Now just one step remains before the developers can take over: importing the existing SCAs into the runtime systems defined for the track. In the CMS Transport Studio, go to the corresponding tab for each system in the track where changes can be made (development and consolidation in the example), click on Select All to select the SCAs, and then click on Import. This step makes the needed archives physically available in each runtime system.

Note!

Figure 14 shows a full setup of runtime systems (development, consolidation, test, and production, as you can see on the tabs), so you can see how easy it is to manage a product with NWDI throughout its lifecycle. Remember that in the example, the Test and Production tabs would not appear since we did not select those runtime systems in the track definition.

We have now defined a new product and configured the two systems (development and consolidation) needed for the development lifecycle of our example Tax product. Again, remember that, contrary to most development systems in the ABAP world, both systems (development and consolidation) are "logical" systems that run on a single NWDI server — as do any other defined tracks.[20] This means that creating a new track does not require installing new systems.

[20] For large development teams, NWDI can be distributed on several servers. For details on hardware requirements, see SAP Note 737368.

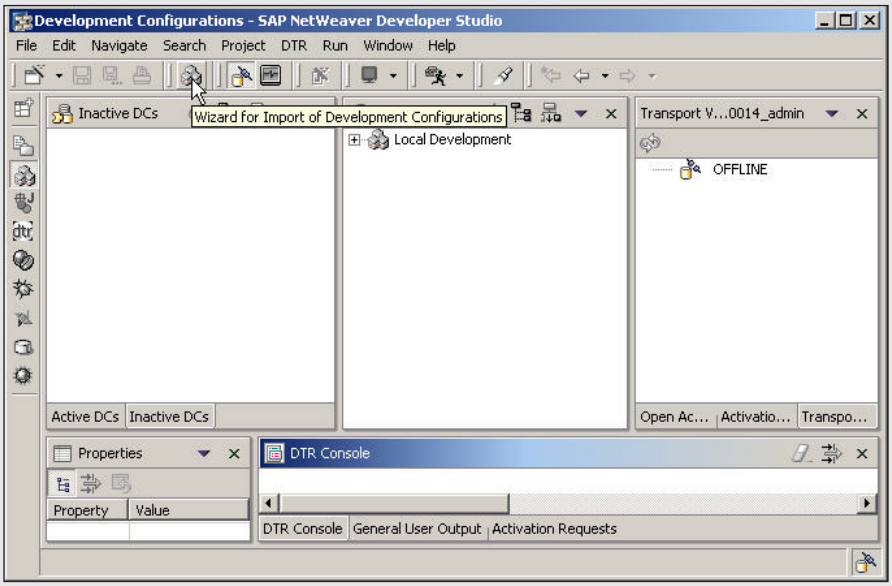

Figure 15 The Development Configurations perspective in NWDS

At this point we have completed the preparation and are ready to move on to the fun part — the development! Let's get started.

Step 5: Import the development configuration into NWDS

The first thing developers must do is import the automatically generated development configuration, which describes the product structure, the available software components, and the URLs of the servers that NWDS must access (the DTR, the CBS, etc.). To do this, developers simply need to be provided with the URL of the SLD,[21] the name of the track they'll be working with, and the overall architecture of the application (i.e., which existing software components are included, which new software components they should append development objects to, etc.).

Developers perform all of their tasks within NWDS, which is organized into different customizable "perspectives" containing task-specific views that can be configured within the perspective according to your needs (you can move the views around, close ones you don't use, etc.). The Development Configurations perspective shown in **Figure 15** is the central perspective where all development with NWDI begins.

Let's take a quick look at the default layout of the views in the Development Configurations perspective. On the left side is a view with tabs for displaying the list of active DCs (successfully built or "activated" objects, both those that can be modified and those that are read-only) and inactive DCs (only objects that can be modified). Both views provide sources and archives, if already available, of all development components. The inactive DCs view is where you create development components. On the right side are three views: Open Activities (activities containing objects you've checked out or newly created activities

[21] This is only necessary if it is the development team's first project using NWDI. The SLD URL does not change when new development configurations are created. The same SLD URL, as the URL for NWDI itself, will be used for all development projects.

waiting to be checked into the DTR), Activation (where you trigger the central build in the CBS of development components affected by the changes to be activated), and Transport (where you can release objects for transport via the CMS). As you can see, these are the interfaces that allow the developer to interact with the NWDI services (the DTR, CBS, and CMS), and that implement the SAP component model in the form of software components and development components.

In the middle is the Local DCs view, which displays all objects that you have synced to the local file system so that you can easily find your own development components among all the development components of a software component (remember that you can see the whole team's development components in the active and inactive DCs views).[22] On the lower part of the screen is the Properties view (where you can find detailed information on development objects, such as a class file of a public part), DTR console view (where you can read messages from the DTR server, such as sync result messages), General User Output view (where general server messages are shown, such as deployment success or failure messages), and the Activation Requests view (where you can monitor the central build of objects you have activated).

To import the Tax1.0_dev development configuration, log into NWDS as an NWDI developer,[23] open the Development Configurations perspective, and click on the wizard icon (see **Figure 15**). Choose "remote"[24] in the dialog that appears to read data

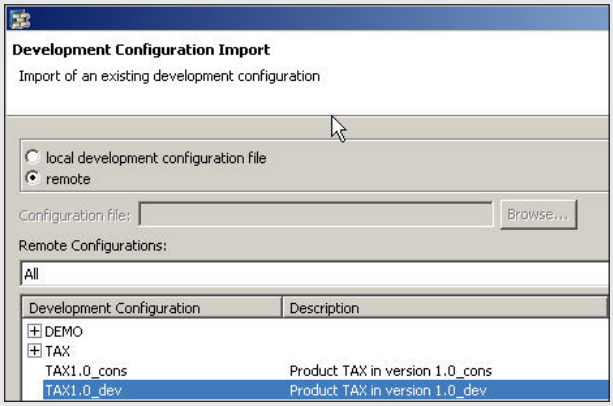

Figure 16 Select the development configuration for import

from the server (**Figure 16**), and select Tax1.0_dev from the list of available development configurations.

> ***Note!***
>
> As you can see in **Figure 16**, there are two development configurations — remember that for each track one development configuration is generated for the (logical) development system and another is generated for the (logical) consolidation system, because these systems have DTR workspaces and CBS buildspaces that NWDS must be made aware of. Here, we import the development configuration for the development system, because we are in the development phase. If we were working on consolidation — performing testing or making emergency corrections (see Step 11) — we would import the consolidation development configuration instead.

Click on Next, which takes you to the dialog shown in **Figure 17** (on the next page). Here, you can see information about the systems you will be

[22] "Local Development" is a kind of sandbox development configuration, somewhat comparable to objects created with $tmp in the ABAP system. Objects created here do not have a connection to a central NWDI and are not available to other developers. You can add local objects to the DTR to share them with other developers, but this requires some manual effort.

[23] As with the NWDI administrator role, the NWDI developer role is provided in the UME and will have the necessary permissions after the installation and configuration of NWDI.

[24] If you have a local copy of the development configuration file, you would choose "local" instead. This option was used in the very early stages of JDI when the CMS was not yet implemented, but it is seldom used in the current NWDI. However, it will be needed more often in some SAP NetWeaver CE 7.1 development scenarios — when you want to develop according to SAP's component model but do not have a full installation of the NWDI, for example.

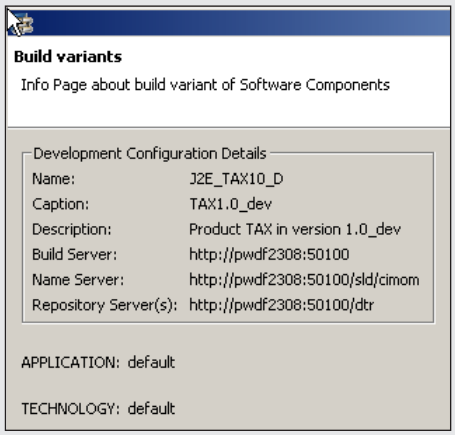

Figure 17 Information on the selected development configuration

> ***Note!***
>
> You are now connected to the development configuration TAX1.0_dev and therefore have access to the DTR workspaces and CBS buildspace that were automatically generated for the Tax development project when we defined the track. You can now simply create and edit files in the project, but you also have the benefit of seeing what other developers connected to this development configuration see. You share the same central systems for sources and archives.

connected to with this development configuration and which software components will be developed (APPLICATION and TECHNOLOGY).

Click on Finish, which will complete the import of the development configuration and return you to the Development Configurations perspective in NWDS. The Active DCs view should now show all the software components of the development configuration, including the components to be developed (APPLICATION and TECHNOLOGY) and the read-only SAP-provided components (SAP_JTECHS, SAP_BUILDT, and SAP-JEE), while the Inactive DCs view shows only those that can be changed (APPLICATION and TECHNOLOGY). All checked-in DCs from the entire development team are immediately accessible to the developer. **Figure 18** shows what these elements might look like in the Development Configurations perspective for the example (it also shows some development components that have been created for TECHNOLOGY, including tech/tax and tech/tax/calc; we'll look at how to create development components in the next step).[25]

[25] You'll notice that the views appear a little differently in **Figure 18** and **Figure 15**. Remember that the view display can be customized to your needs, like the one in **Figure 18**, as well as easily restored to the default layout shown in **Figure 15**.

Step 6: Develop new components

We're now ready to start developing the application. **Figure 19** shows a more detailed design of the example Tax application. As you can see, the application follows the design described in Step 1 (refer back to **Figure 3**): it is divided into two software components — TECHNOLOGY, which will contain the tax calculation functionality, and APPLICATION, which will contain the user interface elements. Both of these components will use items contained in the standard SAP components SAP-JEE, SAP_BUILDT, and SAP_JTECHS. Second, the TECHNOLOGY and APPLICATION software components will each be organized into two development components:

- TECHNOLOGY is made up of tech/tax/calc, which is a Java development component that contains the tax calculation functionality, and tech/tax, which is a J2EE library that is used to wrap the Java development component for deployment to the server, because Java archive (JAR) files, which are the build result of a Java development component, cannot be deployed to SAP NetWeaver directly.
- APPLICATION consists of appl/tax/servlet, which is a Web module development component

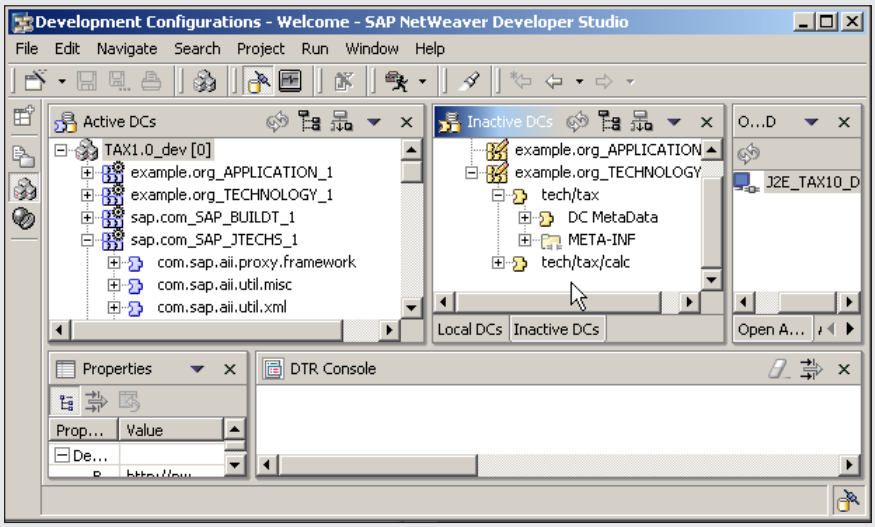

Figure 18 The imported TAX1.0_dev development configuration

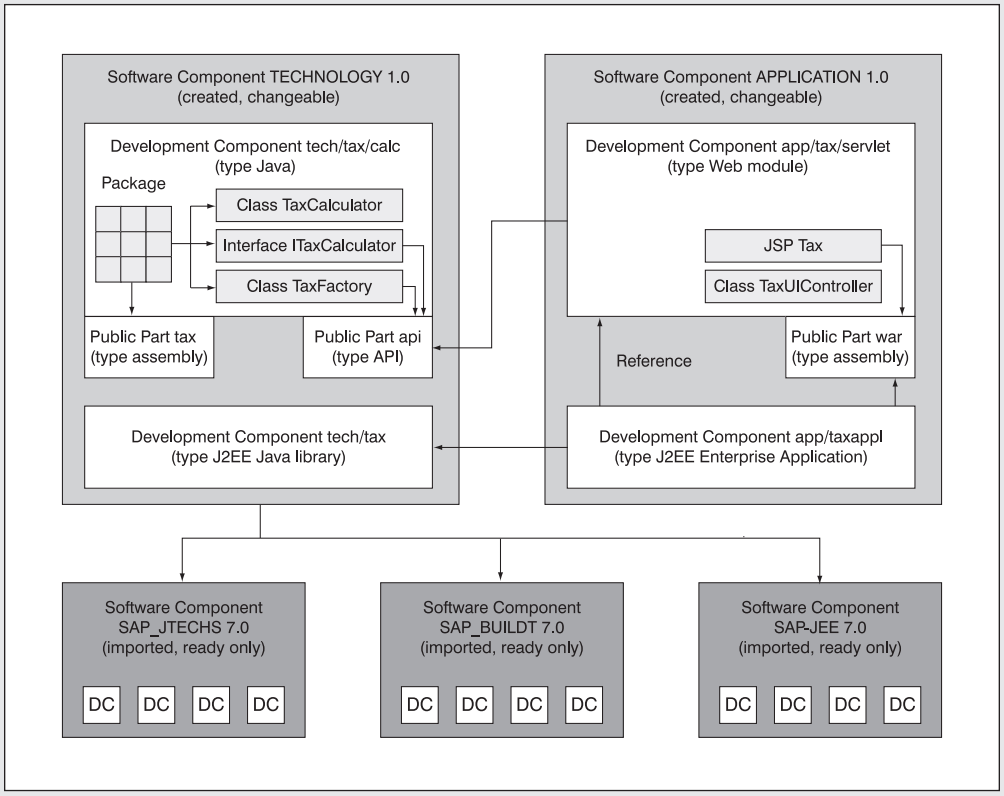

Figure 19 Structure of the example application

containing the user interface, and appl/taxappl, which is a J2EE Enterprise application that contains no coding of its own and is used to wrap the Web module development component for deployment purposes, because Web Archive (WAR) files, which are the build result of a Web module development component, also cannot be directly deployed to SAP NetWeaver.

Finally, the development components within TECHNOLOGY will contain custom Java classes and interfaces, and the APPLICATION component will contain a JSP and associated class for rendering the user interface.

Note!

As you will see, the actual development of technical objects with Java doesn't change significantly when using NWDI. For this reason, I don't go into the Java coding details in this step. Instead, I highlight the additional steps that are required for component-based development with NWDI, in particular defining the metadata for the interfaces (known as "public parts") a development component exposes to other development components, specifying the dependencies a development component has with other development components, and moving the development components along the path defined by the track (i.e., checking them in, activating them, and releasing them to consolidation). Complete details on all of the development steps and the coding are available at http://help.sap.com.

Let's now implement this design.

Create a new development component (DC)

We now need to create development components (DCs) to act as containers for the actual development objects (Java classes, JSPs, etc.) that constitute the application. These units are what you create in the Development Configurations perspective and serve as the build unit in the CBS. Remember that a DC is always assigned to a software component, so to create a new DC that will define the user interface for the Tax application, you right-click on the software component for which you want to create the new DC in the Inactive DCs view of NWDS (remember that only inactive software components can be modified) and select Create New DC. Let's assume that the DCs for TECHNOLOGY have already been created, so that the necessary tax calculation functionality is available for use in the development of the user interface, so here we create a new DC for the APPLICATION software component, as shown in **Figure 20**.

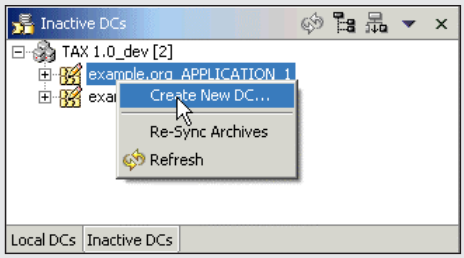

Figure 20 Creating a new DC within the selected software component

Define the DC

In the New Development Component project wizard that appears (**Figure 21**), you define a DC project. It is important that each DC contains its vendor flag (i.e., the domain of your company), so that you can differentiate your DCs from those delivered by SAP or developed by a third party. The name of the DC project defines the hierarchical structure of the folder that will contain the development objects in the DTR. The first part of the name ("appl" in the example) is the prefix defined by the administrator in the SLD

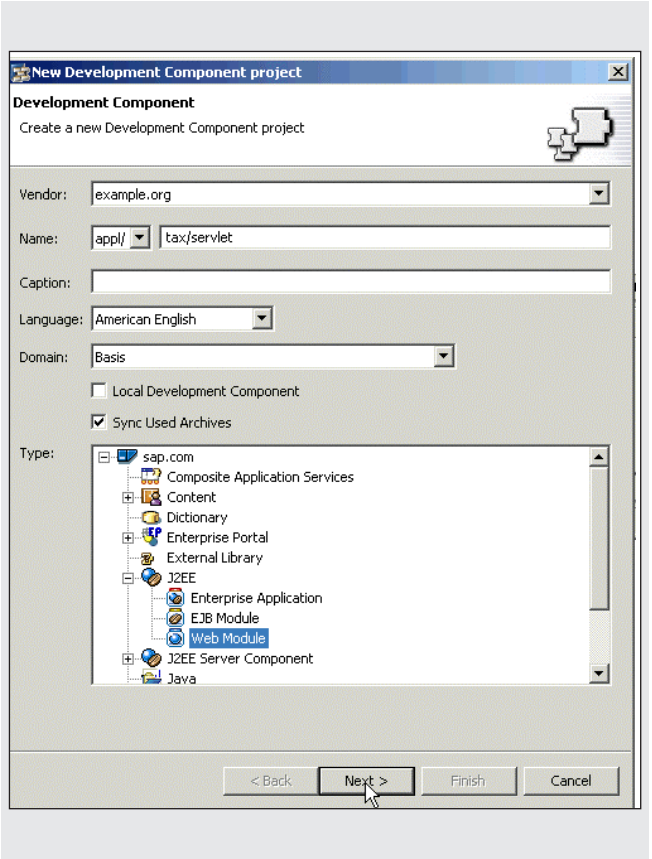

Figure 21 The DC project wizard

name service to organize software development.[26] The second part of the name ("tax/servlet" in the example) is defined by the developer. It can be up to 40 characters long (including the prefix) and should reflect the DC's function. You can also optionally provide a description of the DC in the Caption field and specify the language of the text elements of the project. Select "Sync Used Archives" so that all required archives — both those needed by default for a certain DC type (e.g., ejb20 for a Web module DC) and those for which a dependency has been defined — are retrieved from the NWDI server and synced to your local PC automatically. Finally, you need to specify the type of DC, which determines the build scripts to be used and the DC project's structure, both of which are created automatically. For the example, we select "Web Module" as the type.[27]

Click on Next, which takes you to the DTR Activity screen.

> ***Note!***
>
> Do not use the Local Development Component setting! This setting works like $tmp in ABAP systems, which means the DC will not be available in the NWDI server, and other developers will not have access to them.

Create a new DTR activity

Remember that the DTR coordinates storage and management of all development objects, so the system asks you to assign the sources you will create for development objects to a DTR activity. The activity makes the DTR aware of changes to the associated development objects and serves as the unit of transport for checking changes into and out of the DTR. Click on New Activity, which takes you to the screen shown in **Figure 22** (on the next page).

> ***Note!***
>
> It's a good idea to use an intuitive name for the activity, so that you can easily group any future changes to this DC project under the correct activity.[28]

[26] There should be only a very limited number of prefixes, to avoid a cumbersome and potentially confusing drop-down list of hundreds of prefixes.

[27] Our simple example application will be a JSP in a Web module. A bigger application would probably be Web Dynpro-based, which is also an option in the DC wizard.

[28] The name need not be unique, by the way. Every activity is automatically assigned a GUID that makes it globally unique.

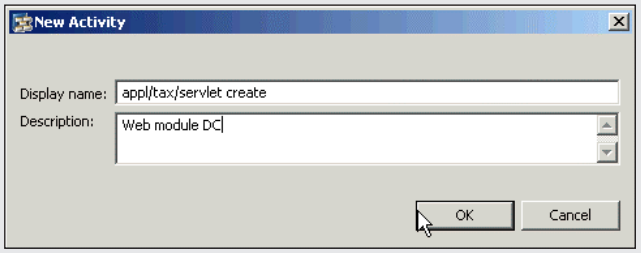

Figure 22 Creating an activity

Generate the DC structure

After creating the activity and clicking on Finish in the DC project wizard, the wizard generates the newly created Web module DC and default structure, retrieves the required archives for this type of DC, syncs them to your PC, and displays the objects in the Local DCs view, as shown in **Figure 23**. The objects will also be displayed in the Inactive DCs view of the PCs of any other developers using this development configuration (in this case, the developers will need to refresh the view to see the newly added objects).

> ***Note!***
>
> The Local DCs view helps you keep track of your tasks. Only those DCs that are synced (as a result of selecting "Sync Used Archives" in **Figure 21**) from the NWDI server — specifically, from the DTR (for sources) and the CBS (for archives) — to your file system are shown here.

As you can see, the default structure of the newly created DC (example.org/appl/tax/servlet) appears under the APPLICATION software component of the TAX1.0_dev development configuration. The DC MetaData folder, which is a part of all DCs, contains the definition information of the DC, including the public parts or interfaces ("war" in the example) of

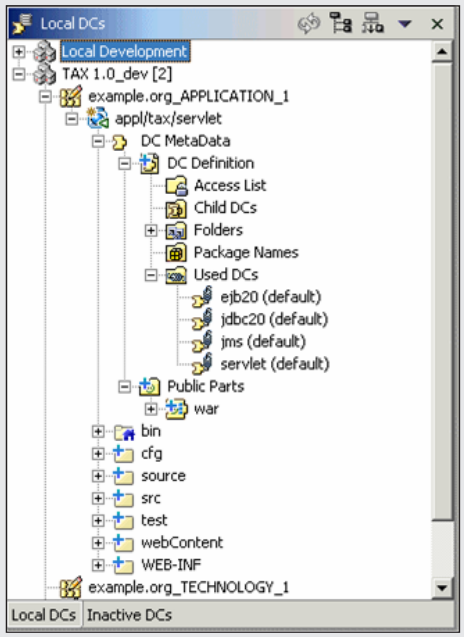

Figure 23 The newly created DC structure

the DC and a list of the required DCs (ejb20, servlet, etc.). Since the DC is a Web module, the DC MetaData folder also includes a webContent folder, in which the JSP file for the user interface will later be created.

> ***Note!***
>
> The icons in the structure list indicate the state of the objects. The plus sign shown on various icons indicates the objects are *checked-out for create* (i.e., these objects are under development), the little blue house shown on the icon next to the "bin" folder means *local only* (i.e., these objects are not going to be checked in, because they are either generated or runtime objects), and the asterisk shown on the icon for the DC itself indicates that the project has changed since the last local build.

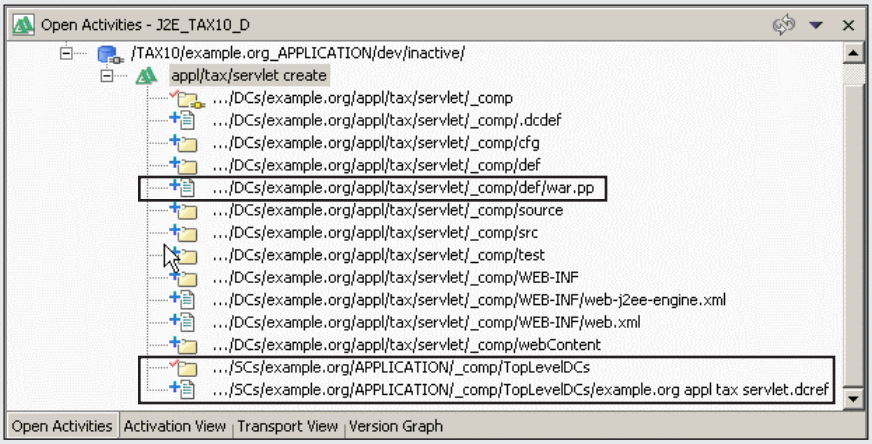

Figure 24 Open activity content in the context of the inactive workspace of the Tax 1.0 track

Figure 24 shows what you would see if you opened the DTR activity (appl/tax/servlet create) created for the DC in the Open Activities view of the Development Configurations perspective in NWDS.[29] As you can see, all objects displayed in the DC structure are also displayed in the activity created for the project (with the exception of some generated objects, such as the .project and the .classpath files, which are not added to the activity). For example, you see the public part file (war.pp). In addition to the objects included in the DC structure, the information on the software component to which the DC is assigned has been created and included in the activity, as shown at the bottom of the figure. Any future changes to the software component are added to the open activity automatically. The green color of the icon next to the DTR activity name indicates that the activity is still open (i.e., the activity can still be changed by adding or deleting objects, or it can even be deleted). The fact that the activity is open also means that its content is not yet available to other developers on the central NWDI server.

[29] You might be wondering why the name J2E_TAX10_D appears at the top of the Open Activities view. Activities are displayed in the context of the associated track and DTR workspace. Remember that "J2E" is the domain name of the CMS that was used to create the track and "TAX10_D" is the track name in the development phase (that's what "D" stands for).

Create the development objects

Since a DC project is only a container for the development objects, next you need to create the development objects themselves. For the example Web module project, we will create a servlet containing a class that controls the return of tax calculation values in the application and a JSP that provides the user interface.

Once you have created a new DC project, NWDS automatically switches to the perspective necessary to work on the development objects within that DC project — for the example Web module project, it switches to the J2EE Development perspective. To create a new development object, you simply navigate to the DC project name in the J2EE DC Explorer and right-click on it (note that in the J2EE DC Explorer, the project name includes the CMS and vendor domain name). **Figure 25** (on the next page) shows the example DC project's context menu, from which you can create both the JSP and the servlet required for APPLICATION.

Creating development objects in NWDS is no different from creating them in other IDEs, so I won't go into the details of coding the JSP and servlet for the example. Let's, however, take a look at the dependencies involved because these are not typical to "traditional" Java coding. As shown in **Figure 25**,

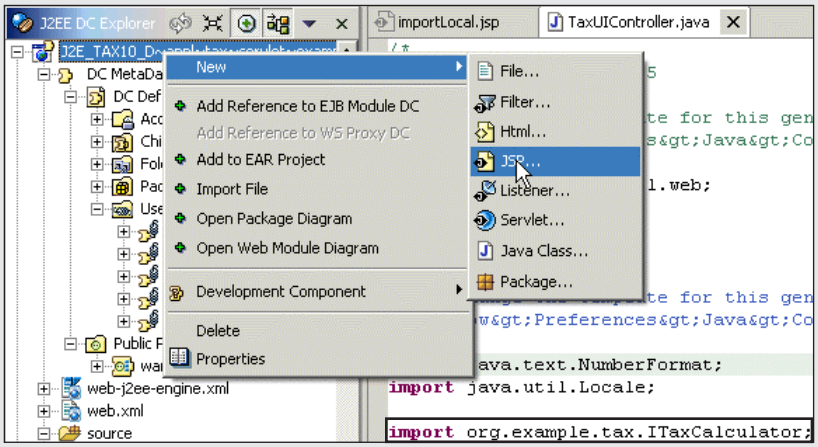

Figure 25 Adding a development object

TaxUIController has been created as part of the servlet to handle the tax calculator. TaxUIController is also used by the JSP, but since all of these objects are part of the same DC, no use dependency is required.

As you can see in **Figure 25**, TaxUIController imports the ITaxCalculator interface, which is a standard Java interface, from one of the DCs of TECHNOLOGY. This import will cause an error in NWDS, as shown in **Figure 26**, because objects of other DCs may only be used if a dependency is explicitly declared. To do this, you must create a public part in the used DC to allow dependencies between the DCs. I show you how to do this next.

> ***Note!***
>
> Do not confuse a Java interface with a DC's public part! Java interfaces are standard Java functionality. To make any object of a Java DC available to other DCs, the object — in this case, the Java interface — still needs to be added to the public part. This does not mean that you are doing the same thing twice. Filling the metadata of a DC (e.g., its public parts and public part entities) is the basis for the incremental build of the components and the automatic control of the buildspace state by the CBS.

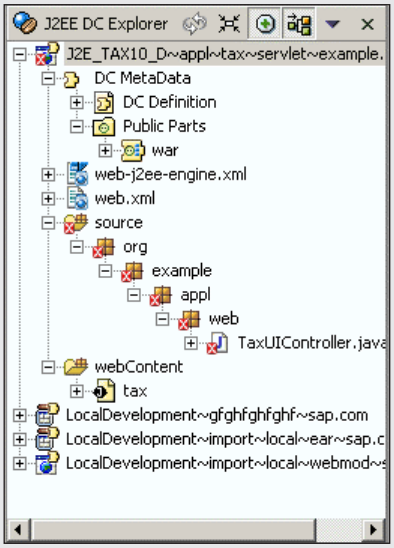

Figure 26 Errors caused by missing use dependency

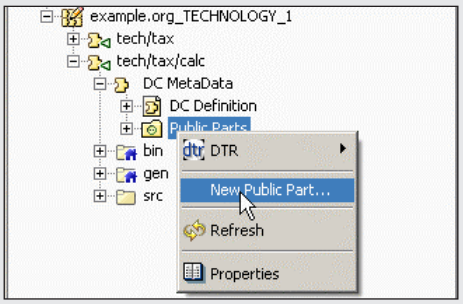

Figure 27 Creating a public part

Create an API public part to enable DCs to share functionality

Public parts enable development components to use one another. A public part is a DC's interface through which a development component accesses the functionality of another development component. To reuse the Java interface of the tax calculation DC (ITaxCalculator) in the user interface, we need to create a public part in TECHNOLOGY, into which we will place ITaxCalculator to make it accessible.

To create a new public part, simply go to any DC editor view in NWDS (e.g., the Inactive DCs view, or the overall DC view of the project, such as the J2EE DC Explorer view), navigate to the DC MetaData folder for TECHNOLOGY, right-click on the Public Parts folder, and select New Public Parts, as shown in **Figure 27**.[30]

Next we have to define the type of the public part — whether it allows the content of a DC to be used in other DCs (API), or whether it allows the content of a DC to be packaged together with another for deployment (assembly). As shown in **Figure 28**, we defined the public part as an API whose entities will be used via an ordinary "import" command in the user interface of our example tax application.

We then need to add the necessary development objects ("entities") to the public part to make them

[30] Public parts are either created using the context menu of the Public Parts folder in the DC MetaData folder, or automatically generated by the system, as in the Web module DC. In any case, the development objects — the "entities" — need to be added to the public part to make them available for use by other DCs.

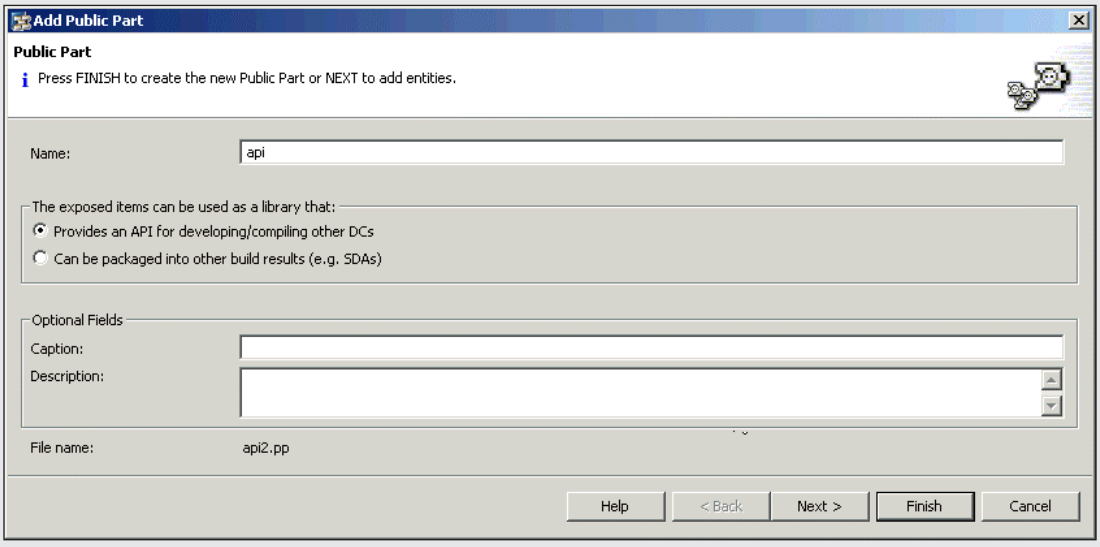

Figure 28 Defining a public part

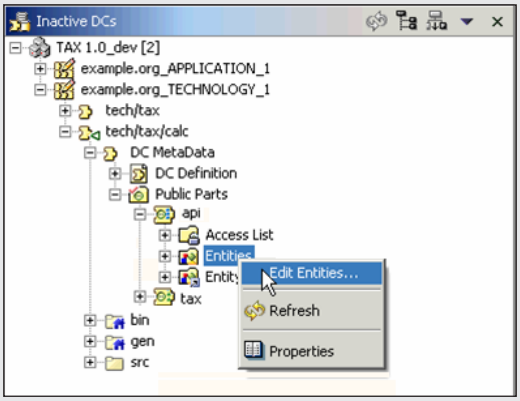

Figure 29 Adding entities to a public part

available for use in other DCs. Expand the newly created "api" public part folder, right-click on Entities, and select Edit Entities, as shown in **Figure 29**.

All objects you created for the DC are displayed in a list you can choose from, as shown in **Figure 30**. You can add them one by one as single files or you can add them as complete packages or package trees. In the case of our example application, we add ITaxCalculator (the Java interface of the tax calculation DC) to the "api" public part, and we also add TaxFactory, which provides new instances of the tax calculation (we do not select TaxCalculator, because only the calculation results are needed, not the calculator itself). We add these two entities as classes rather than as sources because we need the compiled version in order to use the function.

Click on Finish to create the "api" public part. The result is displayed in **Figure 31**.

Create an assembly public part to wrap a Java DC for deployment

There is one more public part we need to create. Remember that the tax calculation functionality is a Java DC. Since a Java DC cannot be deployed directly to the J2EE engine — SAP NetWeaver does not allow direct deployment of a JAR file — we also need to create a second public part of type assembly for the

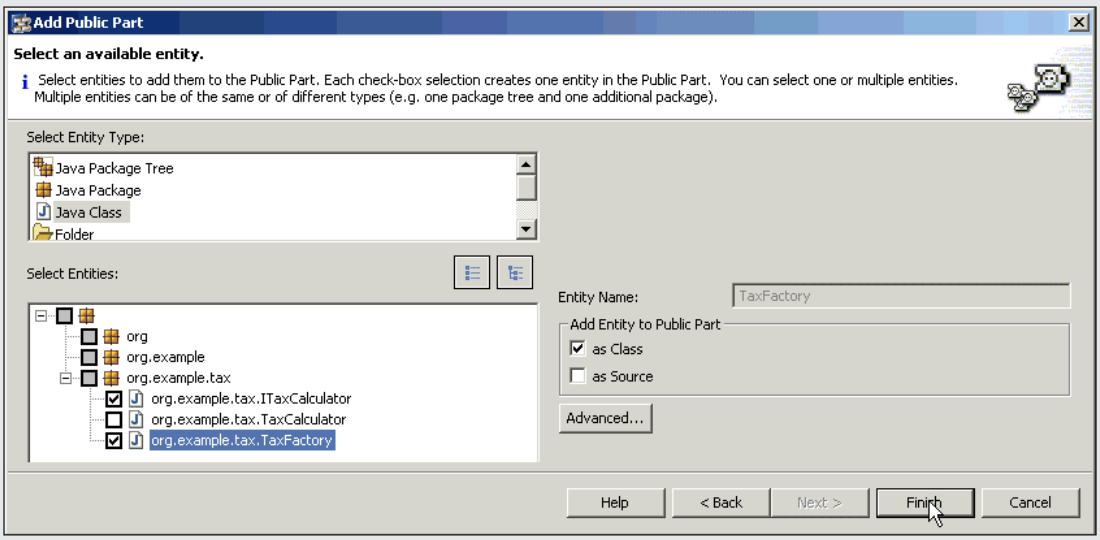

Figure 30 Add objects to the public part

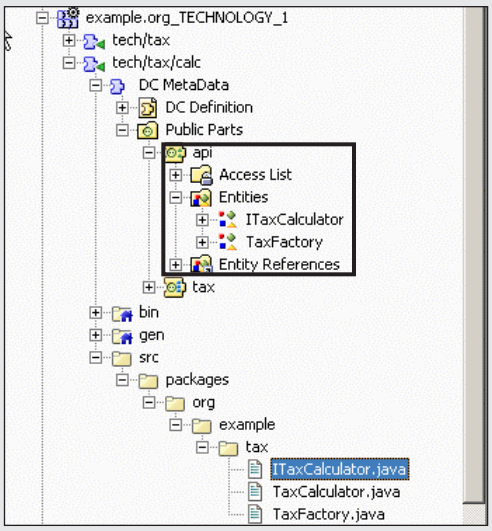

Figure 31 The newly created “api” public part

tax calculation DC. A public part of type assembly allows you to package the build result of a DC for deployment as a ZIP file. Follow the same steps as for creating the “api” public part, only name the public part “tax” in this case and specify that the public part allows the content of a DC to be packaged together with another (assembly). For the content of this public part, we need to add all packages of the DC project, so on the Add Public Part screen select the entire package tree. **Figure 31** shows the “tax” public part added to the DC MetaData folder of tech/tax/calc and the entities it exposes.

> ***Note!***
>
> Making objects available for assembly does not allow their import into another DC, so in our case both types of public parts (API and assembly) are needed. For a DC with a deployable build result (such as an Enterprise Application DC) the assembly part would not be needed.

Create a use dependency between the dependent DC and a target DC’s public part

Next we need to add a use dependency between the servlet DC and the “api” public part of the tax calculator DC to make the calculation functions available for the user interface. This dependency will be checked in the CBS build process (remember that we specified that dependencies be checked at build time), ensuring that the structure of the entire application follows the SAP component model (i.e., only objects in public parts are used in other DCs, and all used objects are in public parts of the correct type). This information will be used by the CBS to organize the incremental build of DCs triggered by the developer, including keeping all archives of the buildspace up-to-date when changes are activated by an internal rebuild of all DCs that use the public parts of the changed one.

Use dependencies are added in the DC MetaData folder of the dependent DC — in the example, the servlet in the user interface part depends on the calculator functions in order to allow the input of numbers and present the result to the user. To create a use dependency, simply go to any DC editor view in NWDS (e.g., the Inactive DCs view, or the overall DC view of the project, such as the J2EE DC Explorer view), navigate to the DC MetaData folder for APPLICATION, right-click on Used DCs under DC Definition, and select Add Used DC, as shown in **Figure 32**.

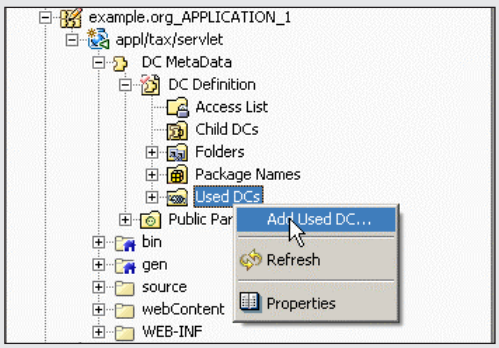

Figure 32 Adding a use dependency

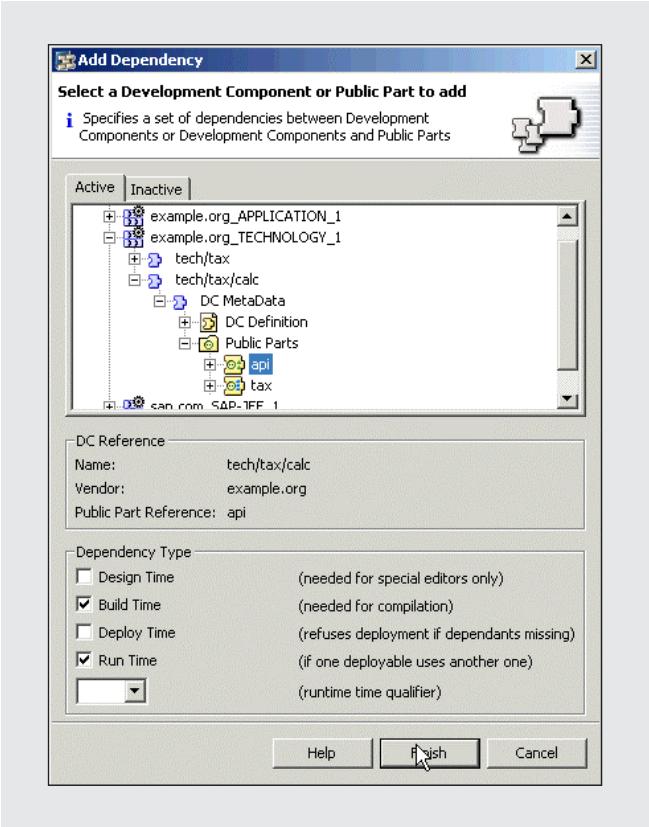

Figure 33 Use dependency definition

In the screen that appears (shown in **Figure 33**), navigate to and select the target interface for the dependency (in the example, public part "api" in the tech/tax/calc DC of the TECHNOLGY software component) and define the dependency type. In the example, we select Build Time and Run Time for the dependency type — you have to allow the use of the calculator functions when the user interface is compiled (Build Time), and you certainly want to allow its use when running the application (Run Time).[31] Once the dependency is declared, the errors shown in **Figure 26** will be gone.

[31] Design-time dependencies are only used for special tasks, whereas deploy-time dependencies ensure that an object has been deployed with dependent objects. With deploy-time dependencies, take care that Java DCs and Web module DCs are not deployed by themselves. In the case of these DC types, a deploy-time dependency will cause errors. The system checks that, for example, a Java DC was deployed when the Web module DC was deployed using its public part; since no Java DC is deployed directly, this must fail.

Create a deployable unit for a Web module DC or a Java DC

Since the build results of Web module DCs (WAR files) and Java DCs (JAR files) cannot be directly deployed to SAP NetWeaver, you need to wrap these types of DCs in a deployable unit. Let's assume that the Java DC containing the tax calculation functionality has already been wrapped in a J2EE library DC. To also bring the Web module DC into the context of the Tax application, we have to wrap it in an Enterprise Application DC. Luckily, in a Web module, JSP files and servlets are added by default to the automatically generated "war" public part, which is of type assembly. So the last thing we need to do is to create the Enterprise Application DC, which will use the public part "war" of the Web module to access the files that need to be wrapped. Creating the Enterprise Application DC is essentially the same as creating the Web module DC, so I won't go into the details. You simply go to the Development Configurations perspective in NWDS, right-click on APPLICATION (so that this software component contains the new DC), select Create New DC, define the new DC as an Enterprise Application, and create a corresponding activity (in the example, appl/tax/servlet create).

We now have all the "pieces" we need for the Tax application sketched in **Figure 19** — i.e., the tax calculation functions of TECHNOLOGY and the user interface parts of APPLICATION. The calculation was created as a Java DC and packaged in a J2EE library. The user interface was created as a JSP and a servlet class that uses the calculation functions of TECHNOLOGY and packaged as an Enterprise Application. The use of the DCs is organized by the public part types provided by the DCs — API public parts provide an interface for using one DC's functionality in another and assembly public parts enable the packaging of functions for deployment — and by the use dependencies defined between the DCs. This structure defines the build process.

With all the pieces of the application in hand, we're ready for the intial build and testing of the application's components.

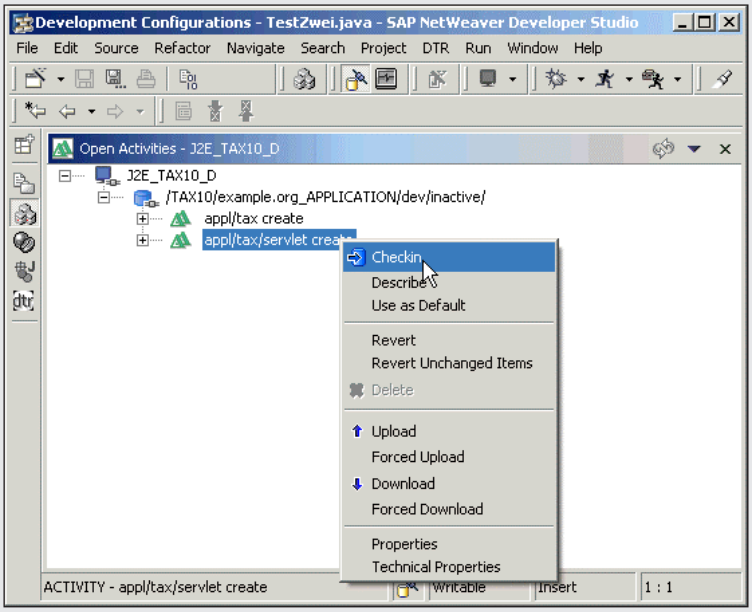

Figure 34 Check in the activity assigned to your development work

Step 7: Locally build and test the components

Next, the developer initiates a local build within NWDS. The resulting archives are stored locally, and the developer is advised of any errors that occur during the build process. The developer deploys the archives to the local J2EE engine and debugs the code locally, if SAP NetWeaver Developer Workplace is installed (NWDS with a local J2EE engine); if NWDS is used without an installed J2EE engine, then any external engine can be used.

Since this process is identical to traditional Java development/debugging, let's move on and find out what happens next.

Step 8: Check in and activate the components

Now it's time for the developer to check in the activities created for the DCs. Remember that the DTR manages changes to all NWDI development objects, and therefore needs to store development work done on the objects in the DCs via activities.

In NWDS, go to the Development Configurations perspective. The Open Activities view enables you to interact directly with the DTR. To check in an activity, you simply right-click on the activity and click on Checkin. In **Figure 34**, we select the activity for the Enterprise Application that is serving as a wrapper for the Web module DC of APPLICATION.

On the next screen that appears, click on OK, which brings you to an Activation screen, shown in **Figure 35** on the next page. As you can see, the activity for the Web module DC (appl/tax create) has already been checked in as well. To share our DCs with other developers, we need to activate the selected activities, which triggers the central build of all the DCs in the CBS that have been affected by the changes in our activities, and makes the "inactive" DCs become "active." Click on the Activate button. The CBS automatically collects the checked-in source code and active versions of any dependent components (based on the use dependency we defined

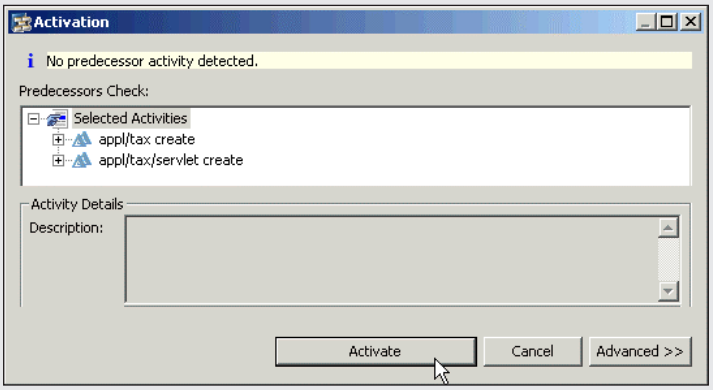

Figure 35 Activate closed DCs

between the Web module DC and the Java DC's "api" public part), and builds them into an executable archive. The checked-in sources are now available at the central NWDI server and are no longer writeable.[32]

If the activation is successful, several things happen — the sources are integrated into the active DTR workspace, the buildspace is updated, and the runtime objects resulting from the build are deployed into the development runtime system (if a central test system for development was defined for this phase). If you have built and tested the components locally, activation will almost always be successful. One of the only reasons for a failed build would be an incompatible change to a used DC's public part — for example, if the owner of the Java DC replaced the public part "api" with another, your build would fail, and the build log would indicate the reason. In this case, you would simply check out the Web module DC (the latest version of the Java DC would be synced from the server), replace the dependency, and then check in and activate the DC again.

Once both the APPLICATION and TECHNOLOGY activities are checked in and activated, we're ready to retest the components on the central test system and release the activities to the next system in the track.

Step 9: Retest the components centrally and release the activities

After a successful central test, the developer can release the activities for transport to the next logical system defined in the track (the consolidation system). To do this, switch to the Transport View in NWDS, shown in **Figure 36**. Let's assume that the TECHNOLOGY activities have already been released (these must be released first, since they contain objects required for the build of the APPLICATION components in the CBS). So we now need to release the APPLICATION activities. As you can see, there are two activities waiting to be released under APPLICATION. Select both activities, right-click, and then select Release from the context menu. When an activity is released, it is automatically added to the CMS import queue of the consolidation system.

With the release of activities by all team members, all DCs of the Tax application should now be available for import into the consolidation system.

[32] To back up files on the server without checking them in, a developer can use the Upload function available on the same context menu as Checkin. In this case, the activity would remain open and other developers would not see an unfinished version on the server; however, after a local crash, the uploaded content could be retrieved from the server.

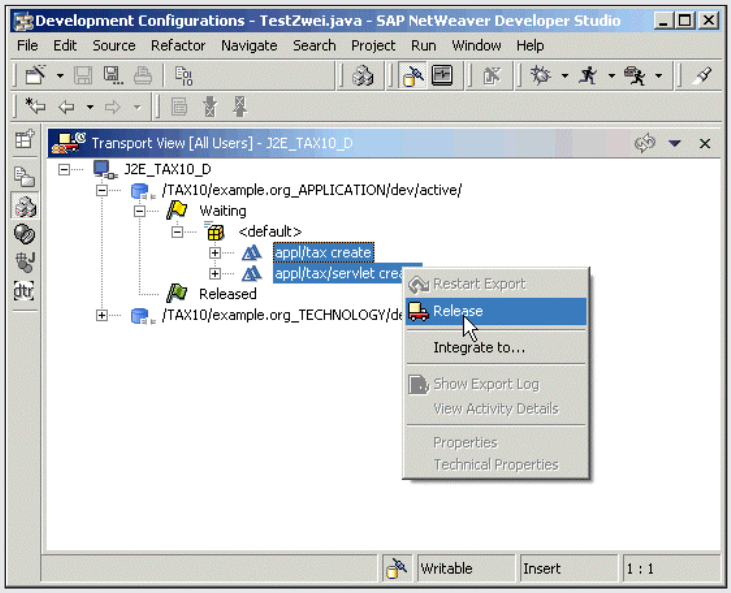

Figure 36 Releasing activities for the Tax application

At this point, the development work is complete, unless there are any corrections that need to be made after testing, and we are ready to import the DCs into the consolidation system for testing.

Step 10: Import the components into the consolidation system

In this step, the transport manager imports the changes into the consolidation system. First, however, it's a good idea to check the state of the central CBS buildspaces to make sure no errors were introduced due to an automatic rebuild of a dependent DC (the developer would have been notified of any other types of errors during activation). To view the central buildspaces, go to http://<NWDI-host:port>/devinf, which displays the CBS Buildspace Information page shown in **Figure 37** (on the next page). As you can see, buildspaces are listed for the development phase and the consolidation phase of each track (remember that only systems that allow code changes have a buildspace). Using the information on this screen, you can see if any problems occurred during the build. For example, the Broken DCs column for the WDTEST1 D buildspace shows that three DCs are "broken" and cannot build correctly. You can identify — and notify, if necessary — the appropriate developer by looking at the details for the buildspace. Once any issues are resolved, import the DCs by logging onto the CMS Transport Studio as an NWDI administrator and navigating to the consolidation system by clicking on the Consolidation tab (see **Figure 38** on the next page).

Let's assume that the TECHNOLOGY DCs have been imported, since TECHNOLOGY is a prerequisite for APPLICATION.Next we need to import the APPLICATION DCs. Select the object containing the list of activities for the DCs you want to import (i.e., the "propagation list") — in the example, example.org.APPLICATION — and then click on Import to import the DCs into the consolidation system. This triggers the build of the DCs for the buildspace of the consolidation system and automatic deployment into the assigned runtime system.

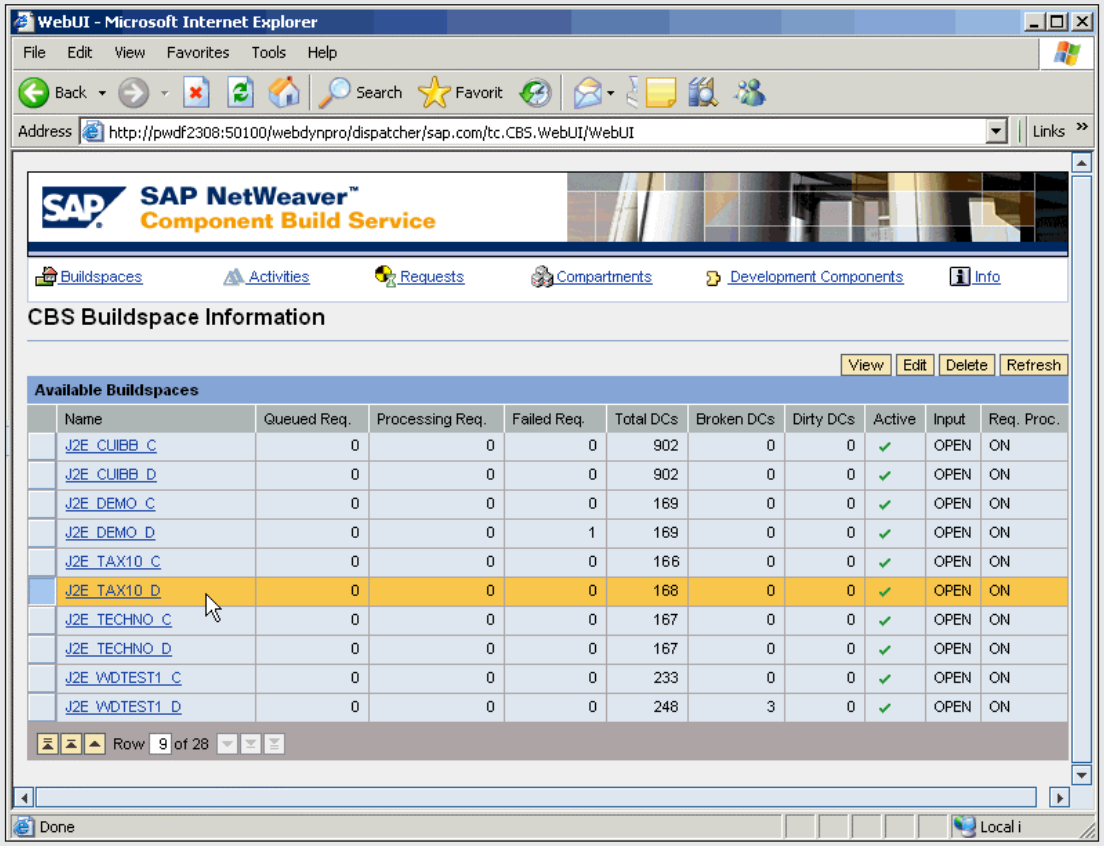

Figure 37 CBS buildspace information

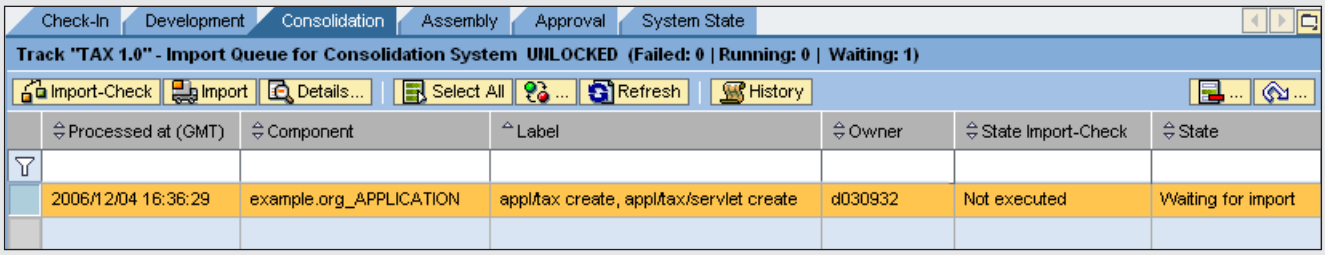

Figure 38 CMS Transport Studio

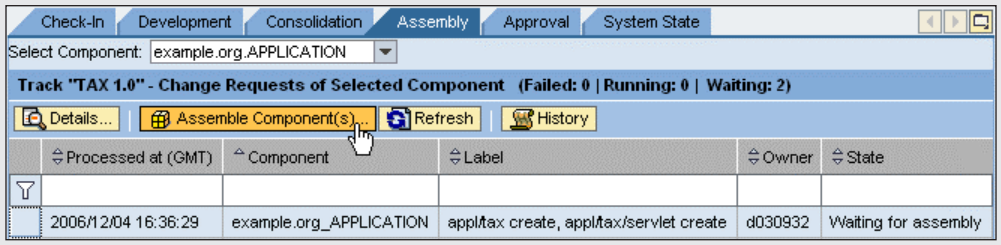

Figure 39 Assembly of software component archives

Now, the sources are retrieved from the active workspace of the development system and built against the used archives of the consolidation system. If successful, the sources are activated and the resulting archives are deployed to the central test server.

> ***Note!***
>
> Activation is repeated in the consolidation system to guarantee that the states of the development and consolidation systems remain isolated. Developers can change the development state system without interfering with test activities in the consolidation system, which will only change after the next import triggered by the transport manager. For this reason, another activation/build is performed as part of the import process.

Step 11: Test and request any necessary changes

The quality manager tests the application in the consolidation system. Any problems are corrected by the responsible developer — remember that in order to perform any consolidation system development work, the developer first needs to import the consolidation development configuration (TAX1.0_cons in **Figure 16**) into NWDS — and the archives are then redeployed and retested. Keep in mind that changes would only be made in this system in case of an emergency — if the project builds, but the tax calculation returns incorrect results, for example — since the consolidation system should remain the second system for testing purposes, not the first. Technically, you would make changes just as would during development.

Step 12: Assemble the software component archives (SCAs)

Once testing has been completed, the transport manager packages (assembles) the application into SCAs for delivery to (final) test and production.

Again, let's assume that the TECHNOLOGY components have already been assembled, since TECHNOLOGY is a prerequisite for APPLICATION. To then assemble the APPLICATION components, go to the Assembly tab in the CMS Transport Studio, select the APPLICATION software component from the Select Component drop-down at the top of the tab, and then click on the Assemble Components button, as shown in **Figure 39**. In addition to choosing a specific software component for individual assembly, you could also select all of the software components

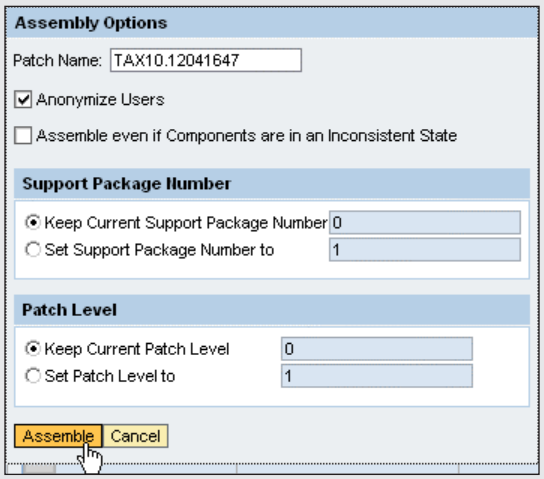

Figure 40 Defining support package numbers and patch levels during assembly

in the track for simultaneous assembly by selecting All Components in the Select Component drop-down list.

On the screen that appears (**Figure 40**), we need to define a support package number and patch level to capture the state of the software components we are developing. This information is written to the software component's metadata so that it can be applied to other systems or tracks when the transport manager triggers the assembly by clicking on Assemble. It is the information that appears in the CMS Transport Studio when you check in a software component to the CBS (refer back to **Figure 14**). As a rule-of-thumb, a new support package should be chosen if you want to provide a significant number of patches (this is used by SAP for shipping updates to customers, for example), while a patch would be suitable for just a few changes.

Once the SCAs are assembled, the archive files are automatically added to the test system import queue, if a test system is specified as part of the track. The build result — the SCA file — can now be used in any runtime system of the corresponding release and can also be used as the input for any follow-up track (e.g., a maintenance track) in the future.

Step 13: Import the archive files into the test system

It's a good idea to import the objects into a final test system that closely matches the state of your production system as a final step before going live with an activity.

> ***Note!***
>
> The key difference between import into a test system and import into a consolidation system is that in the test system, the deployment works the same way it works in the production system (based on the SCAs instead of DCs and without using a central build again), giving you the best test possible before going live or shipping a product.

The procedure is identical to the import of the development components into the consolidation system — only in this case, no build starts — and

the SCAs are deployed to the test runtime system (if one is available for this phase).

Step 14: Test and approve

In this step, the quality manager performs final testing of the application, if a test system is used. Once the final testing is successful, the archive files are automatically added to the production system import queue, if a production system is specified as part of the track.

Step 15: Import the archive files into the production system

Once the application is successfully tested and approved, the transport manager imports the application archives into the production system, if a production system is used.

Step 16: Define a new track for the next release

The final step is for the NWDI administrator to define a new track for subsequent fixes or development work. If this work represents major enhancement, you might want to first define a new product version as well, starting the whole process over again.

Here is a useful tip to save time when maintaining subsequent releases of a product: Rather than creating the track from scratch, create a copy of the existing track.[33] With the copied tracks, both the predefined software components *and* the software components for development will be checked in and imported: development starts with the versions (sources and archives) released with the "parent" track. You then update the used archives in the track with the versions to be used for the next release. The benefit is that developers will go on with their work almost without any interruption — they only need to reimport their development configuration.

[33] You use one track for each state of a product (or, to be precise, a set of software components, which will in most cases form a product) that you are going to support. This might be done on a release level or even on a support package level. If a product is under maintenance and under ongoing development, you should be able to patch the productive state any time. For this scenario, please refer to the SDN blog "Track Design for Ongoing Development" (see the note at the end of the article for information on how to find NWDI in SDN).

Conclusion

You have now seen all the development steps — from creating a new product, ready to be delivered, to beginning the maintenance cycle, and everything in between — that together constitute a real process for the entire development lifecycle: a number of well-defined steps that result in a predictable result. You have also seen that the structure of a new product can be defined in a new and efficient way as a set of reusable and manageable components.

You have also seen that by defining dependencies between development components, you can use a completely new build process and archive management on a component basis, which together with the source code management in workspaces guarantees a consistent development environment. On a software component level it guarantees that any development landscape setup can easily be replicated — for maintenance of a custom-developed product or to modify an SAP-delivered product, for example.

With the comprehensive software logistics — including automated transport and deployment into runtime systems — NWDI puts you in control of your software development projects. Furthermore, these consistent software states can be transported into other NWDI instances. Any track can be set up identically at different locations because it is simply described by a set of well-defined software components with known dependencies. Even conflicts that arise during support package import due to foreign versions caused by modifications are taken care of: Since versioning information is transported with sources, changes transported from one DTR to

another integrate correctly, ensuring safe handling of the delta between two tracks — including conflict detection and resolution. The practical outcome is a comfortable development and modification concept for Java software development lifecycle management that is being successfully used by customers, partners, and SAP alike. To give just one example, NWDI is used by customers to enhance and tailor the e-commerce capabilities in SAP CRM 5.0 and SAP ERP 6.0 to meet customer needs — to adapt the Web shop user interface to a pixel-perfect, specified design, for instance. Such scenarios require the unique capabilities of NWDI.

Note!

You'll find useful information on NWDI at the SDN Web site (open http://sdn.sap.com, and under SAP NetWeaver Technolgy Map look for Life-Cycle Management → Software Logistics → Software Change Management to find the SAP NetWeaver Development Infrastructure pages), which contains in-depth articles, a forum on NWDI, a link to the official documentation, demo videos on the NWDI-based development process, and even a link to download a trial version of NWDI.

