

**Galileo Knowledge Products:
Softwareentwicklung mit der
Java Development Infrastructure**



Autor: Matthias C. Kneissl
Produkt Management Conigma™
Galileo Galileo Group AG
Perchtinger Str. 6
D-81379 München
Tel. +49 89 710 463 60
info@galileo-group.de

1 Genereller Überblick

Der SAP WebAS Java beinhaltet einen JEE Server, der seitens SAP mit zusätzlichen Eigenschaften erweitert wurde. Eine wesentliche Erweiterung ist die Java Development Infrastructure (JDI), die den üblichen, JEE basierten Softwareentwicklungsprozess automatisieren soll. Dabei handelt es sich um ein webbasiertes Werkzeug von SAP, das Entwicklungsverantwortliche und Entwickler bei der Implementierung und Verteilung von Software unterstützt. Der typische JEE Verteilungsprozess, der meist manuell oder skriptgesteuert abläuft, wurde durch ein von SAP entwickeltes Prozessframework ersetzt, das die einzelnen Entwicklungsschritte vereinfacht und webbasiert steuern lässt. Wesentliche Komponenten dieser Infrastruktur und der damit verbundene Prozess werden im Folgenden vorgestellt.



2 Die verschiedenen Komponenten in der JDI

2.1 Die Java Development Infrastructure im Gesamtüberblick

Die JDI ist nicht standardmäßig auf allen eingesetzten SAP WebAS Java Systemen vorhanden, sondern muss eigens auf einem System installiert werden. Dies ist klassischerweise das Entwicklungssystem. Alle weiteren Systeme werden von der JDI entsprechend angebunden und kontrolliert. Die Kernstücke der JDI bestehen aus dem Design Time Repository (DTR), ein von SAP entwickeltes Versionsverwaltungssystem, einem Change Management Service (CMS), der Änderungen an Software verwaltet und einem Component Build Service (CBS), der dafür verantwortlich ist, neue Archive zu erstellen, die auf WebAS Java Servern installiert werden können. Zusätzlich ist auch ein Name Service enthalten, der für die systemweit eindeutige Namensvergabe von Entwicklungskomponenten zuständig ist.

SAP hat mit der JDI versucht die üblichen Fehler, die in einem Java-basierten Entwicklungsprozess auftreten können, zu minimieren. Diese bestehen bei Java basierten Softwareentwicklungsprozessen darin häufig, die richtigen Sourcen zu finden, die dazu passenden und verlinkten Bibliotheken einzubinden und einen lokalen Build und Test durchzuführen. Bei diesem Schritt kann es vorkommen, dass abhängige Komponenten nicht entsprechend geprüft werden und es daher beim Einchecken zu entsprechenden schwerwiegenden Fehlern kommt. Weitere Problemfelder verbergen sich im zentralen Build, da es durchaus vorkommen kann, dass sich die lokale Entwicklungsumgebung anders verhält als die zentrale Systemumgebung. Diese Fehler treten vor allem im WebAS ABAP Stack nicht auf, da hier die Entwicklungsumgebung und auch die Laufzeitumgebung zentral auf einem System liegen. Um den üblichen Java Entwicklungsprozess zu verbessern hat SAP dazu Anleihen aus dem bereits erprobten ABAP Stack entnommen und den Entwicklungsprozess für den WebAS Java entsprechend angepasst.

2.2 Das SAP System Landscape Directory

Das System Landscape Directory (SLD) dient innerhalb einer SAP Systemlandschaft als zentrales Verzeichnis, in dem erfasst wird welche Software Komponenten auf welchem System in welcher Version vorhanden sind. Datentechnisch werden diese Informationen über *das Common Information Model (CIM)* abgebildet.

Abbildung 2-1 zeigt hierbei die Elemente des SAP Komponentemodells. Ein Produkt kann dabei aus mehreren Softwarekomponenten bestehen, denen jeweils ein Release zugeordnet

ist. Diese Komponenten bestehen aus eindeutig zugeordneten *Development Components*, zu Deutsch Entwicklungskomponenten. Diese stellen Entwicklungseinheiten dar, die durch die *Development Objects* untergliedert werden. Letztere werden hierbei als versionierte Quelldateien abgelegt.

Bei der Erstinstallation einer SAP Netweaver Instanz ist es Pflicht, ein neues

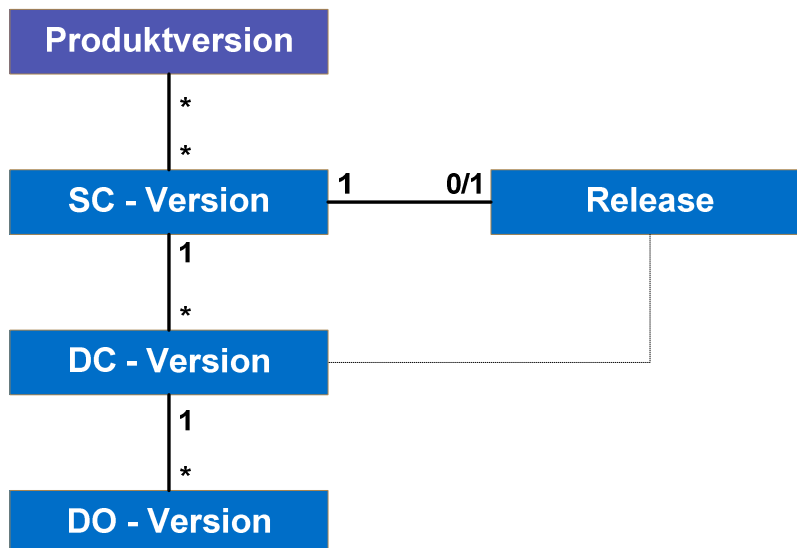


Abbildung 2-1: Elemente des SAP Komponentenmodells

SLD zu erstellen oder an ein bereits bestehendes anzubinden. Administriert wird dieses dabei über eine komfortable Weboberfläche. Ziel ist es, Aufschluss zu erhalten, welche Software auf welchen Systemen zum Einsatz kommt. Mit der initialen Installation sind bereits alle von SAP erhältlichen Softwareeinheiten im SLD bekannt.

Sobald SAP neue Patches für eigene Komponenten ausliefert, werden diese Informationen publiziert und stehen über einen Datenimport ins System Landscape Directory dem Kunden umgehend zur Verfügung. Bevor ein Kunde eine Eigenentwicklung einem SAP System im SLD zuordnen kann, müssen zunächst dieses Produkt und alle Abhängigkeiten im SLD definiert werden. Durch die Gliederung des Komponentenmodells ist dabei jeder Softwarekomponente auch eine eindeutige Releasenummer zuzuordnen.

Während für ABAP Entwicklungen keine technische Voraussetzung besteht, dass diese nach Software und Komponenten strukturiert im SLD definiert sind, verhält sich dies bei Entwicklungen in der Java Development Infrastructure anders. Hier besteht der erste Schritt bei einer Neuentwicklung einer Software oder einem Releasewechsel darin, das Produkt und dessen Abhängigkeiten im Verzeichnis zu pflegen. Dabei müssen auch die Softwareeinheiten angegeben werden, auf die in bestimmten Phasen des Lebenszyklusses referenziert wird. Beispielsweise können dies für die Phase Kompilierung Soft-

warebibliotheken sein, die vorliegen müssen oder bei einer geschichteten Entwicklung die Software der unterliegenden Schicht, die referenziert wird. Die Information, auf welchen Systemen diese zum Einsatz kommt kann nur manuell gepflegt werden. Dieser Schritt ist für den Entwicklungsprozess nicht notwendig, wenngleich es für komplexe Systemlandschaften sehr hilfreich sein kann, diese Informationen zentral zu sammeln. Leider wird bei einem Transport von Softwarekomponenten im SAP Java Stack oder einem Einspielen eines SAP Patches diese Information nicht automatisch aktualisiert, obwohl es vom softwaretechnischen Standpunkt durchaus realisierbar wäre und manuelle Nacharbeit ersparen würde. Eine automatisierte Aktualisierung würde Fehler weitestgehend ausschließen und dafür sorgen, dass sich das Verzeichnis immer in einem aktuellen und konsistenten Zustand befindet.

In der aktuellen Ausprägung stellt dabei das System Landscape Directory zumindest für den Java Stack lediglich eine Verzeichnis definierter Softwarekomponenten dar, das sehr aufwändig manuell zu pflegen ist. Es ist daher davon auszugehen, dass in großen Systemlandschaften dieser sehr hohe Pflegeaufwand vernachlässigt wird und lediglich Softwarekomponenten definiert werden, um die Voraussetzungen für die WebAS Java Entwicklung zu schaffen.

2.3 Das Design Time Repository

Die Sourcen innerhalb eines Tracks werden in der JDI über das Design Time Repository (DTR) verwaltet. Diese Technologie besteht aus einem DTR Server und einem DTR Client, die als Austauschprotokoll den DeltaV Standard benutzen. Der Client ist dabei fest in die von SAP modifizierte Eclipse Entwicklungsumgebung integriert, so dass alle Aufgaben, die im Rahmen der Softwareentwicklung anstehen, zentral gesteuert werden können. Der Zugriff auf das Repository ist mit jedem beliebigen anderen Client möglich, sofern dieser das WebDAV (Web Distributed Authoring and Versioning) Protokoll unterstützt.

DeltaV erweitert den WebDAV (Web Distributed Authoring and Versioning) Standard um eine Versionierungskomponente und bietet damit alle Funktionalitäten, die an eine zentrale Sourcecodeverwaltung gestellt werden. Vergleicht man das DTR mit Subversion oder in der Funktionalität ähnlichen Werkzeugen, so sind einige Besonderheiten zu bemerken. Der DTR Server legt die Sourcen als Binary Large Objects (BLOB) in der internen Datenbank ab. Als besonderes Feature kann damit auch ein uncheck-out durchgeführt werden, welches den zuvor durchgeführten check-in Vorgang rückgängig macht.

Im Sinne des DTR müssen Änderungen an Sourcen immer einer Aktivität zugeordnet werden. Eine Aktivität ist genau einem Entwickler zugeordnet, kann nicht über verschiedene Entwicklungsumgebungen geteilt werden und wird atomar in das Repository eingebracht.

Damit ist die Aktivität mit dem WebAS ABAP spezifischen Container der CTS-Aufgabe vergleichbar.

Änderungen können in der JDI immer nur auf Aktivitätenebene in das Repository eingebracht werden. Durch diese Tatsache wird die übliche Problematik, dass Entwickler einzelne, geänderte Sourcen einchecken, was bei einem check-out zu Compilefehlern führt, behoben, da sich nicht alle betroffenen Klassen aktualisiert im Repository befinden. Aufgrund dieser Funktionalität ist das DTR eng mit der Entwicklungsumgebung gekoppelt, so dass stets eine Verbindung bestehen muss. Im Gegensatz zu Subversion ist es damit beispielsweise nicht möglich, dass Dateien offline dem lokalen Repository hinzugefügt und beim nächsten Synchronisationsvorgang global festgeschrieben werden. Sobald Änderungen in das Repository eingebracht sind, können diese im SAP Netweaver Developer Studio aktiviert und damit die Übersetzung angestoßen werden.

2.4 Technischer Überblick zum Component Build Service

Der Component Build Service ist für die Übersetzung von Entwicklungskomponenten zuständig. Das Design Time Repository speichert aus der Entwicklungsumgebung eingebrachte Sourcecodeänderungen zunächst im Verzeichnis ‚inaktiv‘ ab. In diesem Status sind die Änderungen für andere Entwickler nicht sichtbar, da ein check-out der Entwicklungssourcen immer aus dem Verzeichnis ‚aktiv‘ durchgeführt wird. Der automatische Übergang in den Status ‚aktiv‘ wird erst dann durchgeführt, wenn der globale Build erfolgreich durchgeführt wurde. Da durch das Trackkonzept der Build komponentenbasiert erfolgt, sind langwierige nächtliche Builds nicht notwendig. Sofern eine Komponente neu übersetzt wird, erkennt der CBS automatisch davon abhängige Komponenten und sorgt dafür, dass auch diese aktualisiert werden.

Ein Build Request kann vom Entwickler in seiner Entwicklungsumgebung an den CBS gesandt werden, nachdem die dazu entsprechenden Aktivitäten in das DTR eingchecked wurden. Der Component Build Service verwaltet diese Build Requests in einer internen Queue, die nach dem First-in-First-out Prinzip verwaltet wird. Sobald eine Aktivität zum Build ansteht, werden die Sourcen und die zusätzlich notwendigen Archive in eine temporäre Verzeichnisstruktur kopiert. Die Plugins, die für den Build Prozess notwendig sind, werden aus dem Speicher geladen und der Build angestoßen. Sofern der CBS den Build erfolgreich durchführen konnte, wird im DTR die Source aus dem Verzeichnis ‚inaktiv‘ in das Verzeichnis ‚aktiv‘ verschoben und steht damit allen Entwicklern zur Verfügung.

Die Resultate des Build Prozesses, die dabei entstandenen Archive, werden in den Archive Pool kopiert. Sofern der Build nicht erfolgreich durchgeführt werden konnte, verbleiben

die Sourcen im inaktiven Zustand und die Komponente wird markiert. Es steht dem Entwickler für Notfälle frei, die Sourcen manuell in den Zustand ‚aktiv‘ zu bringen, allerdings wird die Komponente vom CBS dann als ‚BROKEN‘ markiert, um anzuzeigen, dass der Build nicht erfolgreich war, und sich die Archive in einem inkonsistenten Zustand befinden. Da besonders in der Entwicklungsumgebung ein Build sehr häufig durchgeführt wird, um Änderungen zu propagieren, kann der Component Build Server auf mehrere Serverknoten verteilt werden oder multithreaded ablaufen. Im Rahmen der Java Development Infrastructure Installation empfiehlt SAP, den Build Service auf mehrere Server zu verteilen, um Lastspitzen und lange Wartezeiten zu vermeiden.

2.5 Das Track Konstrukt im WebAS Java

Parallel zum Änderungsauftrag im SAP ABAP Stack findet sich zunächst keine Entsprechung im Java Stack. Programmänderungen werden in einem zentralen Versionsrepository erfasst. Zur Verwaltung hat SAP das *Track Konstrukt* geschaffen, das sich als ein Container von Speichern und Laufzeitsystemen oder aber auch als Workflow verstehen lässt. Unabhängig von diesen beiden Betrachtungsweisen referenziert jeder Track auf eine oder mehrere Softwarekomponenten, die in dem entsprechenden Track entwickelt werden. Zusätzlich kann innerhalb eines Tracks auf abhängige Softwarekomponenten referenziert werden, die zu bestimmten Phasen vorhanden sein müssen. Diese Abhängigkeiten können beispielsweise Bibliotheken darstellen, die zur Kompilierung notwendig sind, oder aber bei einer geschichteten Softwareentwicklung grundlegende Softwarebibliotheken auf die zugegriffen wird. Die Konfiguration und das Festlegen dieser Referenzen geschieht im *System Landscape Directory*.

Nebenstehende Abbildung zeigt hierbei schematisch das Track Konstrukt. In dem konkreten Beispiel wird in Track 1 die Komponente 1 verwaltet, wobei beispielhaft Entwicklungskomponente 1 als Voraussetzung dient. Ein Track lässt sich als ein Workflow verstehen, der die Stati Entwicklung, Konsolidierung, Assembly, Test, Genehmigung und Produktion beinhaltet, die strikt linear durchlaufen werden müssen. Im Rahmen dessen werden die Zustände der Sourcen verwaltet. Dadurch besitzt ein Track im Gegensatz zu einem SAP Änderungsauftrag keinen eigenen Zustand besitzt, da sich zu jedem Zeitpunkt verschiedene Sourcecodes in verschiedenen Stati befinden können.

Den Schritten Entwicklung, Konsolidierung, Test und Produktiv kann dabei jeweils ein physikalisches WebAS Java System zugeordnet werden. Auf diesem kann zu einem Zeitpunkt immer nur eine Version der Softwarekomponente ausgeführt werden, wobei der Track die automatische Installation der der im entsprechenden Schritt aktuellsten Softwarekomponente steuern kann.

Um komplexe Systemlandschaften oder Entwicklungsszenarios abzubilden

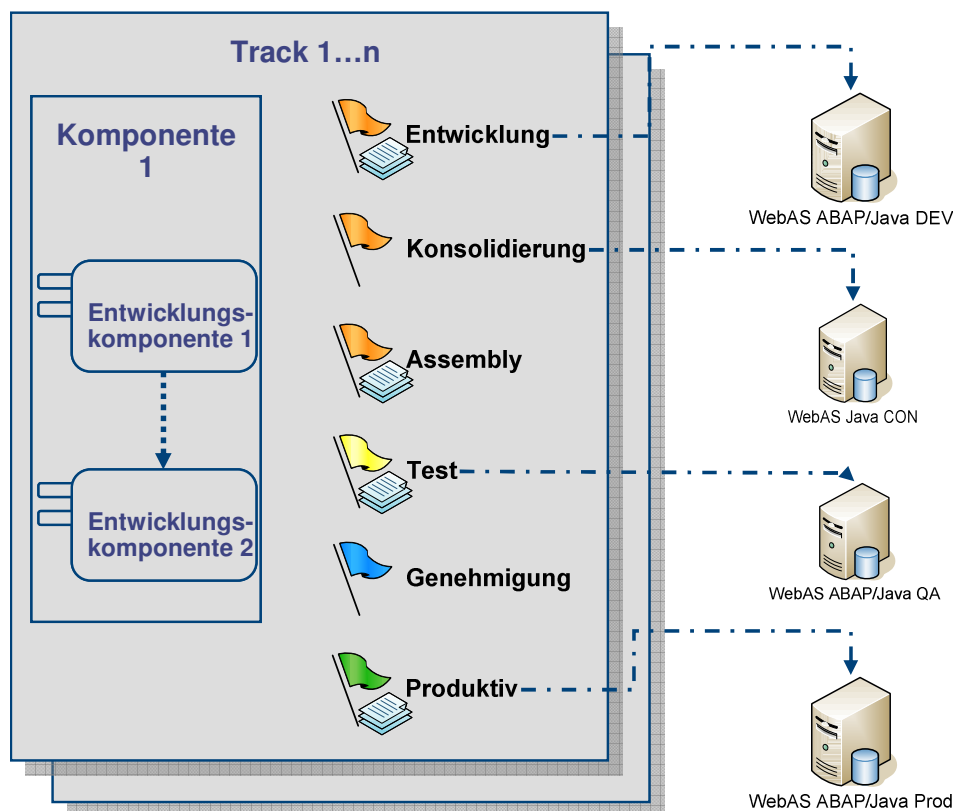


Abbildung 2-2: Schematische Darstellung des Track Konstruktes

kann dabei ein Track mit einem anderen verbunden werden. Dafür stehen zwei festdefinierte Arten, *Reparaturroute* und *Nachfolgeroute* zur Verfügung. Eine Reparaturroute übernimmt hierbei die Weiterleitung aller Change Requests, die den Status Genehmigung passiert haben in den Zustand Entwicklung des anderen Tracks. Dort muss vom jeweiligen Entwicklungsverantwortlichen entschieden werden, ob die Änderung importiert wird, oder aber verworfen werden kann. Bei einer Nachfolgeroute werden alle Sourcen, die in die Queue Produktiv importiert wurden, in den Status Entwicklung des nachfolgenden Tracks weitergeleitet, sollten dort jedoch nicht mehr modifiziert werden.

Im Rahmen einer releaseorientierter Softwareentwicklung empfiehlt es sich, je Releaselevel einen Track zu definieren. Die Wartung eines Release wird im gleichen Track vorgenommen, wobei jedoch Änderungen aufgrund von Fehlern im Nachfolgerelease nachge-

zogen werden müssen. Dies kann dadurch geschehen, dass beide Tracks durch eine Reparaturroute miteinander verbunden werden. Ist dies der Fall, so werden alle Änderungen, die im Wartungstrack stattfinden automatisch in die Importqueue des Nachfolgetracks kopiert. Der jeweilige Entwicklungsverantwortliche des Folgerelease hat zu Entscheiden, ob die Änderung relevant ist und nachvollzogen werden muss oder verworfen werden kann.

Im Fall einer geschichteten Softwareentwicklung müssen Folgetransporte definiert werden, so dass die Existenz von Softwarekomponenten, die innerhalb eines anderen Tracks als Abhängigkeiten referenziert werden, sichergestellt ist.

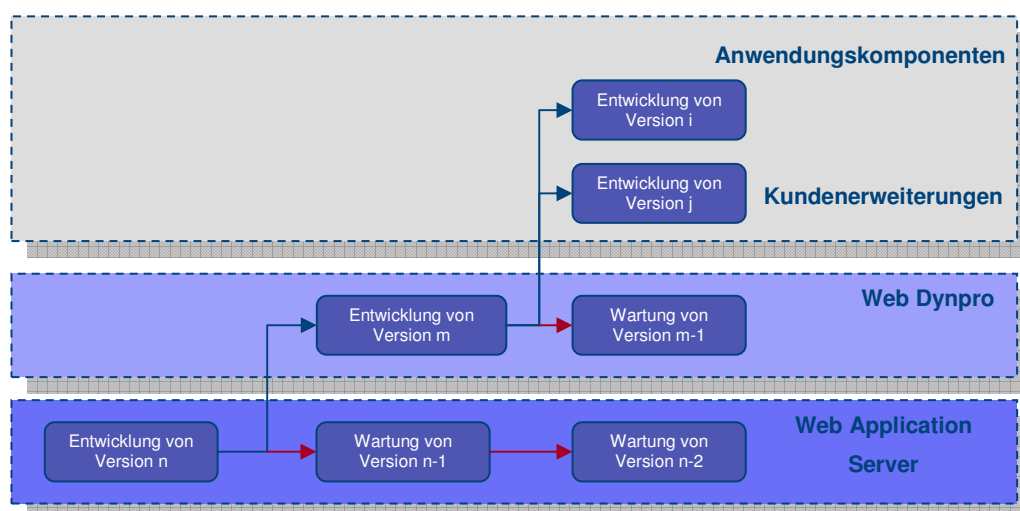


Abbildung 2-3: Softwareschichten im Entwicklungsprozess

Weitere Änderungen an der Software sollen auf im selben Track implementiert werden, so dass in diesem Kontext der Track auch als ein Container von Systemen und Speichern verstanden werden kann. In einer geschichteten Softwareentwicklung steht nach einer Genehmigung des Webarchivs der Sourcecode zum Import in den darauf aufbauenden Track zur Verfügung, sofern dies als Follow-On Track Verbindung eingestellt wurde. Mittels dieses Konzepts wird die komponentenbasierte Entwicklung ermöglicht und dafür gesorgt, dass die Arbeiten an der aufbauenden Komponente erst dann beginnen können, wenn die darunter liegende Komponente erfolgreich den Qualitätssicherungsprozess bestanden hat.

Im Gegensatz zum klassischen WebAS ABAP ist es mit dem WebAS Java möglich, aufeinander aufbauende Releases zu definieren. Dies geschieht durch die Definition von unterschiedlichen Versionen einer Softwarekomponente im SLD. Für jede Versionsnummer wird im CMS ein Track definiert. Wichtig ist hierbei, dass diese beiden, zunächst unabhängig erscheinenden Tracks über eine Reparaturroute miteinander verbunden werden. Damit wird sichergestellt, dass eine durch einen Fehler hervorgerufene Aktivität und damit ver-

bundene Änderung von Sourcecode auch in der Importqueue des Nachfolgereleases erscheint. In diesem Release ist dann von einem Entwickler zu prüfen, ob der Fehler auch in der aktuellen Konstellation auftritt und mit der eingespielten Aktivität zu beheben ist oder die Aktivität verworfen werden kann. Im Gegensatz zu der Follow-On Route wird bei einer Reparaturroute der Nachfolgetrack mit dem Change Request versorgt, sobald er in das Konsolidierungssystem importiert wurde.

Hierbei gibt es zwei Möglichkeiten. Innerhalb einer Schicht, beispielsweise Web Application Server, sind die Tracks über Reparaturrouten miteinander verbunden. Aufeinander aufbauende Schichten werden durch Follow-On Routen verbunden. Damit entsteht bereits durch einige wenige Tracks ein komplexes Gefüge von Verbindungen, die gewartet werden müssen.

Eine derartige Logik zum Management unterschiedlicher Releases wird bisher vom WebAS ABAP nicht unterstützt und muss bei einer Integration entsprechend mit bedacht und abgebildet werden. Problematisch erscheint an dieser Stelle die Definition der Reparatur- und Belieferungsrouten, da diese Trackverbindungen in einer Weboberfläche definiert werden. In einer großen IT-Landschaft kann man davon ausgehen, dass mehr als 100 Tracks für Softwarekomponenten definiert wurden. Möchte man hier auch die eben skizzierte Folgereleaselogik der JDI verwenden, so befinden sich mehr als 200 Tracks im CMS, für die händisch die Verbindungen gepflegt werden müssen. Hier ist zu beachten, dass im üblichen Softwareentwicklungsprozess nicht von Anbeginn an alle Softwarekomponenten und Versionsstände im SLD erfasst werden, sondern dies erst dann geschieht, wenn die Implementierung und Entwicklung eines neuen Release ansteht. Da hierbei auch die Trackverbindungen manuell zu pflegen sind, wird es sehr komplex. Übersieht man dabei, eine Verbindung vergessen manuell zu pflegen, so kann dies zu Inkonsistenzen im Entwicklungsprozess führen.

Ein Beispiel für ein derartiges Szenario liegt darin, dass in einem aktuellen Release 1.0 Fehler auftreten, die zu Änderungen führen, die Fehler auch in einem, sich aktuell in der Entwicklung befindlichen Release 2.0 befinden aber die Reparaturroute nicht definiert wurde. Bei großen Entwicklungsteams die an den unterschiedlichen Releases arbeiten, kann diese Konstellation bei einem Deployment des Nachfolgereleases 2.0 dazu führen, dass Fehler, die zwischenzeitlich als behoben galten wieder auftreten. Diese Problematik kann durch eine intelligente Logik, die automatisiert die Verbindungen zwischen Tracks festlegt behoben werden.

Problematisch gestaltet sich die Definition der Abhängigkeiten zwischen den einzelnen Tracks. Diese können zwar über ein Webinterface festgelegt werden, jedoch die Darstel-

lung ist sehr unübersichtlich, so dass Fehlkonfigurationen, die zu Inkonsistenzen führen können, nur schwer zu finden sind.

Aufgrund der technischen Konfiguration von Verbindungen innerhalb eines Tracks und des Genehmigungssystems kann eine Instanz eines Tracks mit einer Ausprägung des *Transport Management System* Konfiguration im WebAS ABAP verglichen werden. Dabei muss jedoch beachtet werden, dass sich trotz alledem die Transportsteuerung im WebAS Java grundlegend vom ABAP Stack unterscheidet.

2.6 Technischer Überblick zum Change Management Service

Der Change Management Service nimmt in der JDI eine zentrale Rolle ein, da er das Deployment von Archiven auf den verschiedenen Systemen und die gesamte Steuerung und Verwaltung der JDI durch den Benutzer übernimmt. Im Rahmen dieser Aufgaben werden über den CMS neue Tracks angelegt, Transportrouten definiert, Importe in Folgesysteme durchgeführt und Genehmigungen für Softwareänderungen erteilt. Der CMS greift dazu auf die bisher beschriebenen Dienste zu und wird seitens des Benutzers über ein Webinterface gesteuert. Während die bisher genannten Services im Rahmen der Java

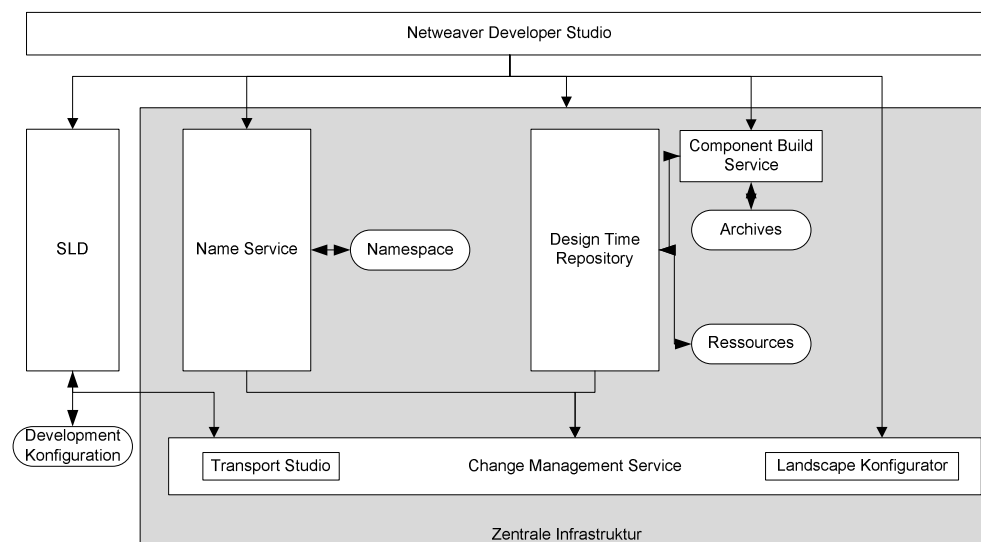


Abbildung 2-4: Aufbau der NWDI

Development Infrastructure eher als Hilfswerkzeuge angesehen werden können, handelt es sich beim Change Management Service um die eigentliche Logik zur Verwaltung von Änderungen und Steuerung der Java Development Infrastructure. Um die Änderungen konsistent verwalten zu können, muss der Change Management Service bei Änderungen im *System Landscape Directory* stets aktualisiert werden.

Die Abbildung 2-4 veranschaulicht hierbei die zentrale Rolle des Change Management Services.

3 JDI Entwicklungsprozessbeschreibung

Für die Beschreibung des Softwareentwicklungsprozesses in der JDI wird von einer projekttypischen Implementierung einer Softwarekomponente ausgegangen, die nach dem erfolgreichen Deployment in den Wartungszyklus übergeht. Der Hersteller, die Software und die davon abgeleitete Komponente werden zunächst im *System Landscape Directory* definiert. Dabei werden bei der Definition auch Referenzen auf die Klassen angegeben, die zum Zeitpunkt der Übersetzung auf dem System vorhanden sein müssen. Üblicherweise sind dies die Bibliotheken gegen die übersetzt wird. Sollte zur Laufzeit die Software von anderen Komponenten abhängig sein, so wird dies ebenfalls zunächst im SLD definiert. Damit nun aufbauend auf dieser Definition ein Track erstellt wird, in dem die Software entwickelt werden kann, wird das Webinterface des CMS aufgerufen.

Um für die eben definierte Komponente einen Track zu erstellen, muss der Change Management Service zunächst mit den Daten des System Landscape Directory synchronisieren. Dieser Prozess muss manuell angestoßen werden. Nach einem Abgleich des CMS mit dem SLD kann der Track für diese Softwarekomponente angelegt werden. Dabei werden automatisch die definierten Abhängigkeiten miteinbezogen.

Probleme dieser Definition entstehen nun darin, dass beim Anlegen verschiedener Tracks die Oberfläche zur Pflege physischer Systeme, die im jeweiligen Trackstatus versorgt werden können, sehr unübersichtlich erscheint. Ist der Track angelegt, müssen zusätzlich die, zur ‚Build-Time‘ erforderlichen Softwarekomponenten, importiert werden. Dies ist notwendig, damit im Konsolidierungssystem eine Übersetzung und Bündelung von Anforderungen zu einem Webarchiv durchgeführt werden kann und die dafür benötigten Java Bibliotheken in diesem Zeitpunkt auf dem System vorhanden sind. Sobald sich der Track im Zustand *Entwicklung* befindet, können zugeordnete Entwickler über das SAP Netweaver Developer Studio¹ auf das DTR zugreifen und die Sourcen mit Sperre oder aber optimistisch, ohne Sperre auschecken. Sobald der Quellcode geändert wird, muss diese Änderung einer Aktivität zugeordnet werden. Im Gegensatz zu einer Entwicklung mit einem Sourceverwaltungssystem wie CVS oder Subversion können hier nur ganze Aktivitäten wieder eingchecked und damit verfügbar gemacht werden. Der Vorteil besteht nun darin, dass unter einer Aktivität alle dazu geänderten Bestandteile ersichtlich sind.

¹ Hierbei handelt es sich um eine von SAP angepasste Eclipse Entwicklungsumgebung. Das Eclipse Framework kann unter <http://www.eclipse.org> bezogen werden.

Sobald eine Aktivität wieder eingecheckt wurde, kann dieser Aktivität keine Änderung mehr hinzugefügt werden. Diese Vorgehensweise ist vergleichbar mit der Freigabe einer CTS-Aufgabe im WebAS ABAP. Die Aktivität wird nach dem einchecken zunächst im Workspace Verzeichnis ‚inaktiv‘ abgelegt, aus der Entwicklungsumgebung heraus kann nun ein Build angestoßen werden. Sollte dieser erfolgreich sein, ist die Änderung öffentlich und im Verzeichnis aktiv verfügbar und wird auch auf dem J2EE Server deployed. Diese freigegebenen Aktivitäten können dann jeweils zu Change Requests gebündelt werden, die in der Import Queue des Konsolidierungssystems auflaufen, importiert und auf dem Konsolidierungsserver deployed werden. Auf dem Konsolidierungssystem kann bei erfolgreichem Import eine softwarekomponentenspezifische Teilmenge der bereits importierten Change Requests ausgewählt werden, die zu einem Webarchiv zusammengestellt wird. Zu diesem Zeitpunkt müssen nun die Bibliotheken, die im Track zur ‚Build-Time‘ referenziert wurden, auf dem Konsolidierungssystem verfügbar sein. Sobald die Erstellung des Webarchivs erfolgreich war, erscheint die Datei in der Import Queue des Qualitätssicherungssystems. Nach einem erfolgreichen Import und entsprechendem Deployment, kann nun dieses Archiv mit den darin beinhalteten Change Requests von einem Qualitätsmanager mittels einer elektronischen Unterschrift genehmigt und damit freigegeben werden. Dadurch erscheint nun das Webarchiv in der Import Queue des Produktivsystems und kann produktiv gesetzt werden. Da bei der Erstellung des Tracks lediglich ein Produktivsystem definiert werden kann, kann das Webarchiv direkt aus dem Dateisystem genommen und in einen beliebigen WebAS Java importiert werden, sofern die Referenzen des Tracks auf dem Zielsystem vorhanden sind. Möglich ist es jedoch ebenso, mittels Nachfolgerouten von Tracks mehrere Produktivsysteme abzubilden.

