Softwareverteilung und -bereitstellung im Internet und Netzwerk auf Basis der J2EE-Architektur

Verfasser

Fokko Erdmann

Mat.-Nr: 1153462

Universität Lüneburg

Fachbereich Automatisierungstechnik

Ingenieur-Informatik

Erstprüfer: Prof. Dr. rer. nat. Dipl.-Inform. Helmut Faasch

Zweitprüfer: Prof. Dr.-Ing. Dipl.-Inform. Eckhard C. Bollow

Fachliche Betreuung: Dipl. Physiker Klaus Hollstein

Kurzfassung

In dieser Diplomarbeit wird untersucht, wie sich ein Softwareverteilungssystem auf Basis der J2EE-Architektur realisieren lässt. Dafür werden alle Prozesse und Personen für solch ein Szenario ermittelt und benannt. Die in einem Softwareverteilungssystem benötigten Komponenten werden aufgezeigt und die Beziehungen zwischen ihnen untersucht. Bei der Untersuchung soll beachtet werden, dass das Verteilungssystem mit jeder Art von Software umgehen kann. Die Abstraktion der Verteilung ist also ein wichtiger Bestandteil der Modellierung.
Die Diplomarbeit unterteilt sich dabei in drei Bereiche. Den Hauptteil nimmt die Untersuchung ein. Im zweiten Teil wird zur Unterstützung der Untersuchung ein Prototyp eines solchen Systems entworfen. Im dritten Teil sollen die im ersten und zweiten Teil ermittelten Ergebnisse getestet werden.

abstract

This thesis analyses how to realize a software distribution system based on the J2EE-Architecture. For this purpose, all processes and persons for such a scenario are determined and named.
Special attention should be given to the aspect, that the distribution system has to be able to handle all kinds of software. Therefore, the abstraction of the distribution is an important element of the systemmodel.
The thesis is devided in three parts. The main part is the analysis. In the second part a prototype of a distribution system is created to assist the analysis.
The determined results of the analysis are tested and validated during the third part of this thesis.

Inhaltsverzeichnis

1 Einleitung

Der Begriff Softwareverteilung kann unterschiedlich verwendet werden. Die Bereitstellung von Installationsdateien im Internet ist Softwareverteilung. Die Verteilung von CD-ROM während einer Messe ist zum Beispiel auch Softwareverteilung.
In dieser Diplomarbeit wird der Begriff genutzt in Bezug auf die Verteilung von Software in einem Netzwerk von zentraler Stelle aus.
Der Administrator einer Firma oder einer Abteilung hat die Möglichkeit, von einer zentralen Position aus die Arbeitsstationen der Mitarbeiter mit Software zu bestücken. Er bedient sich dabei aus einem Pool von Software. Danach wählt er aus, auf welchen Arbeitsstationen die Software installiert werden soll.
Software ist mit den Jahren komplexer und die Computernetze sind immer größer geworden. Deswegen müssen bei Vertrieb und Installation der Programme einige Dinge bedacht werden: Client-Server-Abhängigkeit, Grundvoraussetzungen (Java, Visual Basic Bibliotheken, DirectX etc.), Plattformtauglichkeit, Systemvoraussetzungen, Lizenzierung, Sicherheit, Versionsabhängigkeit usw.
Diese Diplomarbeit widmet sich dem Thema Softwareverteilung und -bereitstellung im lokalen Netz und im Internet.
Die Nutzung eines Verteilungssystems über das lokale Unternehmensnetzwerk hinaus hat den Vorteil, dass dadurch die Vernetzung unter den Unternehmen zunimmt. Informationen können aufgrund derselben Plattform sogar weltweit ausgetauscht werden. Dadurch wird die globale Kommunikation zwischen den Unternehmen gefördert.

2 Untersuchung

2.1 Systemidee

Das Ziel ist die Entlastung von Softwareentwicklern, Administratoren und Anwendern. Dazu soll die Verteilung und Installation von Software in Firmen vereinfacht werden. Die zu installierende Software wird von zentraler Stelle aus auf die Plattformen[1] verteilt und installiert. Als Plattformen kommen sowohl Server, Arbeitsstationen als auch Programme in Frage.
Zur Unterstützung der Vernetzung von Unternehmen soll das zu entwickelnde Verteilungssystem auch über die Grenzen des Unternehmens hinweg einzusetzen sein. Dabei sollen im Internet mehrere so genannte Depots entstehen, wo die Software gelagert wird. Von dort können die Installationspakete dann bezogen werden.
Das Verteilungssystem soll in der Lage sein, Software auf Plattformen zu de-/installieren und zu aktualisieren. Die Möglichkeit, eine vollständige Konfiguration von Software vorzunehmen, wird nicht Gegenstand dieser Arbeit sein und wird nicht im Verteilungssystem integriert.

2.2 Systemanalyse

Die Analyse des Verteilungssystems soll ermitteln, welche Prozesse in dem System stattfinden können und welche Personen dabei involviert sind.

2.2.1 Rollen des Verteilungssystems

Innerhalb des Verteilungssystems nimmt jede involvierte Person eine Rolle ein. Jede Rolle hat ihren Aufgabenbereich. Dabei kann die Tätigkeit einer Rolle von einer Person oder mehreren durchgeführt werden. Letzteres kann der Fall sein, wenn der Aufgabenbereich

[1] In der Informatik werden Betriebssysteme wie Windows, Linux, Unix etc. als Plattformen bezeichnet. In diesem Fall sind mit Plattformen alle Orte gemeint, wo Software installiert werden kann.

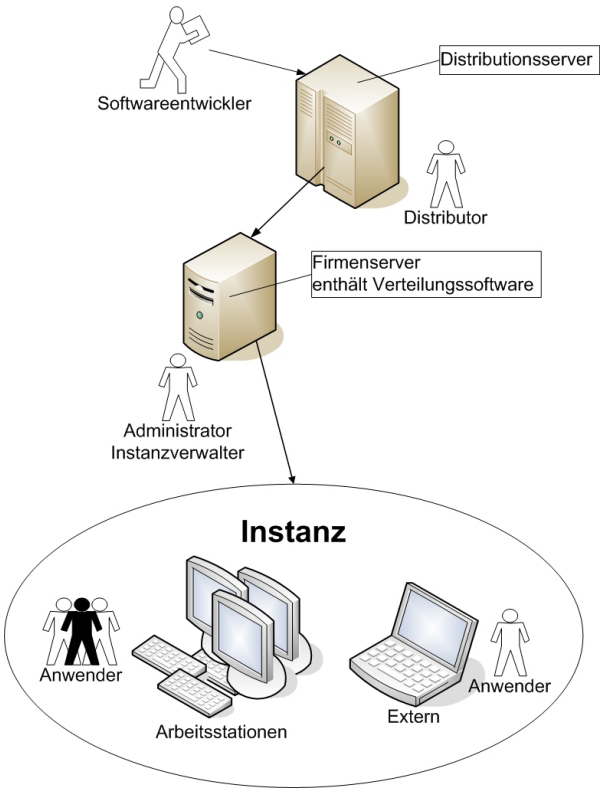

Abbildung 2.1: Infrastruktur

für eine Person evtl. zu umfangreich ist. In kleineren Firmen kommt es allerdings des Öfteren vor, dass eine Person mehrere Rollen übernimmt, weil es an Personal mangelt oder die Aufgaben den Arbeitstag der Person nicht ausfüllen würden.

2.2.2 Aufgaben

Der **Softwareentwickler** erstellt die Software, die verteilt wird. Er hat die Aufgabe, die Korrektheit seiner Software zu gewährleisten oder bei Bedarf schnellstmöglich ein Update zu liefern.
Der **Distributor/Paketierer** stellt die Software des Entwicklers zu Paketen zusammen. Pakete sind Einheiten, die zwischen den Depots verschickt werden. Da „das Rad doch oft wieder erfunden wurde“, gibt es meist verschiedene Software, die die gleiche Funktionalität bietet. Die eine kann dies besser, die andere schlechter. Auch arbeitet manche Software nicht mit einer anderen zusammen. Eine gute Zusammenstellung zu finden und bereitzustellen ist die Hauptaufgabe des Distributors/Paketierers.
Außerdem synchronisiert er das Depot mit denen im Internet und stellt den Instanzverwaltern dann die Software aus dem Depot für die Installation zur Verfügung.

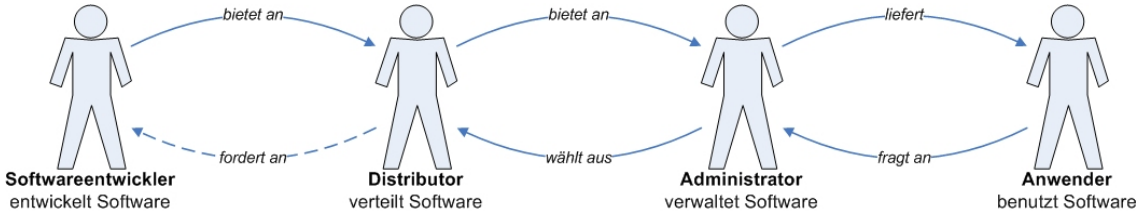

Abbildung 2.2: Rollen

Der **Administrator** oder auch **Instanzverwalter** konfiguriert die Instanzen der Firma und stößt die Installationen an. Er hat somit die Entscheidungsgewalt, welche Software für den Anwender installiert wird. Allerdings kann er nur auf Software zurückgreifen, die ihm der Distributor zur Verfügung stellt. Er muss auch dafür sorgen, dass der Anwender eine funktionierende Arbeitsumgebung hat. Dabei ist er auf die Rückmeldung von Anwendern angewiesen. Nur so können Probleme aufgedeckt und schnellstmöglich beseitigt werden.
Der **Anwender** arbeitet mit der installierten Software. Diese Rolle hat am wenigsten zu leisten. Dennoch ist sie sehr wichtig, denn allein für den Anwender wird die Software installiert. Er nutzt sie und benötigt eine einwandfreie und funktionstüchtige Installation, denn Anwender sind für den produktiven Teil in einer Firma zuständig. Arbeitskräfte sind teuer und deshalb möchte man, dass sie ohne Probleme arbeiten können, um ihren Dienst für die Firma zu leisten.

2.2.3 Abläufe im System

Die Aufgaben der einzelnen Rollen können als konkrete Abläufe im Verteilungssystem beschrieben werden. Dadurch kann ermittelt werden, welche Funktionen das Verteilungssystem bieten muss. Auch hilft es bei der Implementierung der Funktionen, weil die in der Analyse ermittelten Ergebnisse mit denen der realisierten Funktionen übereinstimmen müssen.
Die nachfolgende Auflistung zeigt die Abläufe im System und die daran beteiligten Personen.

Einstellung eines neuen Softwarepaketes

Akteure: Softwareentwickler, Distributor

Beschreibung: Eine neue Software wurde erstellt und wird dem Verteilungssystem zur Verfügung gestellt.

Auslöser: Der Softwareentwickler liefert eine neue Software.

Ergebnisse: Die Software steht im Verteilungssystem bereit.

Konfiguration einer Plattform

Akteure: Instanzverwalter

Beschreibung: Eine Plattform wird einer Instanz zugefügt. Das heißt, alle nötigen Konfigurationen werden durchgeführt.

Auslöser: nicht konfigurierte Plattform

Ergebnisse: Plattform ist im Verteilungssystem verfügbar und einsatzbereit.

Installation einer Software

Akteure: Instanzverwalter

Beschreibung: Eine Software wurde zur Installation ausgewählt und wird auf der entsprechenden Plattform installiert.

Auslöser: Die Software wurde ausgewählt.

Ergebnisse: Die Software ist auf der Plattform installiert.

Deinstallation einer Software

Akteure: Instanzverwalter

Beschreibung: Eine Software wurde zur Deinstallation ausgewählt und wird auf der entsprechenden Plattform deinstalliert.

Auslöser: Software wird nicht mehr benötigt.

Ergebnisse: Software ist deinstalliert.

Update einpflegen

Akteure: Softwareentwickler, Distributor, Instanzverwalter

Beschreibung: Ein Softwareupdate liegt vor. Der Administrator integriert das Softwareupdate in das Verteilungssystem und die Installationen werden aktualisiert.

Auslöser: Der Softwareentwickler liefert ein Update für seine Software.

Ergebnisse: Alle Plattformen mit der installierter Software sind durch das Update aktualisiert worden.

Depots synchronisieren

Akteure:	Administrator
Beschreibung:	Das eigene Depot wird auf den Stand eines anderen gebracht.
Auslöser:	Der Administrator startet die Synchronisierung.
Ergebnisse:	Das eigene Depot enthält nun alle Software des anderen Depots.

2.3 Anforderungen

Aus den beschriebenen Prozessen resultieren Anforderungen an das Verteilungssystem. Dazu gehören Anforderungen, die auf jeden Fall erfüllt werden müssen, aber auch solche, die für weiterentwickelte Versionen interessant sein können.

Zu den Pflichtanforderungen gehören:

- Die Integration einer Software in das Verteilungssystem ist nur dann erfolgreich, wenn alle Daten (Installationsdateien, Informationen zur Software) sicher übertragen und abgelegt wurden und die Informationen in keinem Konflikt stehen mit bereits integrierter Software.
- Es kann nur Software installiert werden, wenn mindestens eine konfigurierte Plattform im System existiert.
- Eine Software darf nur (de)installiert werden, wenn die Voraussetzungen für eine (De)Installation erfüllt sind.
- Eine Installation ist nur dann erfolgreich, wenn die installierte Software auf der Plattform funktioniert.
- Die Synchronisierung des Depots kann jederzeit abgebrochen werden, ohne dass das System in einen inkonsistenten Zustand gerät.
- Die Synchronisierung des Depots kann jederzeit erneut durchgeführt werden.

Vorschläge:

- Eine Software kann nur integriert werden, wenn sie von dem Softwareentwickler durch ein Zertifikat signiert worden ist.

- Die Integration einer Software in das Verteilungssystem ist nur dann erfolgreich, wenn die Signierung der Software mit einem gültigen Zertifikat durchgeführt worden ist
- Ein Update kann nur integriert werden, wenn es von demselben Softwareentwickler signiert worden ist wie die Software, für die das Update bestimmt ist, oder wenn der Softwareentwickler explizit eine Erlaubnis erteilt hat an denjenigen, der das Update liefert.

2.4 Komponenten des Verteilungssystems

Neben den Rollen betrifft die Verteilung auch mehrere Komponenten. Dies sind sowohl abstrakte als auch reale Begriffe. Das System kann auch in diese beiden Ebenen aufgeteilt werden – siehe Abbildung 2.3.
Die linke Seite in der Abbildung zeigt das Depot, hier Repository[2] genannt. Es liefert die Software für den Instanzverwalter und die Anwender auf der rechten Seite. Dort wird die Software installiert, darum ist dieser Bereich als Installation bezeichnet.

2.4.1 Repository (real)

Der Softwareentwickler arbeitet auf der realen Ebene. Er entwickelt seine Software und stellt sie dem Repository zur Verfügung. Die von den Entwicklern angefertigten Module werden bei einer Installation auf den Plattformen installiert.

2.4.2 Repository (abstrakt)

Auf der abstrakten Ebene werden die Module zu **Features**. Ein Feature gibt Auskunft darüber, was das Modul für Fähigkeiten besitzt. Die Informationen sind zusammen mit den Modulen in einem Paket verpackt. Die Pakete werden im Repository ablegt. Die Information in den Paketen stehen in einer Beschreibungsdatei. Dort sind zudem Abhängigkeiten, die zwischen einzelnen Modulen, Features und Plattformen existieren, eingetragen. Die Abhängigkeiten werden im Abschnitt 2.5 näher erläutert.

[2] Der Begriff Repository ist die englische Bezeichnung für Depot. Für die Programmierung werden meist englische Begriffe verwendet. Diese sind oft kürzer und es lassen sich meist prägnantere Wörter finden als im deutschen Sprachgebrauch. Außerdem erleichtert die Verwendung der englischen Sprache die internationale Zusammenarbeit, falls das Projekt weitergeführt wird.

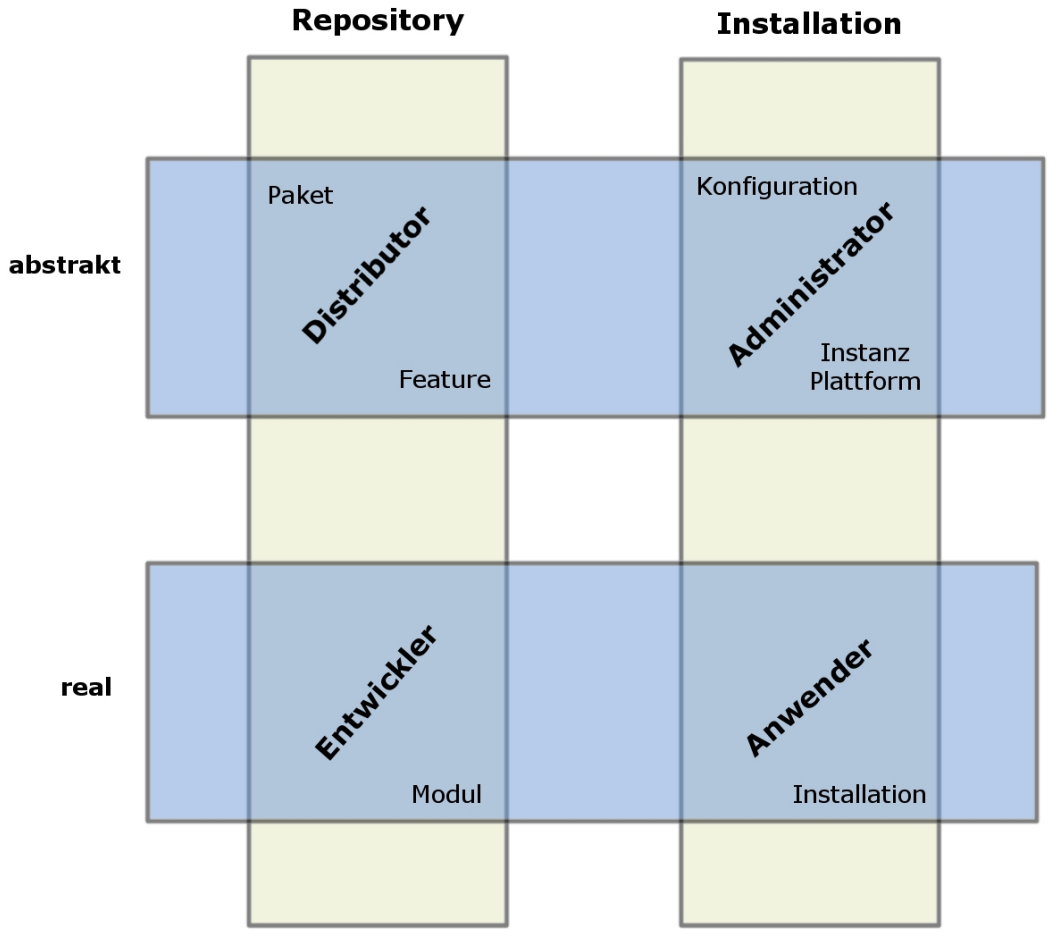

Abbildung 2.3: Bereiche des Verteilungssystems

2.4.3 Installation (abstrakt)

Eine abstrakte Installation sieht so aus, dass der Administrator/Instanzverwalter aus einer Liste an Features, die der Distributor ihm zur Verfügung stellt, auswählt. Seine Auswahl stellt eine Konfiguration dar. Er gibt damit an, welche Features er auf der Instanz installieren will.
Eine **Instanz** ist eine Zusammenstellung von Plattformen. Die **Plattformen** werden in einer Instanz so konfiguriert, dass sie aufnahmefähig sind. Das bedeutet, bekommen sie ein Modul zur Installation geliefert, dann wissen sie damit umzugehen und installieren es am richtigen Ort. Diese Installationsorte für Software können variabel sein. Dies kann ein Webserver wie Tomcat sein, auf dem eine Webanwendung installiert wird, oder eine Netbeans-Plattform, welche durch Hinzufügen von Modulen weitere Funktionalität erhält.

2.4.4 Installation (real)

Aus der Konfiguration resultiert ein Installationsprozess. Es wird eine Berechnung durchgeführt, die ermitteln soll, welche Module des Repositorys benötigt werden, um nach deren Installation, die gewünschten Features abzudecken.
Die ermittelte Liste an Modulen wird dann auf den Plattformen der Instanz installiert. Ein installiertes Modul auf einer Plattform wird durch den Begriff **Installation** definiert.

2.4.5 Vorteil der Abstraktion

Die Einführung einer abstrakten und einer realen Ebene hat folgende Vorteile. Der Instanzverwalter braucht sich keine Gedanken darüber zu machen, welche Software er auf welcher Plattform installieren kann. Das System übernimmt diese Aufgabe. Der Instanzverwalter konfiguriert lediglich die Instanz mit Features, die Installation der Module erfolgt dann in der realen Ebene. Es kommt auch nicht darauf an, welche Art von Software verteilt wird, denn solange eine Plattform zu dem Modul existiert, welches verteilt wird, kann es auch installiert werden. Man ist somit gänzlich unabhängig und könnte auf diese Weise anstelle von Software auch Dokumente oder sonstige Daten verteilen.

2.5 Abhängigkeiten zwischen den Komponenten

Software ist heutzutage komplexer als je zuvor. Dies bedeutet, dass Software nicht mehr einfach nur installiert wird, sondern oftmals Zusatzbedingungen erfüllt werden müssen. Es macht zum Beispiel wenig Sinn, eine Client-Anwendung zu installieren, wenn der dazugehörige Server fehlt. Oder es liegen nicht die entsprechenden Hardwarevoraussetzungen vor. Dann kann eine Software nicht genutzt werden, weil z.B. kein Scanner vorhanden ist.
In dieser Diplomarbeit wurde untersucht, welche Abhängigkeiten zwischen Software und Hardware existieren können. Von der Menge der ermittelten Abhängigkeiten stellten sich drei als die wichtigsten heraus: require[3], deny[4], update[5].

[3] require (engl.): bedingen, benötigen, brauchen
[4] deny (engl.): verweigern, leugnen
[5] update (engl.): aktualisieren

Diese Abhängigkeiten werden in einer Beschreibungsdatei eingetragen, welche dem Paket beiliegt.
Die Abhängigkeiten können sowohl auf der realen Ebene, also zwischen den Modulen, als auch zwischen den Features auf der abstrakten Ebene existieren.
Alle Abhängigkeiten, die nicht diesen drei angehören, müssen aber auch beachtet werden. Abschnitt 2.5.4 erklärt, wie auch diese im System abgebildet werden können.

2.5.1 Modul-/Featureabhängigkeiten

Es gibt zunächst die Require-Abhängigkeit. Manche Software ist für sich allein nicht funktionstüchtig und benötigt zusätzliche Software. Dies lässt sich am besten an einem Beispiel erläutern:
Ein Programm stellt einen News-Server dar, bei dem sich Clients einloggen können. In diesem Fall greift bei der Installation des Clients die Abhängigkeit und verlangt, dass der Server auch installiert wird.

Die Deny-Abhängigkeit erwirkt das genaue Gegenteil. Auch diese Abhängigkeit lässt sich leicht an einem Beispiel erklären:
Ein Server benötigt ein Programm, welches E-Mails verschicken kann. Im Repository existieren zwei Module, die diese Funktionalität bieten. Es ist nicht sinnvoll, beide zu installieren, im schlimmsten Fall würden dann die E-Mails doppelt verschickt werden. Um dies zu verhindern, werden die beiden Module mit einer Deny-Abhängigkeit belegt, die das jeweilige Konkurrenzprodukt ausschließt.

2.5.2 Plattformabhängigkeiten

Diese Abhängigkeit ist bestimmend für die Umgebung, wo eine Software installiert werden kann. Für manche Software sind gewisse Voraussetzungen nötig. Diese kann nur eine bestimmte Plattform bieten. Auch hier ein Beispiel:
Ein Entwickler hat ein Plugin für den Firefox-Browser entwickelt. Die ist spezifisch für Firefox und würde im Internet Explorer nicht funktionieren. Die Bedingung für dieses Plugin ist also, dass es im Firefox installiert wird.

2.5.3 Updateabhängigkeiten

Als Updates werden Softwarepakete bezeichnet, die Mängel an ausgelieferter Software beseitigen oder die kleine Verbesserungen enthalten. Größere Veränderungen bzw. Ver-

besserungen an der Software werden hingegen als Upgrade[6] bezeichnet. Im allgemeinen Sprachgebrauch verschmelzen diese Begriffe oftmals und wenn von einem Update die Rede ist, kann es sich in Wirklichkeit auch um ein Upgrade handeln. Deshalb wird für jegliche Softwareaktualisierungen in diesem Verteilungssystem der Begriff Update benutzt. Updates kennzeichnen sich meist dadurch, dass sie eine höhere Versionsnummer als die alte Software besitzen. Dabei existieren unterschiedliche Konventionen. Manche erhöhen die Versionsnummer bei jeder Aktualisierung, andere führen Unternummern ein (Abbildung 2.2). Die Idee ist es, zusätzlich zu den anderen Abhängigkeiten auch Updateabhängigkeiten im Verteilungssystem einzuführen. Man kann nun völlig auf die Versionsnummern verzichten, denn es ist klar beschrieben, welches Modul ein anderes aktualisiert. Allerdings können dadurch andere Probleme auftauchen. Die Update-Abhängigkeiten lassen sich als ein gerichteter Graph darstellen. Gerichtet bedeutet, dass die Updates nur in eine Richtung erfolgen. Der Logik zufolge ist dies klar, da Updates immer eine neue Version bedeuten und nicht auf etwas Altes verweisen. Solch ein Graph besteht aus Kanten und Knoten. Knoten sind in diesem Fall die Module und Kanten die Beziehungen zwischen den Modulen.

alt	neu
1	2
1.0	1.1
1.0.0.0	1.0.0.1

Tabelle 2.2: Nummern

Nachteile gegenüber Versionsnummern

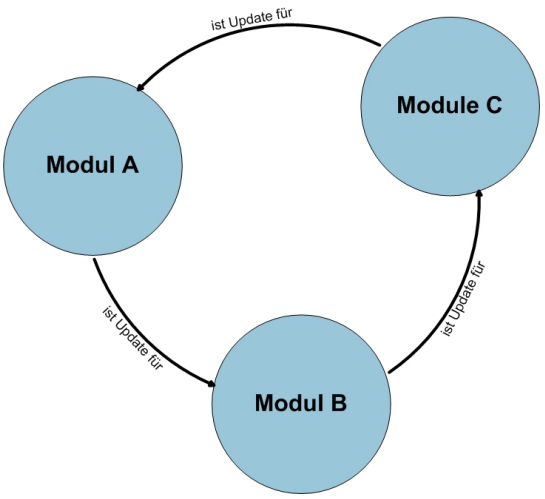

Abbildung 2.4: Zyklus

Da die Update-Abhängigkeiten beliebig Module miteinander verknüpfen können, kann es zu einem Kreisschluss(Zyklus) führen – siehe Abbildung 2.4. Wenn man ermitteln möchte, welches Modul denn nun das aktuellste ist, tritt ein Problem auf. Der Algorithmus findet kein Ende, da jedes Modul wiederum von einem anderen aktualisiert wird.
Die Bedingung für den Updatemechanismus im Verteilungssystem ist demnach, dass alle Updateabhängigkeiten zyklenfrei sind.
Man könnte denken, dass für die Require-Abhängigkeiten das gleiche Problem auftritt. Würden aber wie in Abbildung 2.4 alle Module sich gegenseitig benötigen, so werden sie alle drei installiert und die Abhängigkeiten sind damit erfüllt.

[6] upgrade (engl.): Aufrüstung, Verbesserung

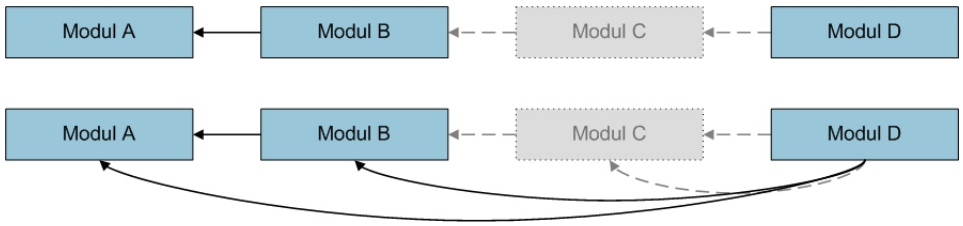

Abbildung 2.5: Fehlendes Glied in der Updatekette

Ein weiteres Problem tritt auf, wenn ein Knoten des Graphen fehlt. Man kann dann nicht mehr den Graphen entlanglaufen, um die neuste Version zu finden. Die Lösung dafür ist, dass das Update bis zu einer bestimmten Anzahl alle alten Module kennt, die es aktualisiert. So wie es der untere Graph in Abbildung 2.5 zeigt.

Vorteile gegenüber Versionsnummern

Durch die freie Wahl der Modulabhängigkeiten kann man ein Update entwickeln, welches nicht nur ein einziges Modul aktualisiert. Man hat also die Möglichkeit, mehrere Module gleichzeitig auf einen neuen Stand zu bringen. Die Versionsnummern erlauben es, nur eine bestimmte Software auf einen neuen Stand zu bringen. So muss zu jeder Software ein eigenes Update entwickelt werden.
Ein weiterer Vorteil ist, dass mehrere Entwickler ein Update für ein und dasselbe Modul entwickeln können. Der Instanzverwalter kann sich dann entscheiden, welches Update er installieren will.

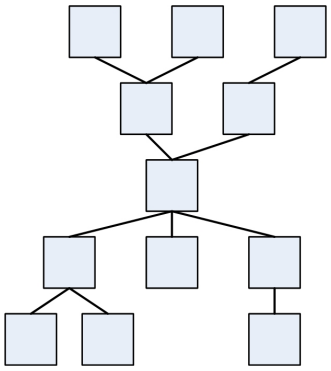

Abbildung 2.6: Updategraph

Der Updategraph, der dabei entsteht, kann sich aufteilen, aber auch wieder zusammenkommen. Abbildung 2.6 zeigt, wie ein solcher Graph aussehen kann.

2.5.4 Spezifische Abhängigkeiten

Die oben genannten Bedingungen werden zum einen vom Entwickler geliefert, können aber auch vom Distributor erstellt werden. Der Entwickler hat zudem die Möglichkeit, noch weitaus spezifischere Abhängigkeiten zu definieren. Diese werden direkt vom Modul über eine Validierungsmethode überprüft. Die Validierung findet nach der Installation der Module statt. Die Module sind dann in der Umgebung, in der sie arbeiten sollen. Schlägt dann die Validierung fehl, wird die komplette Installation rückgängig gemacht.

Dadurch dass sich das Modul in seiner Arbeitsumgebung befindet, können umfangreichere Bedingungen erstellt werden. So kann die direkte Zusammenarbeit mit anderen Modulen auf anderen Plattformen gefordert werden. Dies würde wie folgt aussehen:

(Module A auf Plattform 1) und ((Module B auf Plattform 2) oder (Module C auf Plattform 2)) = gültig

2.6 Berechnung für die Installation

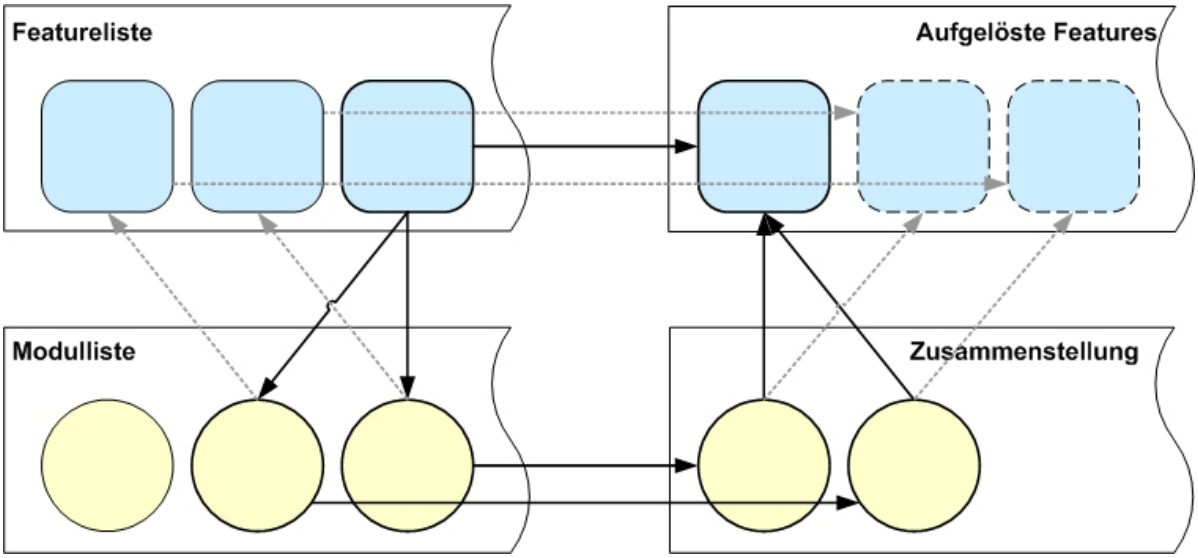

Abbildung 2.7: Berechnung

Die Installation von Software wird durch den Instanzverwalter veranlasst. Dieser entscheidet sich für Features, die er auf der Instanz installieren will. Die reale Installation passiert dann mit dem Installieren von den Modulen auf den Plattformen der Instanz. Welche Module für die Installation in Frage kommen, wird im Voraus berechnet.
Die Berechnung startet mit den Features. Manche Features werden von anderen benötigt. Damit dies gewährleistet ist, werden alle benötigten Features zur Auswahl hinzugefügt. Diejenigen, die bereits auf der Instanz installiert sind, fallen aus der Auswahl. Im selben Zuge wird ermittelt, ob Features in einem Konflikt (deny) zueinander stehen. Ist dies der Fall, wird die Berechnung abgebrochen und der Instanzverwalter muss eingreifen. Ihm wird mitgeteilt, dass seine aktuelle Auswahl nicht möglich ist. Entfernt er den Konflikt, kann er die Berechnung erneut starten.
Nun erfolgt die Berechnung der zu installierenden Module. Am Ende muss es mindestens eine Zusammenstellung aus Modulen geben, die alle gewünschten Features abdecken

kann und somit gültig ist. Aber es kann auch mehrere gültige Zusammenstellungen geben. Welche installiert wird, das ist dem Instanzverwalter überlassen.
Es gilt nun für jedes Feature aus der Liste ein geeignetes Modul zu finden. Es kommen dabei nur Module in Frage, die auf mindestens einer Plattform der Instanz installiert werden können.
Dafür wird diese Featureliste nun von oben her abgearbeitet. Damit alle möglichen Kombinationen von Modulen erfasst werden, wird die Rekursion angewandt. Begonnen wird mit einer leeren Zusammenstellung. Die Methode beginnt mit dem Aufruf des ersten Features aus der Featureliste. Jedes Feature enthält eine Menge von Modulen. Jedes Modul wird untersucht. Dafür wird die Menge durchlaufen. Es wird eine Kopie der aktuellen Zusammenstellung erzeugt und dann ein Modul hinzugefügt.
Nun startet die Untersuchung. Dafür wird ermittelt, welche Features durch das hinzugefügte Modul abgedeckt werden. Diese Features werden dann aus der oben erwähnten Featureliste entfernt. Danach wird eine Entscheidung gefällt. Ist die Featureliste nun leer, so sind alle Features durch die Module in der Zusammenstellung abgedeckt. Es wird dann nur noch die Zusammenstellung auf Gültigkeit geprüft. Ist diese gültig, wird sie zur Liste der Zusammenstellungen hinzugefügt. Existieren noch weitere Einträge in der Featureliste, so wird die Methode erneut aufgerufen.
Mit Hilfe der Rekursion können somit alle Kombinationen an Modulen zusammengestellt und überprüft werden. Mit steigender Anzahl an Features steigt auch die Anzahl der Module. Die Anzahl der Kombinationen vergrößert sich dabei exponential. Das bedeutet für den Berechnungsalgorithmus außerdem eine nicht-lineare Steigerung der Ausführungszeit. Um dem Algorithmus zu verbessern, kann im Voraus eine Sortierung der Featureliste durchgeführt werden.
Die Reihenfolge wird folgendermaßen definiert. An erster Stelle steht das Feature, zu dem nur ein einziges Modul in dem Depot existiert. Das bedeutet, nur dieses Modul bietet das Feature an, insofern muss es auf jeden Fall installiert werden. Danach folgen dementsprechend die Features, welche nur von wenigen Modulen unterstützt werden. Am Ende der Liste steht das Feature, welches von den meisten Modulen unterstützt wird.
Das zweite Sortierkriterium bezieht sich auf die Features, die unterstützt würden, wenn das untersuchte Feature installiert wird. Da einige Module nicht nur ein Feature unterstützen, werden bei der Installation eines Features und dessen Modul(en) gleich ein paar andere Features mit abgedeckt. Da jedes Modul eine unterschiedliche Anzahl von Features unterstützt, wird für jedes Feature ein Durchschnitt über alle Module gebildet.

Um ein Beispiel zu geben: Ein Feature, dessen Module im Durchschnitt 1,25 Feature unterstützen, steht an höherer Stelle in der Liste als ein Feature, dessen Module nur 1,05 Feature unterstützen.
Wenn man die Anzahl der Module in einer Zusammenstellung als Güte betrachtet, so nimmt mit zunehmender Ausführungszeit des Algorithmus die Güte der Zusammenstellungen ab. Damit nicht alle Kombinationen, und damit auch die weitaus schlechteren, errechnet werden, bricht der Algorithmus nach einer bestimmten Anzahl von Rekursionsschritten ab.
Am Ende der Berechnung erhält man eine Liste von Zusammenstellungen. Diese werden dem Instanzverwalter vorgelegt. Er hat jetzt die Entscheidung, eine dieser Zusammenstellungen zu wählen.

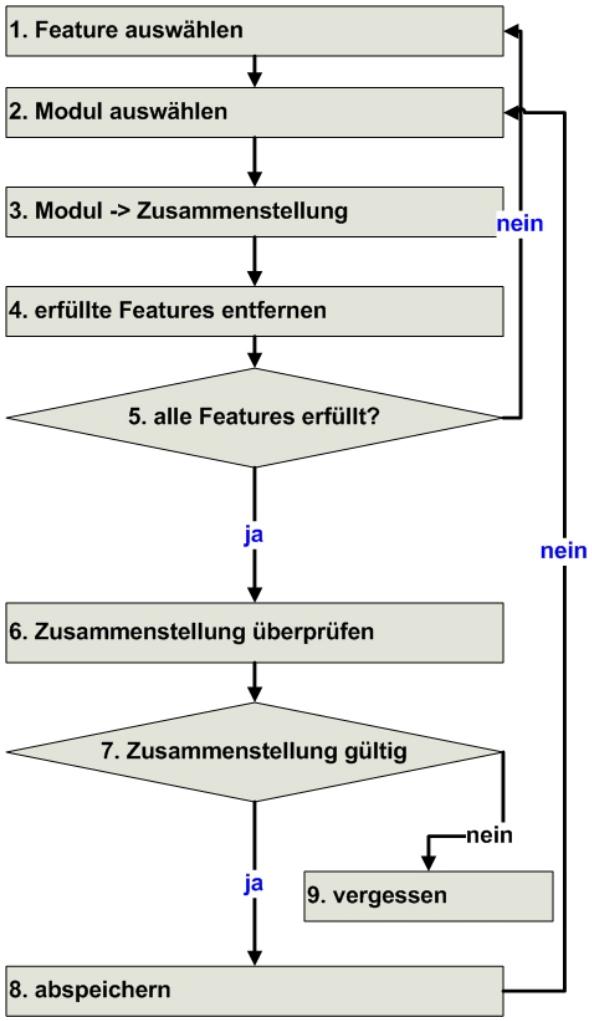

Abbildung 2.8: Ablauf

1. *Das erste Feature aus der Liste wird gewählt.*
2. *Die Liste der Module, die das gewählte Feature unterstützen, wird abgearbeitet und jeweils ein Modul ausgewählt.*
3. *Das gewählte Modul wird der Zusammenstellung hinzugefügt.*
4. *Es wird ermittelt, welche Features durch das Modul abgedeckt werden, um sie dann aus der Liste zu entfernen.*
5. *Sind alle Features aus der Liste entfernt? Wenn ja, gehe zu 6. Bei nein, zu 1.*
6. *Die Zusammenstellung wird geprüft.*
7. *Ist die Zusammenstellung gültig? Wenn ja, gehe zu 8. Bei nein, zu 9.*
8. *Die Zusammenstellung wird abgespeichert. Nun wird der nächste Rekursionsschritt durchgeführt.*
9. *Die Zusammenstellung wird verworfen.*

3 Entwurf und Implementierung

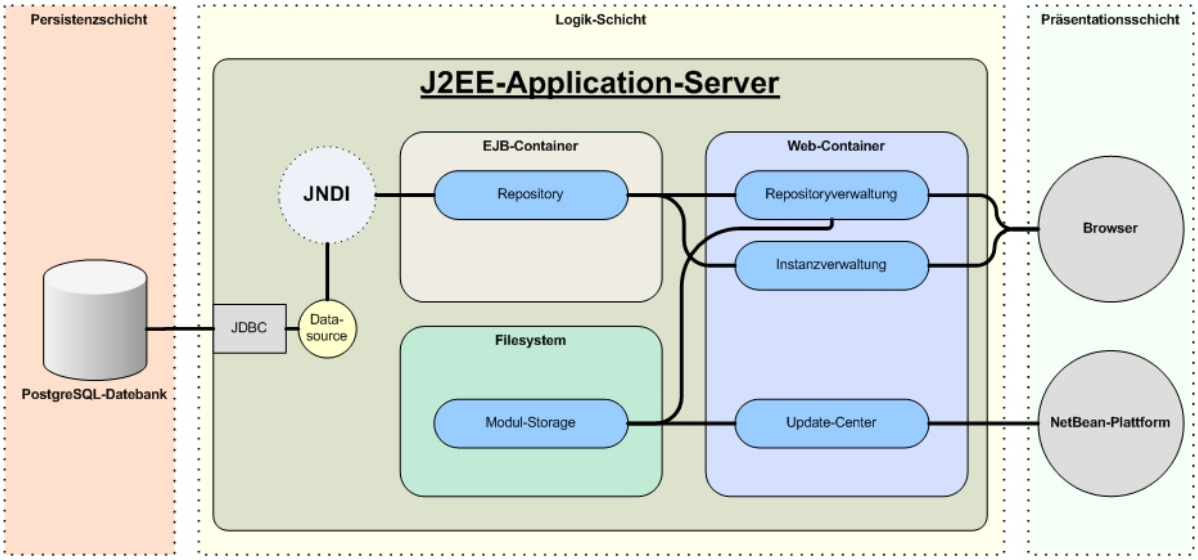

Abbildung 3.1: Prototyp

Für die Diplomarbeit wurde ein Prototyp des Verteilungssystems entworfen. Dieser soll die Untersuchungen aus Kapitel 2 überprüfen und unterstützen.
Die Untersuchung soll prüfen, wie sich ein Verteilungssystem auf Basis der J2EE-Technologie realisieren lässt. Der Prototyp wurde daraufhin als eine Multi-Tier-Anwendung (siehe 3.1) entwickelt. Dabei arbeiten eine PostgreSQL Datenbank zusammen mit einem Sun Application Server als Verteilungsserver und Webbrowser bzw. NetBeans-Plattformen als Clients.

3.1 Multi-Tier-Architektur

Multi-Tier-Architektur heißt übersetzt Mehrschicht-Architektur. Mehrschichtig deshalb, weil man die einzelnen Komponenten einer Anwendung auf einzelne Schichten verteilt. Am geläufigsten ist das Drei-Schicht-Modell (Three-Tier). Man unterteilt dabei in Persistenzschicht, Logikschicht und Präsentationsschicht. Die Persistenzschicht ist zuständig

für die Datenhaltung und wird meist durch eine Datenbank repräsentiert. Die Logikschicht kümmert sich um die Verarbeitung der Daten. Die Darstellung und Eingabe der Daten geschieht über eine GUI oder Weboberfläche in der Präsentationsschicht. Die Präsentationsschicht besteht meistens aus zwei Teilen. Der eine Teil liegt dabei zusammen mit der Logikschicht auf ein und demselben Server und der andere Teil kann auch als Client-Schicht bezeichnet werden. Dies sind die Programme wie Webbrowser usw., die von dem Teil der Präsentationsschicht auf dem Server ihre Daten erhalten.
Die Technologie, die pro Schicht zum Einsatz kommen soll, ist nicht vorgeschrieben, wichtig ist nur, dass die einzelnen Schichten sich problemlos untereinander verständigen können. Dabei müssen sich die Programme, die in den einzelnen Schichten arbeiten, nicht einmal auf demselben Rechner befinden. Dadurch sind Anwendungen in Multi-Tier-Architekturen gut skalierbar, denn rechenintensive Teile können auf stärkere Computer umgelagert werden.
Eine Technologie, die sich auf Multi-Tier spezialisiert hat, ist J2EE. J2EE ist eine Spezifikation der Sun Microsystems Inc., welche Lösungen liefert, um Multi-Tier-Anwendungen zu entwickeln und zu veröffentlichen. Auf Basis der J2EE-Spezifikation haben mehrere Firmen Application Server entwickelt. Dazu gehören unter anderen BEA Systems mit dem Produkt BEA Weblogic Server, IBM mit Websphere Application Server, und Sun Microsystems Inc. liefert den Sun Java System Application Server. Außerdem gibt es einige Open-Source-Projekte. Das bekannteste Produkt aus dieser Sparte ist der JBoss Application Server. [Quelle: (Monson-Haefel 2001, Seite 443f)]
J2EE Application Server übernehmen die Aufgaben der Logikschicht im Three-Tier-Modell und bieten gleichzeitig standardisierte Schnittstellen für die Kommunikation mit der Präsentations- und Persistenzschicht an. Die Abbildung 3.2 zeigt eine schöne Übersicht des Drei-Schichten-Modells mit der J2EE-Technologie.
Auf den Application Servern können Web-Anwendungen, Enterprise Java Beans (EJB) oder eine Zusammensetzung aus beiden, genannt Enterprise-Anwendung, installiert werden. Der Vorgang einer Installation wird als Deployen bezeichnet. Die Anwendungen werden als Archive geliefert. Je nach Anwendungsart handelt es sich dabei um war-Archive (Web-Anwendung), jar-Archive (EJB) oder ear-Archive (Enterprise Anwendung). Diese Archive enthalten außer den Programmklassen einen Deployment-Deskriptor. In diesem ist die Anwendung beschrieben und es sind weitere Informationen enthalten, die für das Deployen wichtig sind.
Da alle J2EE Application Server auf der gleichnamigen Spezifikation aufbauen, können Enterprise-Anwendungen ohne gravierende Anpassung auf einen anderen J2EE Appli-

cation Server portiert werden. Für die Entwicklung des Verteilungssystems wurde der Application Server der Sun Microsystems Inc. genutzt. Dieser ist für die Nutzung im Betrieb nicht kostenfrei. Aber für die Entwicklung bietet er eine gute und übersichtliche Webbackend-Oberfläche. Dort lassen sich die Anwendungen leicht verwalten. Außerdem können dort auch alle weiteren Einstellungen, z.B. für die Datenbank-Anbindung, durchgeführt werden.

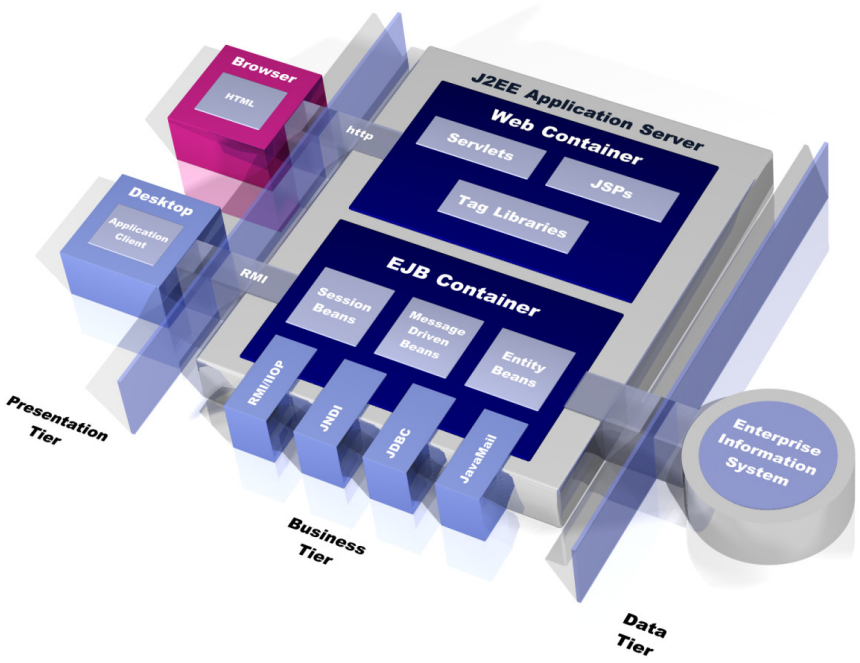

Abbildung 3.2: Java Three-Tier-Architektur [Quelle: bio]

3.1.1 Enterprise Java Beans

Die Hauptarbeit der Geschäftslogik wird von den Enterprise Java Beans übernommen. „Enterprise Java Beans stellen wichtige Konzepte für Unternehmensanwendungen bereit, z.B. Transaktions-, Namens- oder Sicherheitsdienste, die für die 'Geschäftslogik' einer Anwendung benötigt werden.“ [Zitat: wikiejb]

Man unterscheidet dabei zwischen drei Arten von Enterprise Beans: den Entity-Beans, Session-Beans und Message-Driven-Beans. Diese werden im EJB-Container des Application Servers abgelegt. Für die Entwicklung des Verteilungssystems sind Message-Driven-Beans vorerst nicht nötig, deswegen wird in dieser Diplomarbeit auf diese Art von EJB nicht weiter eingegangen. Nähere Informationen über Message-Driven-Beans und den dazugehörigen Java Message Service (JMS) findet man unter [sunmdb] und [sunjms].

Entity-Beans

Entity-Beans bilden die Persistenz in der Logikschicht ab. Man kann sie auch als Geschäftsobjekte bezeichnen, da sie oftmals direkt mit den Geschäftsdaten aus der Datenbank verknüpft sind.
Die Entity-Beans unterteilen sich jeweils in zwei Klassen. Es gibt Beans mit Bean-verwalteter Persistenz (BMP) und Beans mit Container-verwalteter Persistenz (CMP). Bei den BMP-Beans hat der Entwickler dafür zu sorgen, dass die Daten, die das Objekt hält, nicht verloren gehen und vernünftig abgespeichert werden, so dass sie beim nächsten Aufruf wieder zur Verfügung stehen. Er hat dabei für die Transaktionssicherheit zu sorgen und muss jegliche Sicherheits- und Schutzmechanismen selber programmieren.
Benutzt der Entwickler hingegen CMP-Beans, so muss er sich darüber keine Gedanken machen, da der EJB-Container diese Aufgaben für ihn übernimmt. Der Container ermöglicht den Kontakt zu der in der Persistenzschicht liegenden Datenbank und stellt die einzelnen Felder der Tabellen zur Verfügung. Der Entwickler braucht lediglich eine Klasse und die Getter- und Setter-Methoden zu definieren, um auf die Felder der Tabelle zuzugreifen. Die aktuelle Version der NetBeans IDE geht dabei sogar so weit, dass von einer bestehenden Datenbank zuerst ein Schema und daraus dann die CMP-Bean-Klassen und deren Methoden generiert werden können.
Seit der EJB 2.0 Spezifikation können Entity-Beans zueinander Beziehungen eingehen, so wie Tabellen in relationalen Datenbanken. Die Beziehungen und deren Felder, also Primärschlüssel und Fremdschlüssel, werden im Deployment-Deskriptor der Enterprise-Anwendung mitgeliefert. [siehe (Monson-Haefel 2002, Seite 203-205)]

Listing 3.1: EJB QL

```
<query>
        <query-method>
                <method-name>findByObjectid</method-name>
                <method-params>
                        <method-param>java.lang.String</method-param>
                </method-params>
        </query-method>
        <ejb-ql>SELECT OBJECT(o) FROM Object AS o WHERE o.objectid = ?1</ejb-ql>
</query>
```

In relationalen Datenbanken werden Einträge über den Primärschlüssel gefunden oder man benutzt eine zusammengesetzte Abfrage. Auch Entity-Beans können über einen Primärschlüssel gefunden werden. Seit EJB 2.0 wurde außerdem die EJB Query Language (EJB QL) eingeführt. Mit der EJB QL ist es nun auch möglich, Entity-Beans über eine

zusammengesetzte Abfrage aufzufinden. Die Abfragesyntax ähnelt dabei stark der bei relationalen Datenbanken verwendeten Structured Query Language (SQL), siehe Listing 3.1.

Session-Beans

Session-Beans bilden die Geschäftsprozesse einer Anwendung ab. Solche Prozesse betreffen meistens mehrere Entity-Beans, so wie auch bei den Abläufen im System mehrere Akteure tätig und mehrere Komponenten betroffen sind.
Auch bei diesen Enterprise-Beans unterscheidet man zwei Arten. Es gibt zustandslose (stateless) und zustandsbehaftete (stateful) Session-Beans. Die stateful Session-Beans haben ein Gedächtnis und können Daten eines Methodenaufrufs behalten. Das heißt, ruft man die Bean erneut auf, kann man auf diese Daten zurückgreifen. Um Instanzen von stateful Beans im Container zu unterscheiden und wiederzufinden, haben sie eine Identität.
Anders die stateless Session-Beans, sie müssen bei einem Aufruf alle Daten, die zur Ausführung nötig sind, geliefert bekommen. Man kann die Instanzen der stateless Beans im Container nicht voneinander unterscheiden. Dies ist auch nicht nötig, denn sie liefern alle die gleiche Funktionalität.

3.1.2 JDBC

Die Persistenzschicht wird mittels JDBC[1] angesprochen. Damit können Verbindungen zu Datenbanken aufgenommen werden. Dann können SQL-Anfragen gesendet werden. Die Ergebnisse werden in eine in Java nutzbare Form gebracht und stehen somit den Programmen zur Verfügung.
Viele Datenbank-Hersteller liefern zu ihren Datenbanken auch entsprechende JDBC-Treiber. Allerdings sind für einige hohe Kosten für die Nutzung fällig. Deshalb gibt es auch einige Projekte, die auf eigene Faust JDBC-Treiber entwickeln.
Findet man trotzdem keinen passenden Treiber, so kann man zur Not noch auf ODBC ausweichen. Sun stellt für Datenbankanbindungen eine JDBC-ODBC-Bridge zur Verfügung. Ist also eine funktionierende ODBC-Verbindung vorhanden, so kann man diese mit Hilfe des Bridge-Treibers nutzen.

[1] Java Database Connectivity

3.1.3 Java RMI-IIOP

Neben der .NET-Technologie von Microsoft gehören Java-RMI und CORBA IIOP zu den drei wichtigsten Objekt-Diensten. Objekt-Dienste erlauben den Zugriff auf entfernte Objekte, als wären sie in der Anwendung direkt präsent.
CORBA IIOP ist weder betriebssystem- noch sprachspezifisch und lässt sich somit in Anwendungen nutzen, die über mehrere Plattformen verteilt und in unterschiedlichen Programmiersprachen geschrieben worden sind.
Java-RMI nutzt das Java Remote Method Protocol (JRMP) und schränkt sich damit auf die Kommunikation zwischen Java-Anwendungen ein. Mittlerweile existiert allerdings auch Java RMI over IIOP (Java RMI-IIOP). Damit ist es möglich, das Programmiermodell von Java-RMI zu nutzen, ohne auf die Vorteile des plattform- und sprachunabhängigen CORBA-Protokolls IIOP verzichten zu müssen. [(Monson-Haefel 2002, Seite 80f)]

3.1.4 JNDI

Das Java Naming and Directory Interface (JNDI) ist eine Schnittstelle, um auf die Namens- und Verzeichnisdienste der Objekt-Dienste zuzugreifen. Über eine URL können Objekte angesprochen und abgerufen werden – Beispiel: „java:comp/env/person“.

3.2 Persistenzschicht

Für die Darstellung der Komponenten und der Abhängigkeiten wurde ein Datenmodell geschaffen. Dieses wurde auf eine SQL-Datenbank portiert, um die Fähigkeiten des Modells, zu testen. Als Plattform diente dabei eine Datenbank im Microsoft-SQL-Server. Ein Schema der Datenbank ist in Abbildung 3.3 zu sehen.
Das Modell repräsentiert ebenfalls die vier Bereiche, die wir in Abschnitt 2.4 kennen gelernt haben. Die linke Seite entspricht dem Repository. Die Tabellen „module“ ist in der realen Ebene und die Tabellen „platform“, „feature“ und „package“ in der abstrakten Ebene. Hier ist die Übertragung des Schemas 2.3 von Seite 12 auf das Datenmodell geschehen.
Auf der rechten Seite befindet sich die Installation. Die Tabellen „configuration“, „instance“, „deployment“ gehören zur abstrakten Ebene und die Tabelle „installation“ repräsentiert die reale Installation auf den Plattformen.

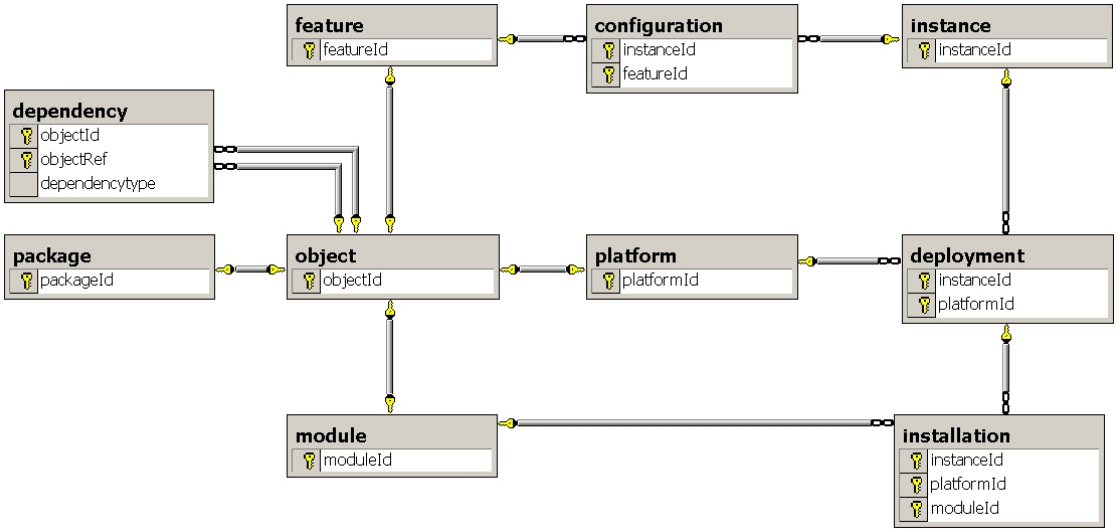

Abbildung 3.3: SQL-Datenmodell

Die Features, Module, Plattformen und Pakete sind alle über eine Tabelle „object“ verknüpft. Und diese wiederum mit einer Tabelle namens „dependency“. Diese beiden Tabellen dienen dazu, die Abhängigkeiten zwischen den Komponenten darzustellen. Da alle vier Komponenten über die einer Tabelle verknüpft sind, ist man so ziemlich flexibel. Denn alle Komponenten können nun Beziehungen zueinander eingehen.
Welche Art von Beziehung man erstellt, kann mit der Spalte „dependencytype“ angegeben werden. Für die Beziehung zwischen Modul und Plattform wird als Typ „PC“ eingetragen. Dies steht abkürzend für „Platform Choice“. Damit ist gemeint, dass dieses Modul auf der angegebenen Plattform installiert werden kann. Es können natürlich noch mehrere mögliche Plattformen für das Modul eingetragen werden, deswegen „Choice“, damit man die Wahl hat, wo das Modul installiert werden soll.
Für die Tests war die Datenbank auf dem Microsoft-SQL-Server ausreichend. Während der Implementierungsphase stellte sich aber heraus, dass dieser nicht gut mit dem Application Server zusammenarbeitet. Ein Hauptgrund ist der JDBC-Treiber des jTDS Projekts [jtds]. Dies ist ein Open-Source JDBC 3.0 Treiber für den Microsoft-SQL-Server. Sicherlich gibt es auch professionelle JDBC Treiber für den Microsoft-SQL-Server, allerdings werde dafür hohe Lizenzkosten verlangt.
Die Wahl fiel dann auf den PostgreSQL 8.1 Server. Dieser arbeitete ohne Probleme mit dem Application Server zusammen und ist zudem frei erhältlich ebenso wie die zugehörigen JDBC Treiber.

3.3 Logikschicht

In der Logikschicht übernehmen Entity-Beans und Session-Beans die Aufgabe der Geschäftslogik. Die über die Präsentationsschicht abgefragten Daten werden aus der Datenbank geholt, von den EJB aufbereitet und dann der Präsentationsschicht zur Verfügung gestellt. Gleiches gilt für die andere Richtung: eingegebene Daten werden in der Logikschicht bearbeitet und dann in der Datenbank gespeichert.
In der Logikschicht des Prototyps arbeitet der J2EE-Application-Server. Dieser beherbergt mehrere Anwendungen.

3.3.1 Repository

Die Repository-Anwendung repräsentiert das Depot des Verteilungssystems. Es liefert die Module und gibt Auskunft, welche Features zur Verfügung stehen. Der Distributor kann auf das Repository mittels einer Administrations-Webseite zugreifen (siehe 3.4.1). Außerdem beinhaltet diese Anwendung den Berechnungsalgorithmus für die Zusammenstellung der Module (Kapitel 2 Abschnitt 2.6).
Der Instanzverwalter kann über eine Webseite eine Konfiguration zusammenstellen. Dafür fragt er die Repository-Anwendung, welche Features zur Verfügung stehen, und konfiguriert damit die Instanz. Die fertige Konfiguration schickt er dann der Repository-Anwendung. Diese berechnet die Zusammenstellungen der Module und schickt dem Instanzverwalter dann die Liste der Zusammenstellungen zu.
Der Instanzverwalter sieht eine Übersicht von den Modulen, die in den Zusammenstellungen vorhanden sind, und kann dabei auch genau erkennen, auf welche Plattform der Instanz sie installiert werden würden. Somit kann er sich schon vorher eine Vorstellung machen, wie es nach der Installation aussehen wird.
Hat er sich dann für ein Zusammenstellung entschieden, teilt er dieses der Repository-Anwendung mit. Diese führt dann anhand der Zusammenstellung die Installation der Module auf den Plattformen durch.
Die Repository-Anwendung besteht aus Entity-Beans und Session-Beans. Die Hauptaufgabe der Session-Beans in dieser Anwendung ist die Kapselung der Entity-Beans. Da auch Entity-Beans entfernte Methoden besitzen können, liegen die Geschäftsdaten völlig offen für alle Clients. Beim direkt Umgang mit den Geschäftsdaten entfällt ein großer Teil der eigentlichen Logik auf den Client. Außerdem kommt es bei komplexen Prozessen, in denen mehrere Geschäftsobjekte beteiligt sind, zu mehrfachen Methodenaufrufen. Dies führt unmittelbar zu Performanzproblemen im Netzwerk.

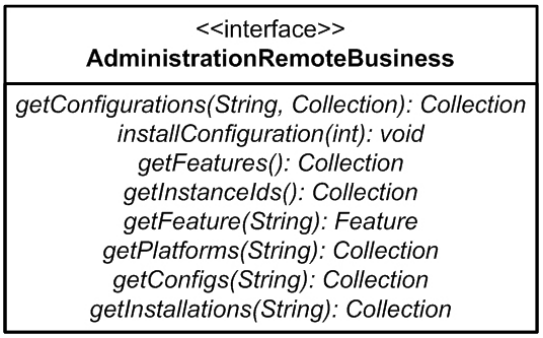

Abbildung 3.4: Administration Remote Interface

Um diese Probleme zu umgehen, baut man eine Fassade aus Session-Beans auf (Session Facade). Dabei übernehmen einige Session-Beans die Kommunikation zwischen den Clients und den Entity-Beans. Die Session-Beans können lokal auf die Entity-Beans zugreifen, was den Netzwerkverkehr verringert.
Die zusätzliche Schicht kann den Zugriff auf die Geschäftsobjekte steuern. Das bedeutet, es können Zugriffe erlaubt oder verwehrt und somit Missbrauch ausgeschlossen werden. Eine Session Facade ist ein J2EE-Pattern. Pattern sind Muster bzw. Muster-Lösungen, um Probleme in der Programmierung anzugehen. Probleme, die immer wieder auftauchten, wurden von mehreren Entwicklern unterschiedlich gelöst. Die besten Lösungen werden dann zusammengetragen und in Büchern oder gar Spezifikationen veröffentlicht. Mehr über die Session Facade und andere J2EE-Pattern kann man in [Alur u. a. (2002)] lesen.
In der Implementierung des Verteilungssystems gibt es eine Session-Bean (AdministrationSB), die einen Haupteil der Anfragen übernimmt. Sie arbeitet mit der Administrationswebseite des Instanzverwalters zusammen. Der Instanzverwalter erhält durch diese Bean Informationen über die Features, die im Repository verfügbar sind. Außerdem kann er abfragen, welche Features und Module bereits auf seiner Instanz installiert sind. Über diese Bean läuft auch die Installation, wie oben beschrieben.
Die Abbildung 3.4 zeigt die Methoden, welche alle über das entfernte Interface aufgerufen werden können.

3.3.2 Updateserver für NetBeans

Diese Anwendung stellt im Prinzip eine Plattform einer Instanz dar. Da die NetBeans-Clients im Netzwerk verteilt sind und auf manchen PCs nicht immer an gleicher Stelle installiert sind, gestaltet es sich schwierig, diese mit neuen Modulen zu versorgen.

Es gibt allerdings ein Updateplugin für NetBeans. Ist diese im Client vorhanden, so kann man es einrichten. Dabei hat man die Möglichkeit, im Plugin mehrere Updateserver einzutragen. Um einen Server nach neuen Updates abzufragen, wird dabei nur eine URL aufgerufen. Diese liefert eine XML-Datei zurück, die alle Information über vorhandenen Module auf dem Updateserver enthält.
Bei einer Installation, bei der auf den NetBeans-Clients Module installiert werden sollen, werden diese dann im Updateserver installiert. Die Dateien werden in einem Verzeichnis abgelegt. Die Module für die NetBeans-Clients sind Archive mit der Dateiendung nbm. Die Updateserver-Anwendung öffnet jedes dieser Archive und holt aus ihnen Informationen über das Modul. Mit diesen Informationen wird die XML-Datei erstellt, die durch das Updateplugin abgerufen wird.
Auf dem Screenshot in Abbildung 3.5 sieht man den neu hinzugefügten Updateserver. Nach dem Verbindungsaufbau mit dem Updateserver wird die XML-Datei runtergeladen und ausgewertet. Die Informationen werden dann im nächsten Bild angezeigt.

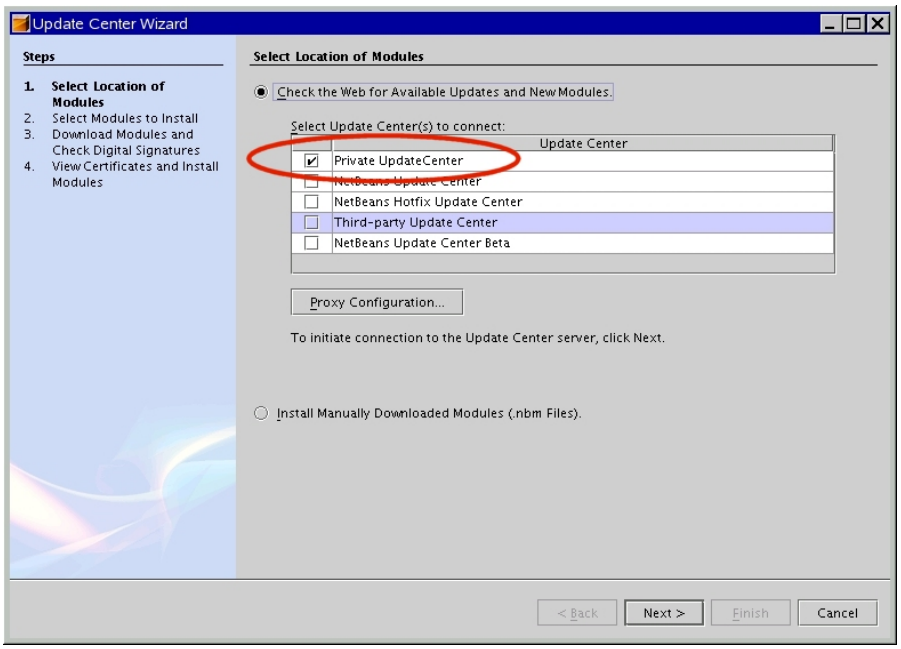

Abbildung 3.5: Updatecenter-Auswahl

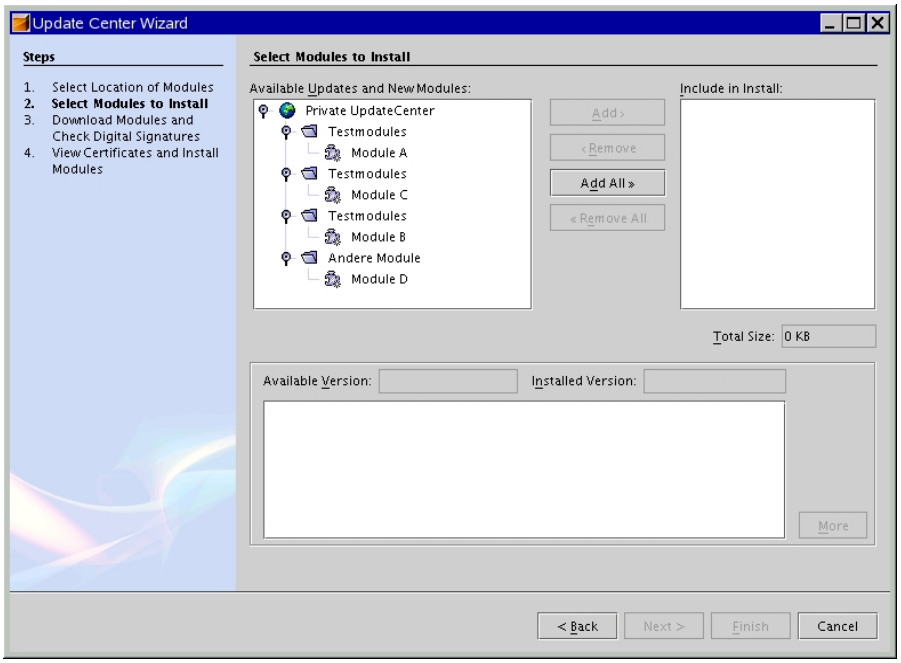

Abbildung 3.6: Updatecenter-Modulliste

3.4 Präsentationsschicht

Die Präsentationsschicht besteht aus mehreren Teilen, die zusammenarbeiten. Die eigentliche Ansicht der Daten und die Eingabe solcher geschieht mit Hilfe eines Browsers oder einer grafischen Oberfläche. Die Information für die Anzeige kommen allerdings vom Server. Dort gibt es Webanwendungen, die die Webseiten für den Browser bereitstellen. Oder es sind Enterprise-Beans installiert, die Kommunikation mit einem grafischen Client-Programm betreiben.

3.4.1 Repositoryverwaltung

Die Repositoryverwaltung ist für den Distributor gedacht. Er hat dort die Möglichkeit sich anzeigen zu lassen, welche Module im Repository vorhanden sind. Er kann sie sich einzeln runterladen und auf seiner Festplatte abspeichern. Auch ist es ihm möglich, über die Webseite Module in das Repository reinzuladen.
Und er kann das Repository mit einem anderen Repository synchronisieren. Dies kann dabei im gleichen Firmennetz liegen, aber auch irgendwo im Internet.

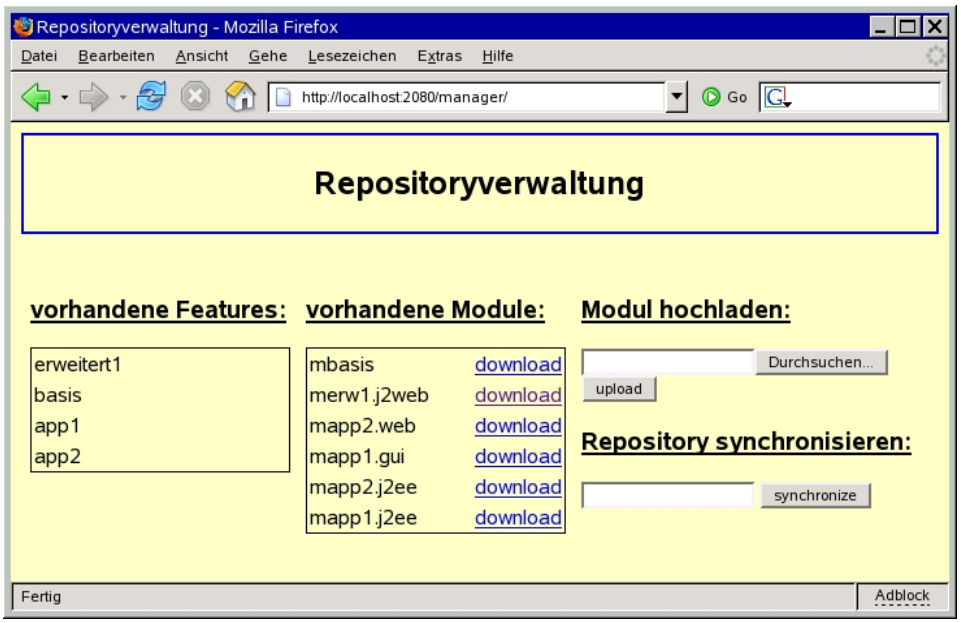

Abbildung 3.7: Repositoryverwaltung

3.4.2 Instanzverwaltung

Ein Teil der Instanzverwaltung wurde schon in Abschnitt 3.3.1 erklärt. Das geschah allerdings aus der Sicht der Enterprise-Beans.
Die Webanwendung besteht im Grunde nur aus einer Webseite. Dort sieht der Instanzverwalter seine Instanz und die darauf installierten Module bzw. Features.
Die Features sind an- und abwählbar und er kann sich zu der erstellten Konfiguration die Zusammenstellungen berechnen lassen.

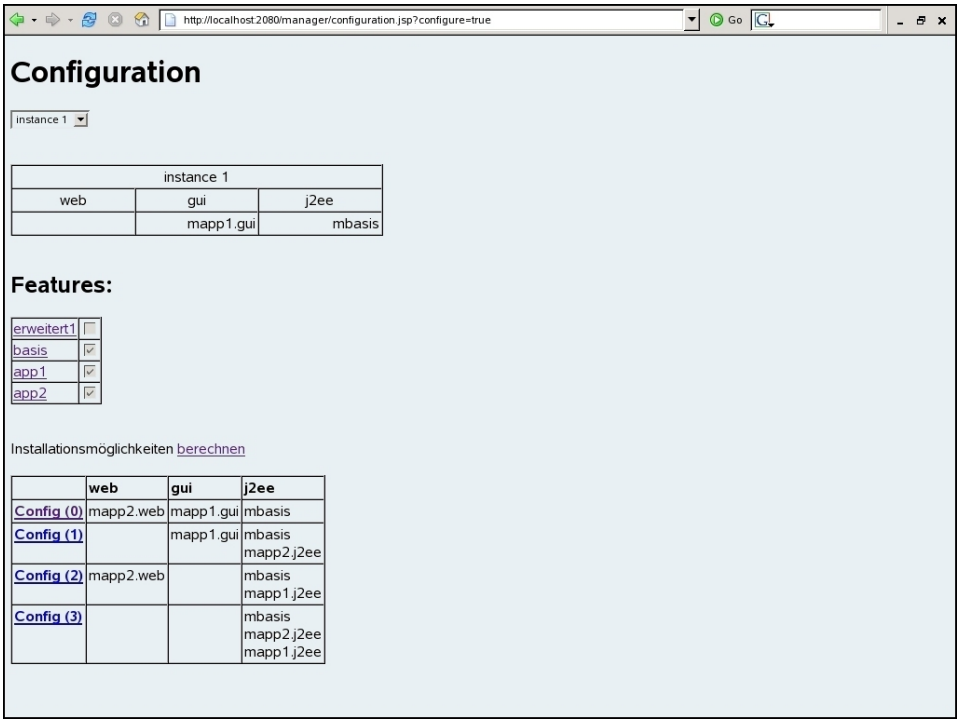

Abbildung 3.8: Instanzverwaltung

3.4.3 Netbeans-Client

Der Netbeans-Client dient als Testplattform. Es wurde ein Testmodul erstellt, welches eine einfache Eingabe von Personendaten zeigt. Diese Daten können abgespeichert werden – siehe Abbildung 3.9. In einer neuen Version sind neue Felder hinzugekommen. Dieses Update soll dann über das System an den NetBeans-Client verteilt werden – siehe Abbildung 3.10.

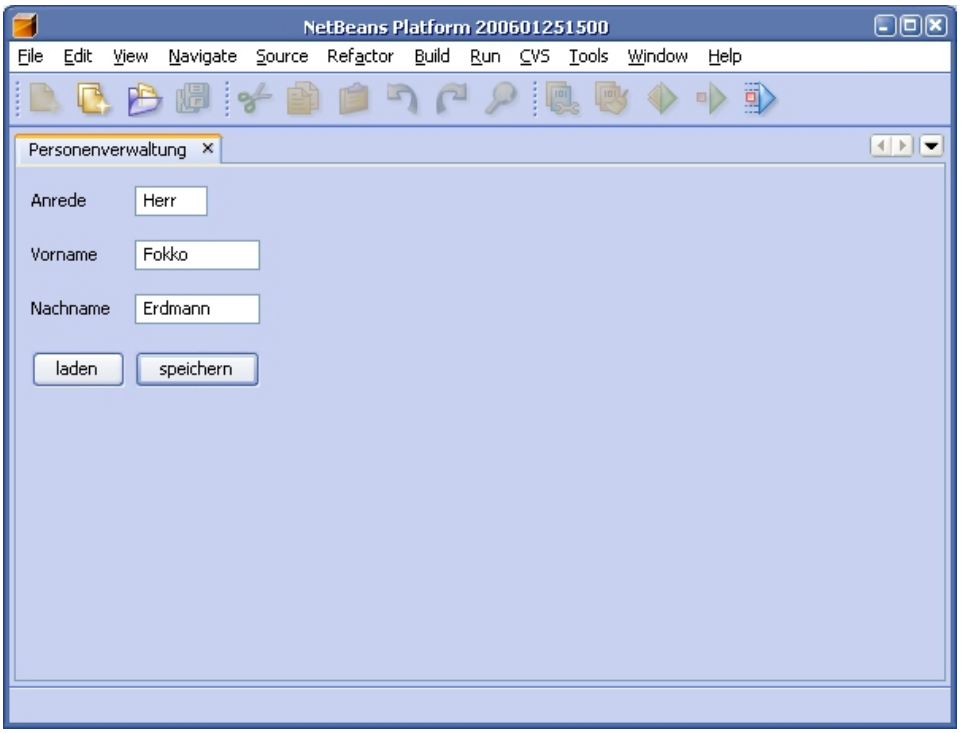

Abbildung 3.9: NetBeans-Modul

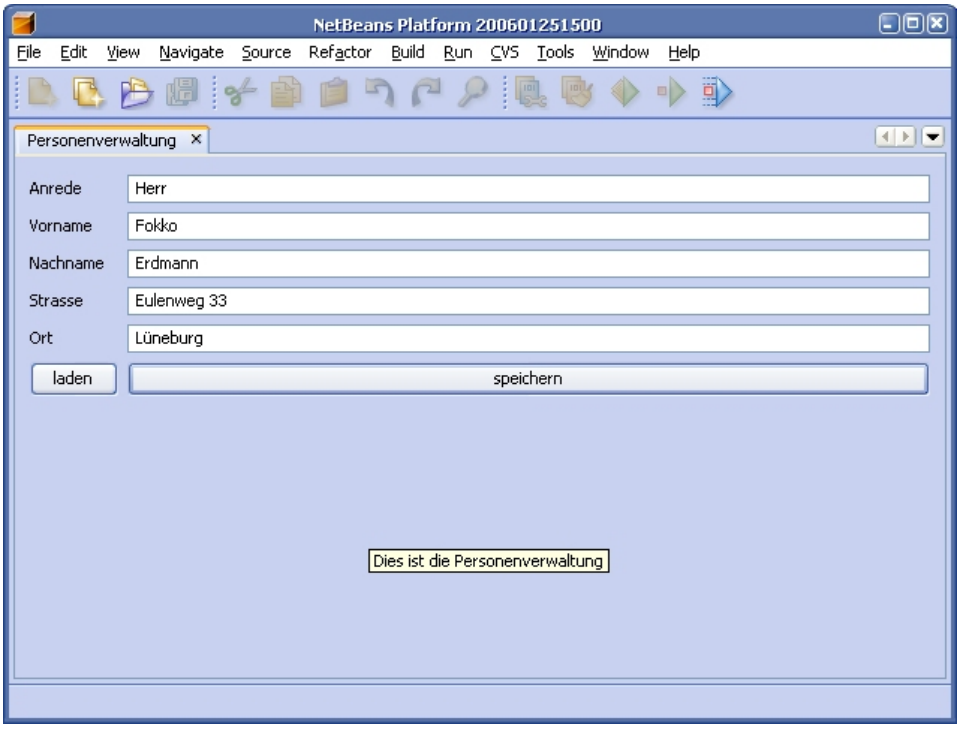

Abbildung 3.10: NetBeans-Modul nach Update

3.5 Beschreibungsdatei

Die während dieser Arbeit schon oft erwähnte Beschreibungsdatei dient zur Übermittlung der Modul- und Featureinformation. Außerdem sind in ihr die Abhängigkeiten zwischen den Komponenten eingetragen. Einen beispielhaften Aufbau einer solchen Datei zeigt Listing 3.2.

Listing 3.2: Beschreibungsdatei

```
modules=M1,M2,M3,M7

module.M1.name=Modul 1
module.M1.filename=module1.nbm
module.M1.features=F1,F2
module.M1.platforms=P1,P3

module.M2.name=Modul 2
module.M2.filename=module2.nbm
module.M2.features=F2
module.M2.platforms=P2

module.M3.name=Modul 3
module.M3.filename=module3.nbm
module.M3.features=F3
module.M3.platforms=P4
```

```
module.M7.name=Module 7
module.M7.filename=module7.nbm
module.M7.updates=M4,M5
module.M7.platforms=P1,P2,P3

# possible options
# module.<modulekey>.require=<modulekey>, ...
# module.<modulekey>.deny=<modulekey>, ...
#
# feature.<featurekey>.require=<featurekey>, ...
# feature.<featurekey>.deny=<featurekey>, ...
```

4 Tests und Validierung

4.1 Testfälle

Die SQL-Datenbank hat den Vorteil, dass nun die Prozesse des Verteilungssystems anhand dieses Datenmodells getestet werden können. Wird in dieser Testphase festgestellt, dass noch zusätzliche Komponenten im System benötigt werden oder Komponenten verändert werden müssen, so kann dies schnell bewerkstelligt werden, während in der Implementierungsphase oder sogar im laufenden Betrieb solche Änderungen viel Zeit und Aufwand bedeuten.
Hauptsächlich galten die Testfälle allerdings zur Überprüfung des Algorithmus, der die Zusammenstellungen für die Installation erstellt. Die Tests soll ergeben, dass alle gestellten Erwartungen erfüllt werden.

4.1.1 Testfall 1: (De)Installation eines Features

Es wurden nun, um die Prozesse im System nachzuahmen, Testfälle erstellt. Diese Testfälle sind jeweils ein SQL-Skript, welches an einem definierten Startzustand beginnt und mit einer Ausgabe als Ergebnis endet.
Listing 4.1 zeigt, wie so ein Testfall aussieht.

Listing 4.1: Testfall 1

```
1  USE master
2  GO
3  RESTORE DATABASE repository FROM DISK='repository_init'
4  GO
5
6  USE repository
7  SET NOCOUNT ON
8
9  PRINT '----------------------------------------'
10 PRINT 'Testfall 1'
11 PRINT '----------------------------------------'
12
13 INSERT INTO object (objectId) VALUES ('PG1');
```

```
14 INSERT INTO package (packageId) VALUES ('PG1');
15
16 INSERT INTO object (objectId) VALUES ('F1');
17 INSERT INTO feature (featureId) VALUES ('F1');
18
19 INSERT INTO object (objectId) VALUES ('M1');
20 INSERT INTO object (objectId) VALUES ('M2');
21 INSERT INTO object (objectId) VALUES ('M3');
22 INSERT INTO module (moduleId) VALUES ('M1');
23 INSERT INTO module (moduleId) VALUES ('M2');
24 INSERT INTO module (moduleId) VALUES ('M3');
25
26 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('PG1', 'F1', 'PF')
       ;
27
28 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('F1', 'M1', 'FR');
29 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('F1', 'M2', 'FR');
30 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('F1', 'M3', 'FR');
31
32 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('M1', 'P1', 'PC');
33 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('M2', 'P1', 'PC');
34 INSERT INTO dependency (objectId, objectRef, dependencytype) VALUES ('M3', 'P3', 'PC');
35
36 INSERT INTO configuration (instanceID, featureId) VALUES ('I1', 'F1');
37
38 -- De-/Installation
39 PRINT ' vor der Installation von Feature 1'
40 EXEC dbo.viewInstall 'I1'
41 EXEC dbo.doInstall 'I1'
42 PRINT ' nach der Installation von Feature 1'
43 EXEC dbo.viewInstall 'I1'
44 DELETE FROM configuration WHERE featureId = 'F1'
45 PRINT ' vor der Deinstallation von Feature 1'
46 EXEC dbo.viewInstall 'I1'
47 EXEC dbo.doInstall 'I1'
48 PRINT ' nach der Deinstallation von Feature 1'
49 EXEC dbo.viewInstall 'I1'
50
51 USE master
52 GO
53 BACKUP DATABASE repository TO DISK='repository_1' WITH INIT
54 GO
```

Man sieht in diesem Testfall 1 zuerst die Wiederherstellung der Datenbank von einem gesicherten Stand „repository_init“. Damit wird sichergestellt, dass zu jeder Zeit, in der dieser Testfall durchgeführt wird, die gleichen Startbedingungen herrschen. Die Datenbank ist vor diesen Testfällen vorbereitet worden. Das bedeutet, es existiert bereits eine Instanz, welche so konfiguriert wurde, dass sie die Plattformen P1, P2 und P3 enthält. Der Testfall 1 beginnt damit, dass ein Paket PG1 in das Repository integriert wird. Dafür werden jeweils für das Paket, das Feature und die Module ein Eintrag in der Objekt-

Tabelle und in den zugehörigen Tabellen erzeugt. In die Dependency-Tabelle werden dann die Abhängigkeiten eingetragen. Dazu gehören die Abhängigkeiten zwischen Feature und Modulen – siehe Listing 4.1 Zeilen 28–30 – und die Abhängigkeiten zwischen den Modulen und Plattformen – siehe Zeilen 32–34. Nachdem dann das Feature F1 in der Instanz für die Installation eingetragen wurde (Zeile 36), wird die Installation der Module angestoßen. Dafür existiert in der Datenbank eine Installationsroutine in Form einer Stored Procedure[1].

Listing 4.2: Stored Procedure

```
1  CREATE PROCEDURE dbo.doInstall
2    @instanceId as varchar(50)
3  AS BEGIN
4    DECLARE @toinstall TABLE(
5          instanceId varchar (50) NOT NULL ,
6          platformId varchar (50) NOT NULL ,
7          moduleId   varchar (50) NOT NULL,
8          aktion     varchar (50) NULL,
9      PRIMARY KEY (       instanceId, platformId, moduleId )
10   )
11   INSERT INTO @toinstall SELECT * FROM dbo.toInstall(@instanceId)
12   INSERT INTO installation
13     SELECT instanceId, platformId, moduleId FROM @toinstall WHERE aktion = 'I'
14   DELETE
15     installation
16     FROM installation AS i1
17     INNER JOIN @toinstall AS i2 ON (
18       i1.instanceId = i2.instanceId
19       AND
20       i1.platformId = i2.platformId
21       AND
22       i1.moduleId = i2.moduleId
23       AND
24       i2.aktion = 'D'
25     )
26 END
```

Der Procedure wird die Instanz als Parameter übergeben. Dieser Parameter wird wiederum zum Aufrufen einer Function[2] benutzt. In dieser Function wird ermittelt, welche Module auf welcher Plattform installiert und welche deinstalliert werden müssen.
Dabei wird folgendermaßen vorgegangen. Zuerst werden die Features ermittelt, die alle zur Installation für diese Instanz ausgewählt sind, und dann die abhängigen Features.

[1] Eine Stored Procedure ist ein gespeichertes SQL-Skript, welches in der Datenbank hinterlegt ist. Es kann somit eine ganze Abfolge von Befehlen mit einem Aufruf durchgeführt werden.

[2] Anders als die Procedure kann eine Function Werte zurückliefern. Der Rückgabewert kann sogar eine Tabelle sein.

Nun werden alle Module, welche den Features angehören, ermittelt und diejenigen, die voneinander abhängen. Aus dieser Modulliste werden dann alle Module entfernt, die bereits durch ein anderes aktualisiert wurden. Die letzte Liste, welche ermittelt wird, umfasst alle Plattformen, welche von den Modulen aus der Modulliste unterstützt werden. Als Ergebnis erhält man eine Tabelle, in der die Instanz, die Module und die Plattformen (wo die Module installiert werden sollen) enthalten sind. Diese Tabelle wird nun aufgearbeitet und um eine Aktionsspalte ergänzt. Dafür wird die Instanz betrachtet und untersucht, welche Module bereits auf welcher Plattform installiert sind. Dies wird wiederum in eine Tabelle eingetragen. Jetzt werden die beiden Tabellen miteinander verglichen. Für diejenigen Module, die bereits auf der entsprechenden Plattform installiert sind, ist keine Aktion nötig. Module, die nicht für die Installation vorgesehen, aber installiert sind, werden mit der Aktion „deinstallieren“ gekennzeichnet. Alle übrigen Module bekommen die Aktion „installieren“.
Aufgrund der Aktionsspalte können nun INSERT- und DELETE-Anweisungen auf der Installations-Tabelle durchgeführt werden. Zur Überprüfung wird jeweils vor und nach der Installation eine Ausgabe der Installations-Tabelle durchgeführt und eine Liste der Aktionen, die vorgesehen sind. Somit kann geschaut werden, ob alles geklappt hat.
Nach der Installation wird das Feature wieder von der Instanz entfernt und es wird abermals die Installationsroutine gestartet. Nachdem diese durchlaufen wurde, muss der gleiche Zustand wie vor der Installation herrschen.
Weitere Testfälle wurden erstellt:

4.1.2 Testfall 2: (De)Installation eines zusätzlichen Features

In diesem Testfall wird überprüft, wie sich die Installationsroutine verhält, wenn bereits ein Feature installiert ist. Dafür wird ein neues Feature F2 der Instanz hinzugefügt. Von den Modulen, welche dieses Feature benötigt, ist bereits eines installiert, da es schon zu dem installierten Feature gehört. Die Installationsroutine erkennt und ermittelt, dass für dieses Modul keine Aktion nötig ist, es also weder nochmal installiert noch deinstalliert wird. Außerdem wird gewährleistet, dass bei der Deinstallation des zusätzlichen mboxFeatures jenes Modul erhalten bleibt. Sprich: es wird wieder keine Aktion für dieses Modul durchgeführt, da es ja noch von dem anderen Feature benötigt wird.

4.1.3 Testfall 3: Integration eines Updates

Dieser Testfall ist dazu gedacht zu überprüfen, was passiert, wenn ein Paket, welches Updates enthält, in das System gebracht wird. Das im Paket enthaltene Modul unterstützt sowohl Feature F1 als auch Feature F2. Außerdem aktualisiert das Modul je ein Modul aus F1 und aus F2. Nach dem Hinzufügen des Updates wird die Installationsroutine gestartet. Diese ermittelt nun die Installation des Updatemoduls und die Deinstallation der beiden installierten Module.

4.1.4 Testfall 4: Installation alter Module

In diesem Testfall wird ein Feature installiert, welches ein Modul beinhaltet, das bereits durch ein anderes aktualisiert wurde. Die Installationsroutine verhindert, dass das alte Modul installiert wird. Das neue Modul wird die Aufgabe des alten übernehmen.

4.1.5 Testfall 5: Modulabhängigkeiten (require)

Es kann vorkommen, dass ein Modul, welches installiert werden soll, zusätzliche Module benötigt, damit es seine volle Funktionalität gewährleisten kann. Dieser Testfall wird diese Abhängigkeiten untersuchen und soll feststellen, ob alle benötigten Module installiert werden.
Dafür wird ein Paket in die Datenbank integriert, welches drei Module enthält. Eines der Module M10 ist dem Feature F1 zugeordnet, welches bereits installiert ist. Die beiden anderen Module M11 und M12 sind direkt bzw. indirekt von M10 abhängig. M10 benötigt Modul M11 und M11 benötigt Modul M12. Die Installationsroutine ermittelt jetzt, dass alle drei Module installiert werden müssen, obwohl nur M10 dem Feature F1 zugeordnet ist.

4.1.6 Testfall 6: Modulabhängigkeiten (deny)

Im Testfall 6 soll absichtlich ein Fehler produziert werden. In diesem Fall ist der Fehler aber eine richtige Reaktion der Installationsroutine und soll auftreten. Es geht darum, dass ein Modul installiert werden soll, welches nicht mit einem anderen Modul zusammenarbeitet.
Das Paket, welches in die Datenbank integriert wird, enthält ein Modul M15, welches zu Feature 2 gehört. Es besteht die Einschränkung, dass M15 nicht mit Modul M2 installiert werden darf. Bereits installiert ist das Modul M6, welches ein Update für M2 ist.

Die Installationsroutine erkennt, dass M2 sozusagen in M6 integriert ist, und meldet, dass M15 nicht installiert werden kann aufgrund der Beschränkung mit M2.

Eine ausführliche Beschreibung der Testfälle ist im Anhang auf Seite 44 beigefügt.

5 Zusammenfassung und Ausblick

Die Motivation für diese Diplomarbeit war, die Administratoren von Firmen zu entlasten. Sie sollten bei der Installation von Software auf Arbeitsstationen unterstützt werden.
Das Resultat dieser Arbeit ist ein System, das alle Personen, die von der Entwicklung bis zur Anwendung der Software beteiligt sind, entlastet. Entwickler können sich voll auf ihre Programmierung konzentrieren und brauchen sich keine Gedanken um den Vertrieb der Software zu machen. Die Distributoren erhalten Forderungen von den Administratoren und können diese an die Softwareentwickler weiterleiten und somit aktuelle und angepasste Software liefern.
Der Administrator braucht nicht mehr zu den Arbeitsstationen zu laufen, um dort Software zu installieren. Er kann dies von zentraler Stelle aus tun. Dafür wählt er in erster Linie nur die Fähigkeiten (Features) aus, die er haben möchte, und braucht sich um die Software, die letztendlich installiert wird, keine Gedanken zu machen. Auch wenn neue Versionen zur Verfügung stehen, werden diese schnell und unkompliziert integriert.
Das entworfene System kann mit aller Art von Software umgehen und diese verteilen. Dafür sorgt die durchgeführte Abstraktion der beteiligten Komponenten.
Der Prototyp wurde auf Basis der J2EE-Architektur entwickelt. Er stellt eine einfache Version des Systems dar, mit dem einige Prozesse schon möglich sind. So können z.B. Module in das Repository integriert werden. Außerdem kann der Instanzverwalter seine Instanz mit Features aus dem Repository bestücken. Die zugehörigen Module werden bisher zwar nur in unterschiedliche Verzeichnisse kopiert, aber selbst das kann schon eine Installation bedeuten. Zum Beispiel bedient sich die NetBeans-Update-Anwendung bei der Zusammenstellung der Info-XML-Datei aus einem Verzeichnis, in dem die Module gelagert werden. Und auch der Sun Application Server besitzt ein Autodeploy-Verzeichnis. Dort abgelegte Application-Dateien werden automatisch in den Server integriert.
Bei der Entwicklung einer J2EE-Anwendung stellt man fest, dass diese sich doch gänzlich von der Entwicklung normaler Java-Anwendungen unterscheidet. Gerade die Enterprise-Beans sind nicht leicht zu handhaben. Für deren Installation im Application Server sind allein fünf Klassen notwendig. Außerdem müssen diverse Einstellungen in den

Deployment-Deskriptoren gemacht werden. Dies bedeutet einen hohen Aufwand, der getätigt werden muss, und hohe Fehleranfälligkeit. Mittlerweile können Programmierumgebungen wie NetBeans diesen Aufwand minimieren, da viele der Einstellungen durch NetBeans übernommen werden. Das bedeutet allerdings nicht, dass dies fehlerfrei vonstatten geht.
Hoffen lässt da die Aussicht auf die neue EJB 3.0 Spezifikation. Die Entwickler dieser Spezifikation haben die Nachteile der aktuellen Version erkannt und entsprechend reagiert. Die Deployment-Deskriptoren sollen verschwinden oder wenigstens deren Umfang verringert werden. Außerdem soll die Anzahl an Klassen pro EJB minimiert werden. Man will zurückkehren zu den einfachen Java Objekten (POJO: Plain Old Java Object und POJI: Plain Old Java Interface). Zur Erstellung von Enterprise Beans sind dann nur noch diese zwei Komoponenten notwendig. Das Objekt beinhaltet die Logik und das Interface ist für den entfernten Zugriff auf das Objekt zuständig. Die üblichen Deklarationen für locale und remote Methoden sollen direkt in den Code fließen und können so schon bei der Übersetzung durch den Compiler überprüft werden. Dies verringert den Zeitaufwand des Testens, da die Enterprise-Beans nicht erst auf den Application-Server deployt werden müssen. Damit kann nun wieder normal objektorientiert modelliert und implementiert werden. Eine Vorstellung dieser Neuerungen fand in Javamagazin Ausgabe 3/05 statt [Ihns (java)].
Abschließend ist jetzt nur noch zu sagen, dass die Realisierung eines Softwareverteilungssystems auf Basis der J2EE-Architektur möglich ist, und der erste Schritt zu einer benutzbaren Anwendung wurde mit dieser Diplomarbeit getan.

Anhang

A Testfälle

Um das entwickelte Datenmodell zu testen, wurde es auf eine Datenbank im MSSQL-Server übertragen. Hier können nun mittels SQL-Skripten unterschiedliche Testszenarien durchgespielt werden. Die daraus resultierenden Algorithmen können dann in Java-Klassen implementiert werden.

Test 1 (Basistest I)

Voraussetzungen:

Es existiert eine Instanz(I1), auf der drei Plattformen(P1, P2, P3) laufen.

Beschreibung:

Das Package 1 (PG1) besteht aus drei Modulen(M1, M2, M3) und einer Deskriptor-Datei. Die drei Module gehören zu dem Feature 1(F1). Die Module M1 und M2 sind der Plattform P1 zugeordnet und Modul M3 der Plattform P3.

Mögliche Konflikte:

- Das Feature 1 kann nicht installiert werden, weil die erforderlichen Plattformen fehlen.
- Die Deskriptor-Datei fehlt.

Testfälle:

1.1 Das Package 1(PG1) wird in das Repository geladen.
Erwartungen:

- Alle Module sind ausgepackt und abgelegt. Die Abhängigkeiten zwischen den Modulen und den Features sind eingetragen, sowie die Abhängigkeiten zwischen Modulen und Plattformen.

1.2 Das Feature 1 wird in der Instanz installiert. Bei der Installation des mbox-Features wird über eine Überprüfungsroutine ermittelt, welche Module auf welchen Plattformen installiert werden müssen. Dann wird die Installation durchgeführt.
Erwartungen:

- Nach der Installation sind auf der Plattform P1 die Module M1 und M2 installiert. Modul 3 ist auf Plattform P3 installiert.

1.3 Das Feature 1 von Instanz deinstallieren bedeutet, dass die Module des Features von den Plattformen entfernt werden.
Erwartungen:

- Alle Module des Features sind von den Plattformen entfernt worden und somit ist das Feature nicht mehr in der Instanz vorhanden.

Test 2 (Basistest II)

Voraussetzungen:

Feature 1 ist auf der Instanz installiert.

Beschreibung:

Das Package 2(PG2) besteht aus drei Modulen(M3, M4, M5) und einer Deskriptor-Datei. Modul M3 ist Plattform P3 zugeordnet. Der Plattform P2 sind die Module M4 und M5 zugeordnet. Die Module gehören zu dem Feature 2.

Mögliche Konflikte:

- Das Feature 2 kann nicht installiert werden, weil die erforderlichen Plattformen fehlen.
- Die Deskriptor-Datei fehlt.
- Modul 3 wird doppelt installiert.

Testfälle:

2.1 Das Package 2(PG2) wird in das Repository geladen.
Erwartungen:

- Alle Module sind ausgepackt und abgelegt. Die Abhängigkeiten zwischen den Modulen und den Features sind eingetragen, sowie die Abhängigkeiten zwischen Modulen und Plattformen.

2.2 Das Feature 2 wird in der Instanz installiert. Bei der Installation des Features wird über eine Überprüfungsroutine ermittelt, welche Module auf welchen Plattformen installiert werden müssen. Dann wird die Installation durchgeführt. In diesem Fall ist das Modul M3 sowohl in Feature 2 als auch in Feature 1 enthalten.
Erwartungen:

- Nach der Installation soll auf der Instanz das Feature 2 installiert sein. Das Modul M3, welches aufgrund von Feature 1 schon installiert ist, wird nicht erneut installiert.

2.3 Das Feature 2 von Instanz deinstallieren bedeutet, dass die Module des Features von den Plattformen entfernt werden. Das Modul M3, welches sowohl in Feature 2 als auch in Feature 1 enthalten ist, wird von der Deinstallation ausgenommen.
Erwartungen:

- Alle Module des Features sind von den Plattformen entfernt worden und somit nicht mehr in der Instanz vorhanden. Außer dem Modul M3, welches noch von Feature 1 benötigt wird. Dieses bleibt weiterhin in der Plattform installiert.

Test 3 (Updatetest I)

Voraussetzungen:

Die Features 1 und 2 sind installiert.

Beschreibung:

Das Package 3 besteht aus einem Modul M6 und einer Deskriptor-Datei. Dies Modul entspricht einem Update zu dem Modul M2 aus Feature 1 und dem Modul M5 aus Feature 2. M6 ist Plattform P1 und P3 zugeordnet.

Mögliche Konflikte:

- Alte Module werden nicht deinstalliert.
- Deskriptor-Datei fehlt.

Testfälle:

3.1 Das Package 3 wird in das Repository geladen.
Erwartungen:

- Das Modul wurde ausgepackt und abgelegt. Die Abhängigkeiten zwischen dem Modul und den Features sind eingetragen, sowie die Abhängigkeiten zwischen dem Modul und der Plattform. Die Überprüfungsroutine ermittelt, dass M2 und M5 deinstalliert werden müssen und dafür M6 als Update installiert wird.

Test 4 (Updatetest II)

Voraussetzungen:

Testfall 3.1

Beschreibung:

Das Package 4 besteht aus drei Modulen (M2, M8, M9) und einer Deskriptor-Datei. Diese Module zusammen bilden das Feature 3. Modul M2 ist P1 zugeordnet. Die Module M8 und M9 sind Plattform P2 zugeordnet.

Mögliche Konflikte:

- Alte Module werden wieder installiert.
- Deskriptor-Datei fehlt.

Testfälle:

4.1 Das Package 4 wird in das Repository geladen.
Erwartungen:

- Die Module wurden ausgepackt und abgelegt. Die Abhängigkeiten zwischen dem Modul und dem Feature sind eingetragen, sowie die Abhängigkeiten zwischen den Modulen und den Plattformen.

4.2 Das Feature 3 wird in der Instanz installiert.
Erwartungen:

- Die Überprüfungsroutine ermittelt, dass M2 von M6 aktualisiert wurde und wird demnach nicht wieder installiert. Die anderen Module M8 und M9 werden in den Plattformen installiert.

4.3 Das Feature wird deinstalliert.
Erwartungen:

- Alle Module des Features 3 werden aus den Plattformen der Instanz entfernt. Es wird nicht versucht, M2 zu deinstallieren, denn M6 ist das Modul anstelle von M2. Aber auch das wird nicht deinstalliert, denn es wird noch von Feature 1 und 2 benötigt.

Test 5 (Abhängigkeitstest I)

Voraussetzungen:

Feature 1 ist installiert.

Beschreibung:

Das Package 5 besteht aus drei Modulen (M10, M11, M12) und einer Deskriptor-Datei. Das Modul 11 wird von Modul 10 benötigt und das Modul 12 wiederum von Modul 11. Das Modul 10 gehört zu Feature 1 und ist Plattform P3 zugeordnet sowie auch die Module M11 und M12.

Mögliche Konflikte:

- Modul 10 wird nicht installiert.
- Module 11 und 12 werden nicht installiert.
- Deskriptor-Datei fehlt.

Testfälle:

5.1 Das Package 5 wird in das Repository geladen.
Erwartungen:

- Die Module wurden ausgepackt und abgelegt. Die Abhängigkeiten zwischen den Modulen und den Features sind eingetragen, sowie die Abhängigkeiten zwischen den Modulen und den Plattformen. Die Überprüfungsroutine ermittelt Modul 10 als neues Modul für Feature 1 und installiert dieses auf der zugehörigen Plattform.

- Außerdem wird erkannt, dass die Module 11 und 12 noch benötigt werden und sie werden ebenfalls installiert.

Test 6 (Abhängigkeitstest II)

Voraussetzungen:

Feature 2 ist installiert und das Update von dem Modul 2 (M6).

Beschreibung:

Das Package 6 enthält ein Modul 15, das nun von Feature 2 benötigt wird. Außerdem existiert die Einschränkung, dass Modul 15 nicht zusammen mit Modul 2 installiert sein darf bzw. mit Updates von Modul 2.

Mögliche Konflikte:

- Modul 15 wird installiert.

Testfälle:

6.1 Das Package 6 wird in das Repository geladen.
Erwartungen:

- Modul 15 darf nicht installiert werden, da Modul 2 bzw. das Update (M6) in Feature 2 enthalten ist.

B Anmerkungen zu Wikipedia

Ich möchte begründen, warum ich bei meinen Recherchen manchmal auf die Webseite von Wikipedia geschaut habe und manche Textzeile in dieser Arbeit genutzt oder zitiert habe.
Momentan geht Wikipedia immer öfter durch die Nachrichten und dabei wird meist ein schlechter Eindruck der freien Enzyklopädie vermittelt. Verständlich, denn jeder kann jederzeit einen Artikel verändern und natürlich dadurch auch leider Falschinformationen verbreiten. Dies ist sogar soweit gegangen, dass US-Kongressangestellte ihre Biographien verschönerten und politische Gegner diffamierten und beleidigten [Quelle: spiegel].
Nichtsdestotrotz bin ich der Meinung, dass gerade im technischen-naturwissenschaftlichen Bereich Wikipedia einen enormen Vorteil gegenüber kommerziellen Online-Enzyklopädien bzw. Lexika hat. Denn die schnelle Entwicklung auf diesem Gebiet lässt die Aktualität von solchen Nachschlagewerken meist schlecht dastehen. Gerade jedem die Möglichkeit zu bieten, einen Artikel zu schreiben und/oder zu ergänzen, ermöglicht die Aktualität, wie sie bei Wikipedia anzutreffen ist.
Dennoch bin auch ich der Meinung, dass man sich nicht auf alles verlassen sollte, was dort geschrieben steht. Um mich zu vergewissern, befasse ich mich deshalb mit anderen Webseiten, die sich dem Thema widmen. Meistens bietet gerade Wikipedia selber eine Liste mit Links, die einem auf die entsprechenden „professionellen“ Seiten führt.
Wenn ich also in dieser Arbeit Zitate oder Textpassagen aus der freien Enzyklopädie benutzt habe, so bin ich sicher gegangen, dass diese auch der Wahrheit entsprechen oder zumindest nicht falsch sind.

Abbildungsverzeichnis

Literaturverzeichnis

Alur u. a. 2002

ALUR, Deepak ; CRUPI, John ; MALKS, Dan: *core J2EE Patterns*. Autorisierte Übersetzung der amerikanischen Originalausgabe: Core J2EE Patterns. Markt+Technik Verlag, 2002. – Übersetzung: Frank Langenau

bio

CENTRAL, BioMed (Hrsg.): *Three tier Java 2 Enterprise Edition software architecture*. http://www.biomedcentral.com/1471-2105/6/101/figure/F1, Abruf: 17. Feb. 2006 19:33

Ihns java

IHNS, Oliver: Mit EJB 3.0 zurück zu POJO/POJI. In: *Javamagazin* 03 (javama05), März, S. 60–62

jtds

http://jtds.sourceforge.net

Monson-Haefel 2001

MONSON-HAEFEL, Richard: *Enterprise JavaBeans*. Deutsche Ausgabe der 2. Auflage. O'Reilly Verlag GmbH & Co. KG, 2001. – Übersetzung: Matthias Kalle Dalheimer

Monson-Haefel 2002

MONSON-HAEFEL, Richard: *Enterprise JavaBeans*. Deutsche Ausgabe der 3. Auflage. O'Reilly Verlag GmbH & Co. KG, 2002. – Übersetzung: Matthias Kalle Dalheimer

postgressql

http://www.postgresql.org/

spiegel

SPIEGEL-ONLINE (Hrsg.): *US-Kongressangestellte manipulierten Wikipedia*. http://www.spiegel.de/netzwelt/politik/0,1518,398357,00.html, Abruf: 16. Feb. 2006 15:43

sunj2ee

SUN MICROSYSTEMS, Inc. (Hrsg.): *J2EE FAQ*. http://java.sun.com/j2ee/faq.html, Abruf: 9. Feb. 2006 15:41. – FAQ

sunjms

SUN MICROSYSTEMS, Inc. (Hrsg.): *Java Message Service (JMS)*. http://java.sun.com/products/jms/, Abruf: 22. Feb. 2006 15:39

sunmdb

SUN MICROSYSTEMS, Inc. (Hrsg.): *What Is a Message-Driven Bean?* http://java.sun.com/j2ee/1.4/docs/tutorial/doc/EJBConcepts5.html, Abruf: 22. Feb. 2006 15:39

wikiejb

WIKIPEDIA (Hrsg.): *Enterprise Java Beans*. http://de.wikipedia.org/wiki/Enterprise_Java_Beans, Abruf: 10. Feb. 2006 12:24

wikigra

WIKIPEDIA (Hrsg.): *Wege, Pfade, Zyklen und Kreise in Graphen*. http://de.wikipedia.org/wiki/Wege%2C_Pfade%2C_Zyklen_und_Kreise_in_Graphen, Abruf: 20. Jan. 2006 11:24. – Graphentheorie

wikipost

WIKIPEDIA (Hrsg.): *PostgreSQL*. http://de.wikipedia.org/wiki/PostgreSQL, Abruf: 18. Feb. 2006 12:13. – PostgreSQL

Name: Erdmann
Vorname: Fokko
Matr.-Nr.: 1153462
Studiengang: Ingenieur-Informatik

An den Prüfungsausschuss
des Fachbereichs Automatisierungstechnik
der Universität Lüneburg
Volgershall 1
21339 Lüneburg

Erklärung zur Diplomarbeit

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Lüneburg, den 27. Februar 2006

Unterschrift