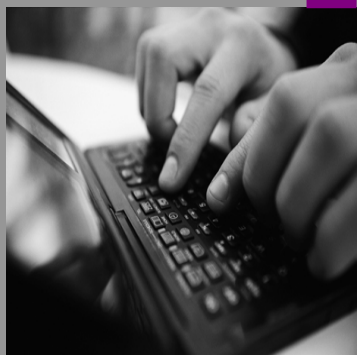

How-to Guide
SAP NetWeaver 2004s

How To... Development Landscape Scenarios in NWDI

Version 1.00 – November 2006

Applicable Releases:
SAP NetWeaver 2004s

1 Scenario

Customers develop or modify Java based applications that run on SAP Application Server Java (short: SAP AS Java). For Java-based development, SAP NetWeaver Developer Studio (short: NWDS) is used. Compared to the ABAP world, where design time and runtime are tightly integrated within the same server, the Java world looks different. Developers work with a local development environment that consists at least of an IDE (here: NWDS). To test new developments, an SAP Application Server Java is needed that is not part of NWDS. This first test server is an essential part of the development environment and landscape.

In NetWeaver ’04, SAP introduced the SAP NetWeaver Development Infrastructure (short: NWDI) for the complete software life cycle management of Java-based applications. NWDI provides central administration of the development environment for all developers, source code and build control, and management of the whole transport landscape.

The development environment (IDE and test engine), in combination with NWDI, provides you with several scenarios for using runtime systems (short: RTSs) for development and early integration tests. This guide helps you to clear up any questions that arise during the setup of the landscape. It focuses on the development landscape scenarios, that is, the environment (test systems) where developers test their actual code.

2 Introduction

This How-To Guide gives an overview of the most common landscape scenarios possible when setting up a development environment with NWDI. Customers have specific requirements because of defined corporate processes and/or hardware and software requirements. The guide introduces the different possible landscapes for the development environment and emphasizes the use of runtime systems for early integration tests.

3 Related Information

Additional information about SAP NetWeaver Development Infrastructure can be found at the following locations:

SAP Help Portal
http://help.sap.com/saphelp_nw2004s/helpdata/en/01/9c4940d1ba6913e10000000a1550b0/frameset.htm

SDN
http://sdn.sap.com/irj/sdn/developerareas/was > Key Topics: Development Infrastructure (NWDI)

4 Developer's Test System

Developing Java-based applications, like J2EE web applications, Web Dynpro or business logic with Enterprise Java Beans technology means, in general, local development instead of the central development that we know from the ABAP world. Local development means that developers usually work with their desktop PCs or laptops and have a local IDE (here: SAP NetWeaver Developer Studio) installed. For testing purposes they need at least an SAP Application Server Java to check deployments, to check whether compiled code executes correctly and to run simple unit tests. The test server can be installed as a local workplace installation for each developer, or a remote engine that is used by all developers for early integration tests.

On the one hand, this local development approach (which is common in the Java community) increases the flexibility of the development process. On the other hand, it may increase the complexity of the system landscape and therefore requires proper process and landscape planning to benefit from this flexibility.

When you set up a development and test landscape with NWDI, several scenarios are possible involving local, remote, or central development systems. But before we introduce these scenarios, some remarks concerning test systems in general:

4.1.1 Setup of Test Systems

In a three or four system landscape, the connected runtime systems usually have different roles for different testing purposes. The first test system, used by the developer directly, should be a system where developers can deploy and run their code. Depending on the type of application, some requirements for the setup of a test server arise. In some cases, a pure SAP AS Java server is not sufficient, because the application uses some additional NetWeaver components or back-end systems to be able to run correctly. For the local developer server, a Developer Workplace installation (usage type AS Java and EP) is generally used. Test scenarios for complex applications using cross components like ESS, MSS, CRM, BI, and so on are not possible out-of-the-box. These scenarios are usually tested in a central test environment because setting them up on each developer's engine would cause much more work. Depending on the scenario, it might be possible to combine local engines with other components or back-end systems from a central test environment.

4.1.2 Upgrades with JSPM

Hint: Be aware that NWDS and the SAP AS JAVA server on which the developed software is deployed must always have exactly the same version (Support Package level). Otherwise you cannot guarantee that they will function properly.

As mentioned, the recommendation is that the Support Package level of the NWDS must match the Support Package level of the runtime systems where the application is deployed. Therefore the test servers have to be updated from time to time. The update (Support Package) is done using JSPM; for upgrades to a target release, the SAPJup tool is used. Furthermore, if additional business packages, such as XSS, are needed in the local test system they have to be installed as described in the installation guides.

5 Development System Scenarios

This guide explains possible development system scenarios with their pros and cons, so guiding you in your decision about how to set up the runtime systems in your development landscape. In the context of NWDI it discusses the runtime systems for developers, which means all systems where developers have access rights for deployments. The complete NWDI transport landscape with CONS, TEST, and PROD systems is not part of this discussion.

5.1 "Private" Test System

The first scenario is a local development environment with a local test system. Every developer has his or her own local SAP NetWeaver Developer Studio (NWDS) for application development. To test the application, a '"private" SAP AS Java engine is used exclusively by each developer and is usually installed on the local machine. The installation would be a Developer Workplace installation where the NWDS with an SAP AS Java engine is installed (for MaxDB or MS SQL Server).

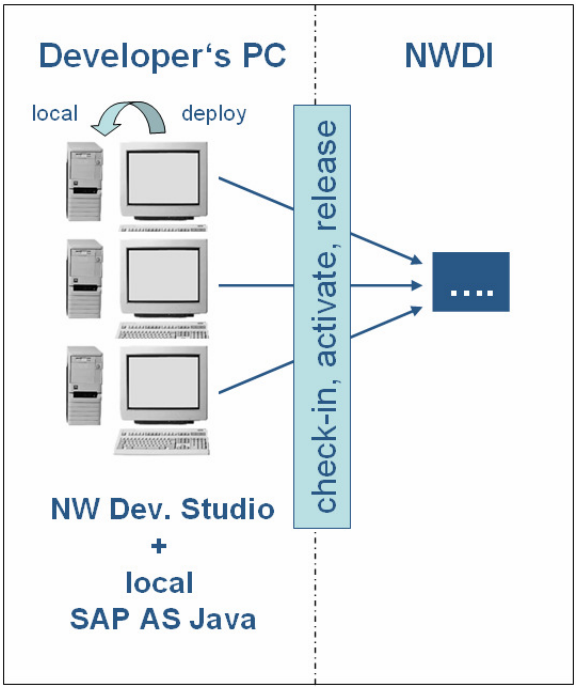

Figure 1: Developer's PC with local SAP AS Java

The engine that the developer uses for local tests is specified in the following NWDS configuration: `Window – Preferences – SAP J2EE Engine`.
Remember that, when we talk about local engines in this guide, we mean the SAP J2EE Engine that is configured by choosing Window – Preferences – SAP J2EE Engine.

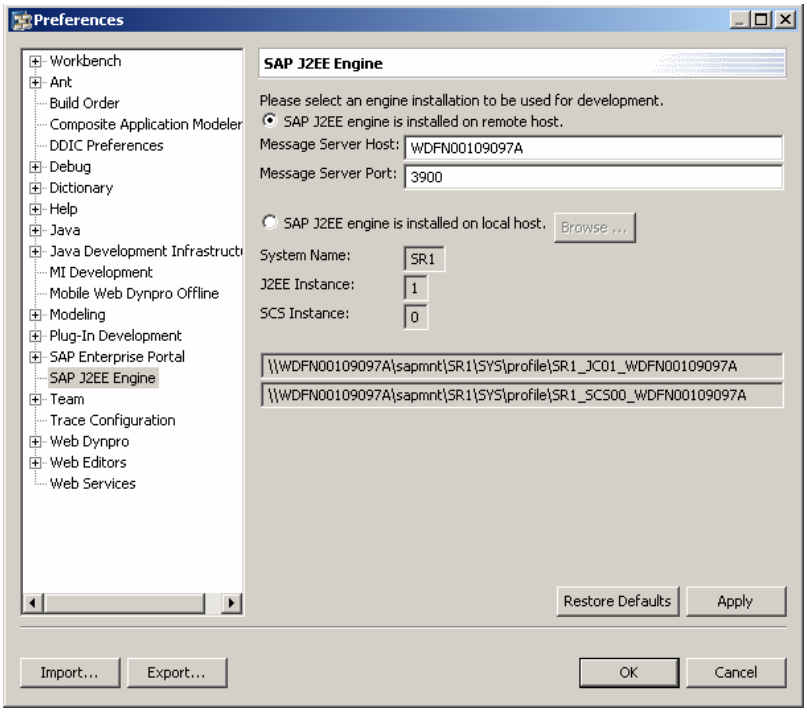

Figure 2: NWDS – Configuring a local/remote SAP J2EE Engine

A remote engine (engine is installed on a separate machine) can also be configured in the case where the developer PC is not powerful enough or an additional engine is needed. But in both cases, the developer uses the engine on its own for “local” deployments and tests (right-click the development component project and select Development Component – Deploy). The phrase “local engine” will be used here, even if the engine is installed on a separate, remote machine. In this scenario, the local engine is not configured, administrated, or controlled by NWDI.

The advantage of a local engine is that every developer has an independent and isolated test environment, which does not affect others during their work. There is no overwriting issue, if developers are working in the same deployment unit. And, last but not least, every developer can debug his or her application whenever he or she likes and does not interfere with others during a breakpoint. Developers can intensively test their local changes in a local test engine before they decide to check them in and make them visible to others. This is an example of the degree of freedom that is not known (and not possible) in the ABAP world. In the Java world the local environment is very common and can increase a developer’s efficiency.

On the other hand, a local engine means that an extra system (SAP AS Java engine) has to be set up and updated for each developer, and sometimes upgraded as well. This is of course administrative effort that has to be calculated and considered.

After a successful local test, developers check their sources into the DTR to make them visible for other team members within the inactive workspace. The Activation step starts the central CBS build and, after a successful build, the automated deployment to the central DEV system (if configured in the CMS track) is triggered.

5.1.1 Deployment of Single DCs

Since the local engine is not administrated or controlled by NWDI, no automatic deployment of used archives (used DCs) takes place. The developer is responsible for getting and deploying the required DCs. In a larger development team, many developers work within one or more software components and develop new development components that have usage dependencies. To test his or her own application at runtime, a developer has to deploy the dependent DCs (from other developers) on the local test machine as well. Usually the source code is not needed just for deployment and runtime tests. Used DCs can be deployed from NWDS. Go to the *Active DCs* view within the *Development Configurations* perspective, right-click the DC that you want to deploy, and choose *Sync Archive*. After the archives have been synchronized, choose *Deploy* from the context menu to deploy the DC to your local engine. This is of course only possible for DCs that have a deployable build result.

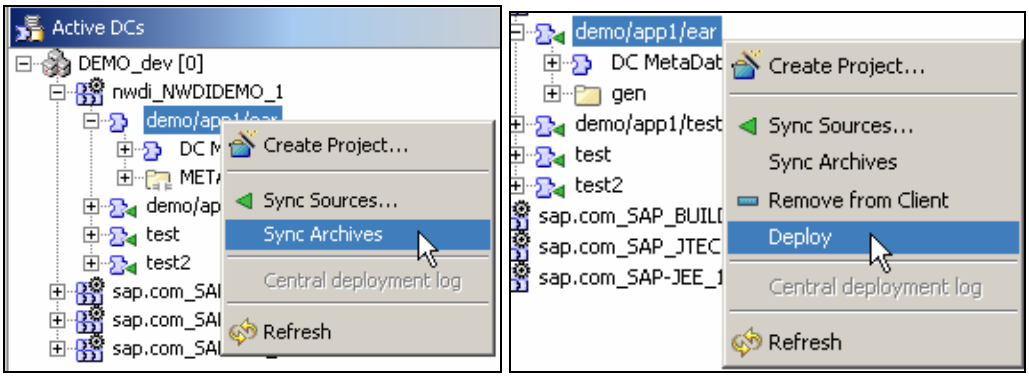

Figure 3: Synchronize and deploy used DCs to local engine

5.1.2 Deployment of SCs

Sometimes you need to deploy a complete SC on which a usage dependency exists on a local test system to test the current application. The recommendation is that a CMS administrator downloads all required SCAs on a regular basis and puts them on a share for developers. The developers then deploy the SCAs on their local machines using SDM.

To download the software components from CMS, use the Download Servlet. Open a browser window and enter **http://<cbshost>:<port>/SCD/SCDownload**. You get the following screen where you enter the DomainID, TrackID, Vendor, and the software component name. You also have to select the system (e.g. Development) and choose the option “without buildarchives”. Click the *Create Archive* button. A link is generated where you can download the created archive (SCA file) for the specified software component.

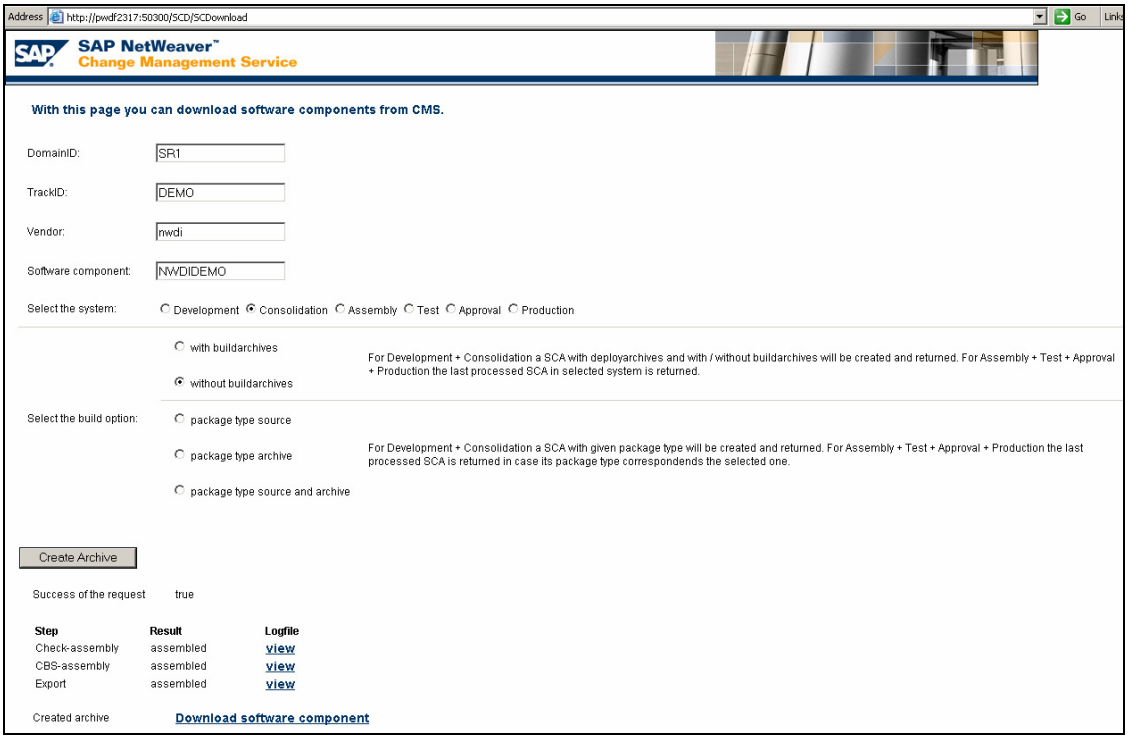

Figure 4: SCDownload Servlet

5.2 Shared Remote Test System

The following scenario is similar to Scenario 5.1. Therefore the topics described in 5.1.1 and 5.1.2 apply here as well. The difference is that several developers share one remote SAP AS Java engine for local deployments and to test their local changes. The remote engine is configured as described in 5.1, but in scenario 5.2 the engine is shared by more than one developer.

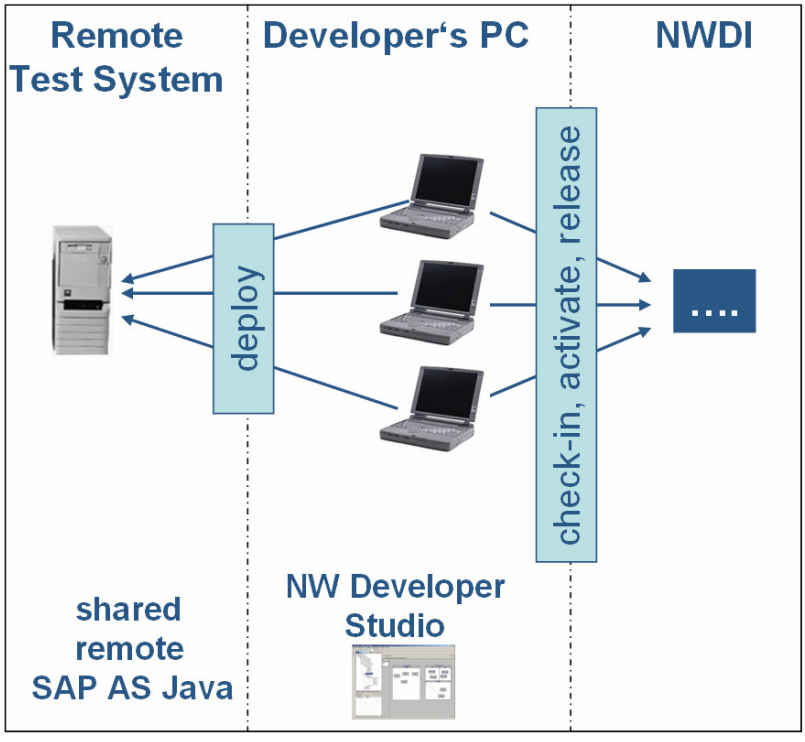

Figure 5: Shared remote SAP AS Java

This scenario can be used when developer PCs are not powerful enough for an additional engine or the required testing environment setup is too complex for each developer. But keep in mind that only one developer at a time can debug (parallel debugging should be possible in the next major release) and that parallel deployments of the same deploy unit lead to overwriting issues.

The fact that several developers deploy their applications and underlying DCs and SCs to the shared engine might cause overwriting problems. Developers cannot be sure that they actually test their deployed versions of their own DCs as well as the dependent DCs if another developer is deploying the same DCs at the same time. To deal with this problem some rules concerning when and what may be deployed are required. In addition, this overwriting problem must be taken into account in the DC and SC architecture.

A software design where the applications are independent deployable units (DCs) that all use the same underlying framework avoids most of these overwriting problems. Moreover, if it is possible to keep the framework layer stable and if it is an independent deployable unit, it can be deployed separately on a regular basis (for example, once a day). In this case, all application developers know that they have to resynchronize the corresponding libraries of the new framework version to rebuild their application.

An example of an architecture that leads to overwriting problems when using a shared remote engine would be the following:
An enterprise application consists of several Web Modules and EJB DCs. Every DC is developed by more than one developer. There is only one deployable unit (the EAR DC) that includes all other DCs (DC reference of type Assembly). Every developer works locally and tests his or her changes without checking them in. When assembling and deploying the EAR file, each developer deploys his actual DC, but as a side effect deploys old versions of all other DCs contained in the same EAR file. It is obvious that each deployment overwrites the previously deployed DCs of all other developers. We do not recommend the shared remote engine approach for this type of architecture.

The advantage of this scenario is the possibility to test developments “locally” before checking them into DTR and releasing them for propagation through the landscape. The TCO is reduced compared to Scenario 5.1 because of a lower number of runtime systems (cost reduction and less administration work). A variant of this scenario is to use multiple group servers. Each development group has its own remote server for its “local” tests.

5.3 One Central DEV System – No Local Test System

The next scenario describes the use of a central NWDI DEV system as the only test system for developers. Here a similar situation arises as in 5.2, where developers share a remote runtime system. The difference is that the SAP AS Java engine is an NWDI controlled system, a runtime system (RTS) that is configured within the DEV system of an NWDI track. Here the local/manual deployment is switched off – no J2EE engine is configured in the Developer Studio preferences. Automatic deployment is enabled in the CMS track.

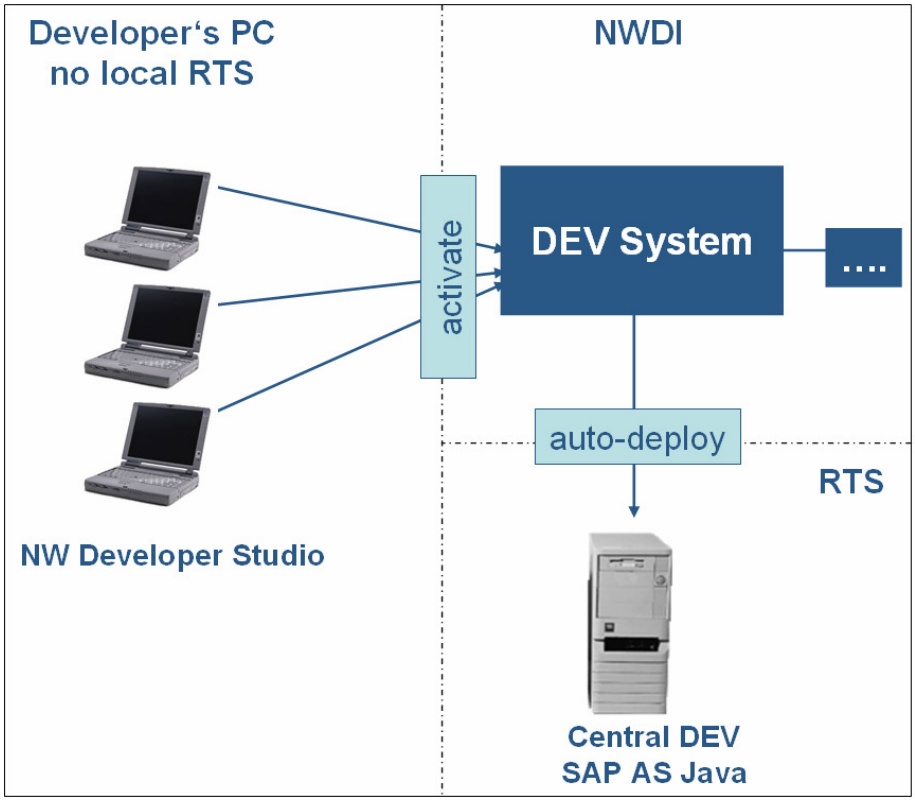

Figure 6: Central NWDI DEV system – no local runtime system

In contrast to Scenario 5.2, the deployment takes place after the developer checks in and activates his or her changes. Only if the activation step is successful, (which means a central build is performed) does the SDM deployment start in the background. The advantage is that the deployment can still be triggered from the developer within the Developer Studio. It is transparent to the developer, which means he or she does not need to know the SDM login credentials, ports, and so on. No CMS administrator is needed to deploy the changed applications in the central DEV system. If the central build requested during the activation step is successful and the automatic deployment is enabled, the changes are deployed to the DEV RTS immediately.

Another advantage of this scenario is that all dependent DCs are always up-to-date and deployed in the connected RTS. For this scenario we do not have an overwriting problem even with a software architecture as described in 5.2 where the DCs of multiple developers are contained in one deployable unit. This is because the centrally built version, including the newest (active) DCs of all developers, is used for the deployment.

As already mentioned, if automatic deployment is enabled, every developer can decide when he or she wants to deploy to the central DEV RTS. This also means, however, that no defined system state is possible in the DEV runtime system. But in contrast to Scenario 5.2, the deployed DCs are always in an active state in the sense of a successful build result.

The disadvantage in comparison to local/remote engines (Scenarios 5.1 and 5.2) is that developers are forced to check in and activate their changes before they can test their code. This leads to a high number of versions and activities. But what is more annoying is the longer development or deployment cycle, that is, the time between saving the source changes in the editor and the time when tests can start. This usually does not meet the developer's requirement for early development tests. Another disadvantage is

that, in contrast to the previous scenarios, developers can only test their changes if the code compiles against the active state within the CBS. This means you have to react to all incompatible changes that other developers activate before you can continue with your tests. Last but not least, the software life cycle theory recommends testing the code before checking it into the DTR.

The development landscape described here is mainly used for maintenance scenarios and not for ongoing development (or only in very rare cases). In a maintenance scenario longer deployment cycles might be acceptable because the number of deployment cycles required for one bug fix is typically very small.

5.4 Central NWDI DEV System for Local and Central Tests

Some situations do not allow a local or remote engine as described in 5.1 and 5.2 because of hardware restrictions or time constraints. But nevertheless developers have a requirement for fast deployments and local tests. Here the following scenario can be used, where the central NWDI DEV system is used also as a “local” engine for local tests. Developers do this by specifying the NWDI DEV system as an SAP J2EE engine in the preferences of their NWDS (see Figure 1).

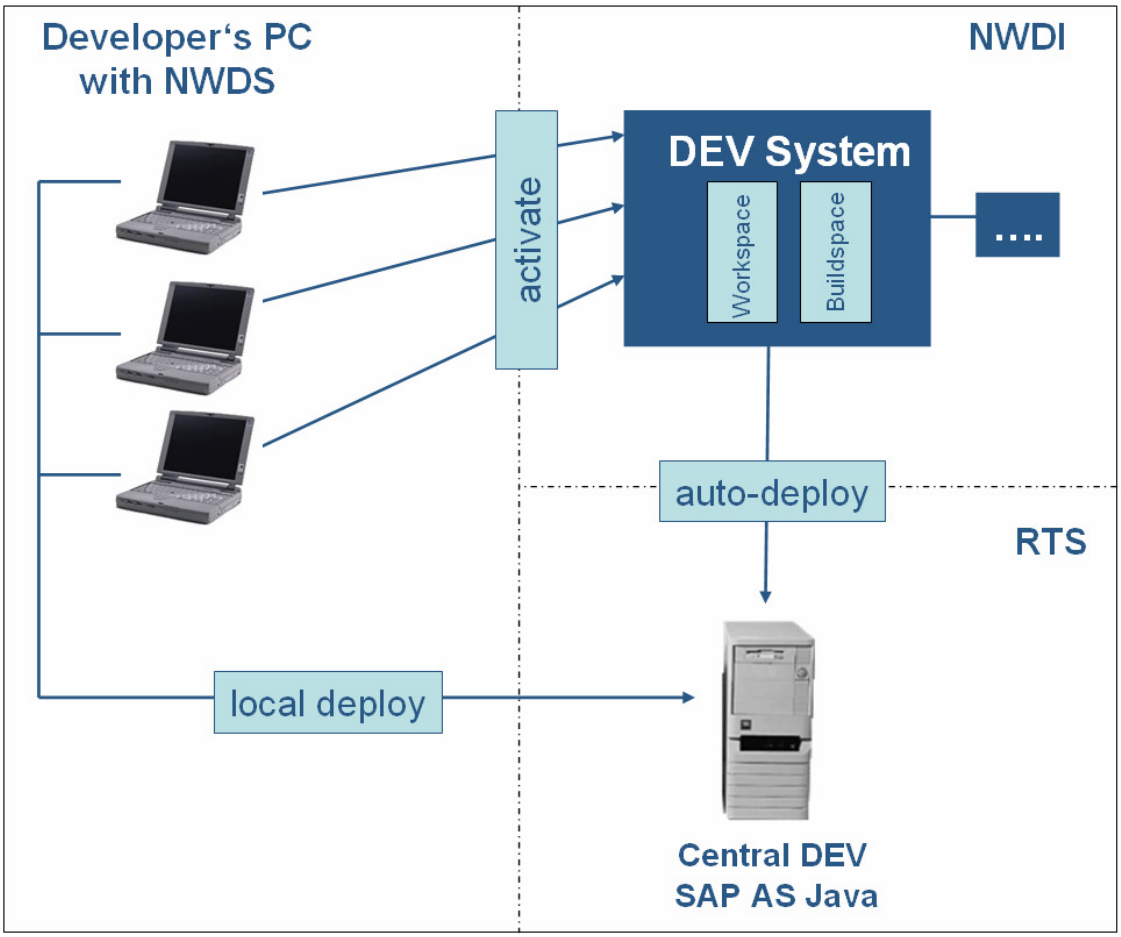

Figure 7: Central NWDI DEV system for local and central tests

This scenario is similar to 5.2 because all developers share a remote SAP AS Java engine to test their changes locally. The difference here is that the remote engine is used as an NWDI DEV system as well. So the deployment takes place twice. First, the

developer deploys directly from the Developer Studio and, later, the activated changes are deployed in an automatic CMS deployment. Developers must be aware that a mixture of inactive and active DCs is deployed to the DEV system. Furthermore the overwriting problem discussed in 5.2 persists here as well.

Even so, this scenario is sometimes useful within a maintenance scenario, where the developer has to develop for a new release and has to fix a bug in a productive version that runs in parallel on an older SP level or release. In such a scenario it is expected that there are only few developers working in parallel, which reduces the overwriting problem. To test a bug fix, a developer would not usually set up a local test system. The configuration of the DEV runtime system (configured in the Maintenance track) as a local engine in the Developer Studio gives you the advantage of direct deployments and early tests, unlike in scenario 5.3.

5.5 Standalone System for Local and Central Tests

The next scenario describes how to use one standalone runtime system as a local test engine for each developer and as a system for early integration tests. The system is configured as an SAP J2EE engine within the Developer Studio of each developer and can be used for local tests.

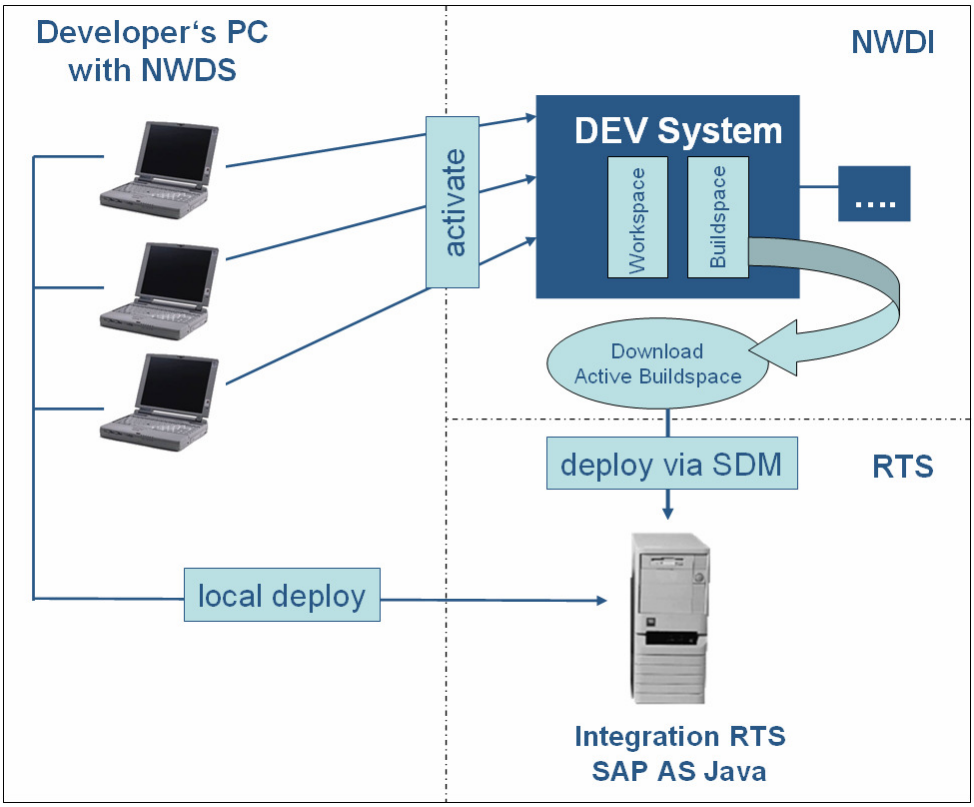

Figure 8: Standalone central RTS for local and central tests

A defined state of the deployed components is needed for integration tests. This is usually the active buildspace that is updated during the activation step. The CMS administrator downloads the build results of the active buildspace from the DEV system and deploys them (using SDM) to the integration runtime system at regular intervals (e.g. every Monday or every night). The integration runtime system is a standalone engine, which means it is not configured as an NWDI runtime system within the track

configuration. Therefore the activation step does not trigger an automatic deployment to this system.

To deploy the active state of the development system, the Download Servlet (see 5.1.2) is used to download the active SCA for a specific software component. Open a browser window and enter http://<host>:<port>/SCD/SCDownload. Enter the *DomainID*, *TrackID*, *Vendor*, and the *Software Component* name. Choose the Development system and the option *without build archives*. Click the *Create Archive* button and wait until the download link appears. Download the sca file and deploy it using SDM.
To enable an integration test of the active development state, the local deployment should be avoided during that test phase. Furthermore, all changes should be checked in and activated. The administrator can also activate all closed activities (activities that are checked in but not activated) in the NW Developer Studio by setting the user filter to *all users*.

You can also use the DOWNLOADSCA command in the CBS Command Line tool to download the SCA from the DEV buildspace.

Hint: During the SCA download process you have to make sure that the CBS build is not running. If a DC is being built, it will not be part of the created archive. Therefore you have to stop CBS by setting the *Request Processing Mode* to *OFF*. This can be done in the CBS WebUI – Buildspaces. Choose the *Edit* button, change the value for that mode, and save your changes (see Figures 10 & 11).

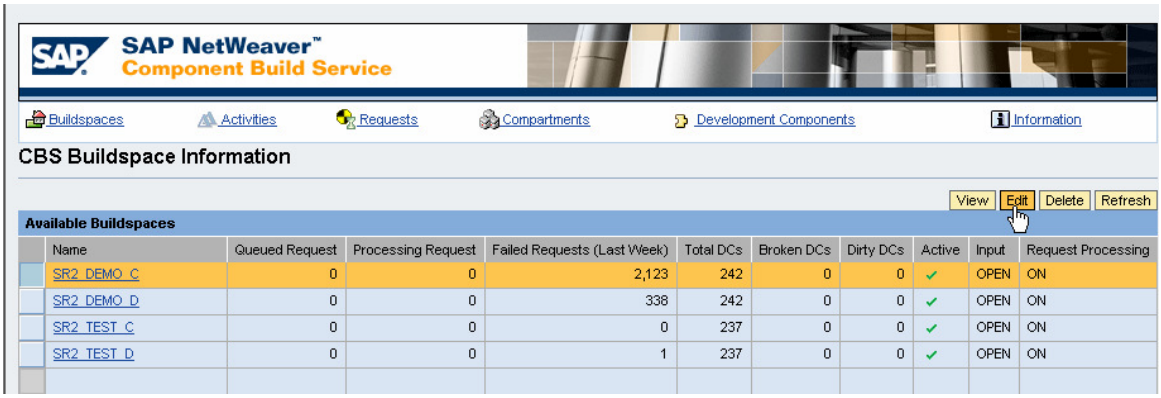

Figure 9: CBS WebUI – Buildspaces – Choose Edit

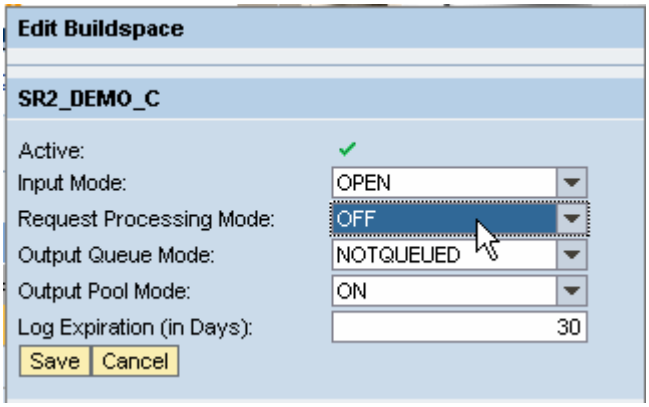

Figure 10: Request Processing Mode - OFF

After downloading the SCA, the administrator deploys it using SDM. Here you have to specify the following option “Update deployed SDAs/SCAs that have any version” (see Figure 12). This is needed because the SDAs deployed directly using Developer Studio might have a newer version. Furthermore, you can have the situation that a new DC is created and deployed in a local deployment, which has not been activated. The downloaded SCA of the active buildspace does not include the newly created but non-activated DC. Since the deployment of the active SCA does not delete DCs that are not part of the SCA, the SDA exists, but not as part of the SCA. If the DC has to be deleted, you have to undeploy it manually using SDM.

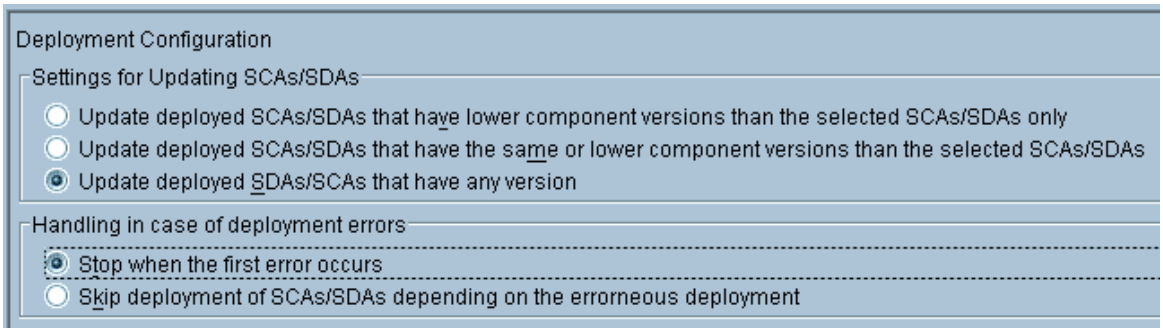

Figure 11: SDM deploy options

5.6 Local Test System + Central NWDI DEV System

A scenario especially useful for ongoing development is to use a local test system and a central DEV system. The local engine is used as described in 5.1 (or 5.2) for independent local tests and the central DEV system is used as described in 5.3 for early integration tests.

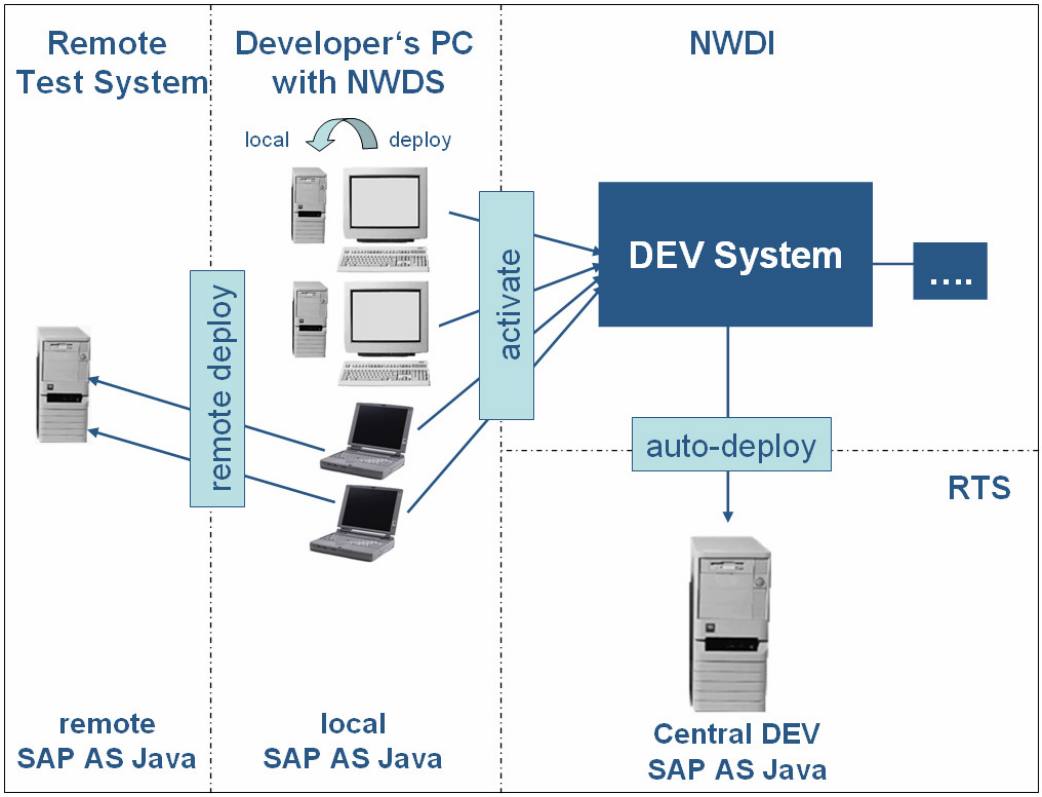

Figure 12: Local Test System + Central NWDI DEV System

The deployment to the central DEV system is done automatically if automatic deployment is enabled. This option enables the “Deploy on Demand” feature, where the developer can decide when the new changes he or she checked into the DTR are deployed to the central DEV runtime system in the activation step.

Since the deployment to the central DEV system is triggered by each developer, it is used by developers to test their new changes together with all other active development, before they release their activities to the CONS system. The CONS system can be used to test a defined state during a specified phase before the software component is propagated to the final assembly test. In a three-system landscape, the CONS system is usually omitted. The combination of local tests and early integration tests combines, of course, the advantages of both scenarios. But on the other hand the number of required runtime systems is higher which leads to higher TCO.

6 Summary

Several possible development scenarios are described in this guide, and the pros and cons, side effects, and things to consider for each scenario have been pointed out to you. In comparison to the ABAP world, the local development environment is much more flexible and gives you greater liberty in setting up a development landscape. This flexibility effectively spoils you for choice and you need to balance your reasons for using one scenario or another. Aspects like software architecture, hardware budget, TCO, administration work, time need for system setup and maintenance, number of developers and size of teams, and so on are important factors in coming to the right decision. There is no recommended scenario because it depends on which scenario fits your landscape, project environment, hardware setup, and so on best. What is important in any case is to consider any side effects or restrictions that are part of the development system setup. Therefore, detailed planning and some rules for the development process should be part of every setup of an NWDI Java development landscape.

7 Abbreviations

CMS	Change Management Service
CONS	NWDI Consolidation System
DC	Development Component
DEV	NWDI Development System
IDE	Integrated Development Environment
JSPM	Java Support Package Manager
NWDI	NetWeaver Development Infrastructure
NWDS	NetWeaver Developer Studio
PROD	NWDI Production System
RTS	Runtime System
SAP AS Java	SAP NetWeaver Application Server Java
SC	Software Component
SCA	Software Component Archive
SDA	Software Delivery Archive
TEST	NWDI Test System
TCO	Total Cost of ownership

www.sdn.sap.com/irj/sdn/howtoguides