NetWeaver Development Infrastructure (NWDI)

Introduction to NWDI

Motivation

The SAP component model for Java was created to do:

- Increase the productivity of Software developers by:
  - Removing the need to write build scripts. Instead, the build scripts are created automatically based on the component type. (Manually written build scripts are still possible as exception)
  - Providing a central build service that permanently offers a consistent pool of up-to-date libraries and other build results, making it easy to work on consistent and up-to-date libraries.
  - Removing the need to know details about the development landscape, repository locations, library versions by encapsulating this information in "development configurations". Similar to the selection of an R/3 development system, developers can select the environment (defined by the "development configuration") in which they want to work.
  - Supporting an automated deployment of the (re)built results into central test systems. This allows early integration tests during development, reducing the effort for later integration tests drastically.
- Support developers by providing specialized tools and an automated workflow so that developers can concentrate on development without having to worry about delivery, deployment, versioning etc.
- Improve management of software dependencies
- Increase reusability and reuse of code segments by encapsulating functionality into small, reusable building blocks.

To achieve these goals the development of a software product was split into software components and development components.

Development Components

Development components (DCs) are the reusable building blocks for software components. A DC is a named container for arbitrary objects, e.g. Java source files, JSPs, dictionary definitions, deployment descriptors, etc. These contained objects are not directly visible to other components, i.e. the DC acts as a black box. You have to explicitly define "public parts" if you want some of these objects to be visible from the outside. A public part has a name and a purpose and contains a list of objects called entities.

The purpose of a public part is either compilation or assembly. Compilation means that other DCs can compile against the content of the public part, i.e. the public part defines the API of the DC. In the figure above, class X is contained in public part P1 with purpose compilation, therefore it can be referenced by classes in component A and the compiler will find class X, while it would not find class Y which is not exposed in a public part.

Purpose assembly means that other DCs can use the public part to assemble larger objects. E.g. a public part containing EJB classes can be used by an EJB Assembly project to assemble a jar file that can be used by an Enterprise Application project to assemble the final ear-file for deployment.

If a DC wants to use another DC it must explicitly declare this dependency. Dependencies are strictly hierarchical, dependency cycles are not allowed (as they would break the automated build).

DCs can also restrict their usage by defining an Access Control List (ACL), only allowing specific other DCs access to their public parts. An access control lists is either empty or contains a list of "Grants", i.e. a list of other DCs that are allowed access. If no grants are listed, any DC has access.

A DC can also contain other DCs. Contained DCs belong exclusively to their parent DC and are not visible to the outside at all. This can be used to encapsulate certain functionality without exposing any interfaces. Nevertheless the parent DC can expose objects that are visible in the public parts of the child DCs by including an "entity reference" to the inner DCs public part in its own public part.

A development component belongs to one software component at any given time. The assignment of a development component to a software component may however change over time.

A DC is always a buildable unit and its build can create a number of build results depending on the type of the DC and the number of declared public parts. A build result of a DC may be deployable directly into a J2EE engine, but not every DC produces such deployable build results. Some DCs just produce build output that is used (packed) by other DCs in order to create deployable results. Some DCs may produce build results that are installable ("deployable") into runtime environments other than J2EE like standalone applications or NetWeaver Developer Studio (as Eclipse plugins).

Software Components

A software Component (SC) defines a deliverable, deployable unit. It consists of a number of development components (DC). A SC is basically only a logical shell around a number of DCs.

Software components may declare dependencies between themselves to grant or restrict dependencies between the development components.
they contain and those contained in other SCs.

Example: SC1 declares a dependency on SC2, thus component B has access to the public part of component Z. Since SC1 does not declare a dependency on SC3, component B can not use the public part of component II.

Development Configurations

A development Configuration is the developers view of the development infrastructure. The developer sees the configuration containing "compartments" that in turn contain DCs. A compartment represents one state or version of a SC. The development configuration - or configuration for short - maps this view onto a build server (CBS) and one or more source code repositories (DTR).

Since development configurations contain all the information about

- the version of the SC
- the build server (CBS) to use
- the repositories (DTR) where sources are managed
- the name server ensuring uniqueness of important names
- the software logistics server (CMS) which is responsible for consolidation, propagation and definition of the development landscape

the only thing a developer has to do to set up his development environment is to import a configuration. See "Importing a Development Configuration" for details on how to do this.

Design Time Repository

The Design Time Repository (DTR) is a source code management (SCM) or version control system (VCS) like Perforce, ClearCase, CVS and others. The DTR operates on files or folders. All files that make up development components are stored and versioned in a DTR. The DTR is integrated with the IDE so that it will automatically add the required source files, property files, metadata files, etc. but not files that are generated at build time like Java class files, build logs, etc.

Information about the DTR server is included in the development configuration. Usually, you do not work with the DTR directly but only through the Active Team View, Inactive Team View, Local View and Activity View that provide some form of abstraction. All these views are part of the Development Configuration perspective.

There is also a special DTR perspective available that provides low-level access to the DTR. Remember that what you see in the Development Configuration perspective is an abstraction, i.e. the structures that are visible in these two perspectives look slightly different.

It is important to remember that the DTR deals with files and folders. Some objects that you deal with while developing on a DC might be stored in more than one single file, e.g. a WebDynpro application. This may cause unexpected behaviour from the IDE. E.g. opening the DC Metadata file (.dcdef) for editing does not automatically open the public part definitions of the DC for editing.

Component Build Service

The Component Build Service (CBS) is a central service that builds your development components. When you activate an activity - automatically after check-in or manually from the Activation View - the CBS builds the modified DCs and all DCs that depend on them. If the build is successful the activity is made active, i.e. the changes are integrated into the version visible in the Active DCs View. You can check if your activation requests were successful in the Activation Requests view.

Information about the CBS is included in the development configuration.

Name Server

The Name Server is a central server which is the naming authority ensuring unique names (e.g. for components, Java packages, context roots of web applications, DB table names etc.). When you e.g. create a new DC the IDE asks the Name Server for a name reservation for the name of the new DC. Only if this reservation can be obtained from the Name Server your new DC will be created otherwise you have to choose another name which is not yet reserved by some other component.

Change Management Service

The Change Management Service (CMS) is a central service which is responsible for definition and administration of a development landscape consisting of one or several CBS and DTR servers and a Name Server. Development Configurations are created by a CMS administrator. In addition it controls propagation of sources and archives between different tracks (development areas for different development projects or different versions of development projects) in the same organization (e.g. SAP) or propagation to other organizations (e.g. customers, partners).