Seminararbeit
Java Naming Directory Interface
(JNDI)

Christoph Malinowski, Felix Mayer
Dozent: Prof.Dr.Rudolf Berrendorf
Fachhochschule Bonn-Rhein-Sieg
Fachbereich Angewandte Informatik
Verteilte und Parallele Systeme

03.06.2003

Java™ 2 Platform Standard Edition v 1.4

In dieser Seminararbeit wird das Java Naming Directory Interface (JNDI) von Sun Microsystems behandelt. Hierbei handelt es sich um eine Standard Erweiterung (*standard extension*) der J2SE v1.4, welche die Nutzung von Namens- und Verzeichnisdiensten ermöglicht. Es werden die im JNDI enthaltenen Pakete analysiert und die jeweils wichtigsten Klassen und Interfaces erarbeitet.
Zum Verständnis dieser Arbeit werden Kenntnisse der Programmiersprache Java vorausgesetzt.

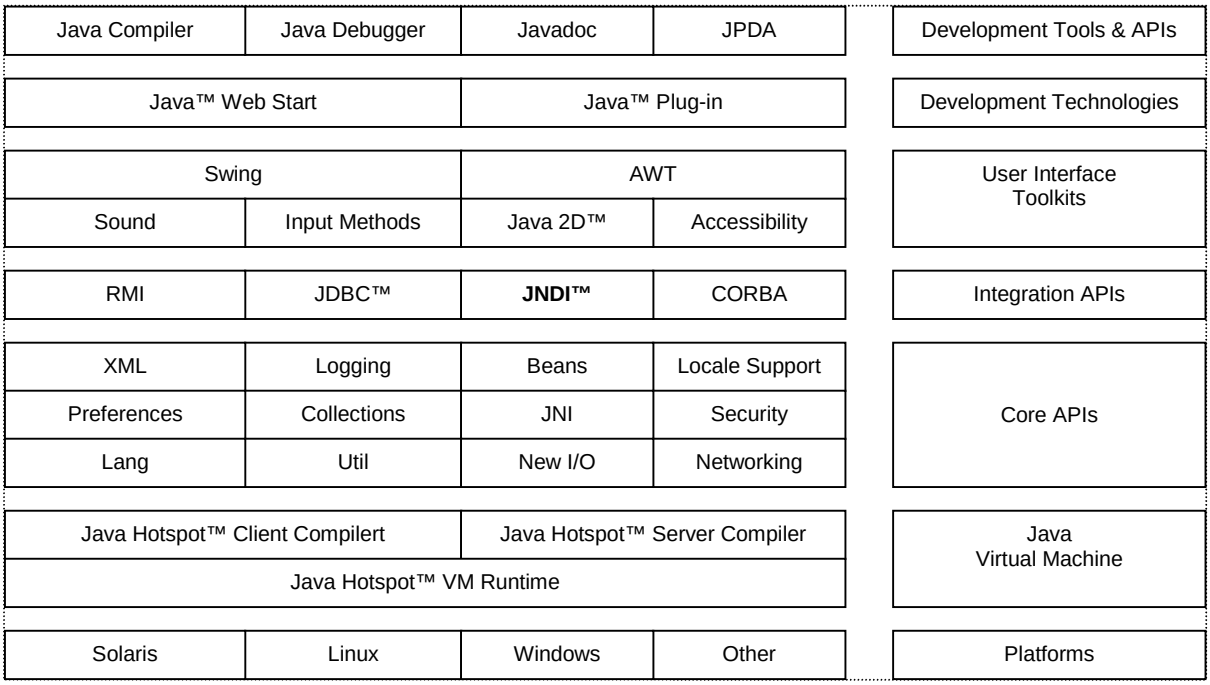

Abbildung 1.1 Java™ 2 Platform Standard Edition v 1.4

Inhaltsverzeichnis

1 Einleitung (F.Mayer)

Java ist eine objektorientierte Programmiersprache, die es ermöglicht, plattformunabhängig, also unabhängig vom Betriebssystem und den Hardwarekomponenten, zu entwickeln. Das Konzept der plattformunabhängigen Programmierung ermöglicht eine vielseitige Kompatibilität zwischen bereits vorhandenen Systemen, die nicht als zueinander kompatibel konzipiert wurden. Ein einfaches Beispiel hierfür sind die verschiedenen Speichermethoden „*little-endian*“ und „*big-endian*“, nach denen Daten hardwareabhängig jeweils von links nach rechts, oder von rechts nach links interpretiert werden. Weiterhin unterscheiden sich verschiedene Systeme in ihrer Organisation von Informationen und Ressourcen durch Namens- und Verzeichnisdienste (*naming and directory services*). Namensdienste ermöglichen dem Anwender die Nutzung von für Menschen verständlichen Namen. Verzeichnisdienste greifen auf diese Namen zurück, um Verzeichnis-Objekte (*directory objects*) zu identifizieren und den Zugriff auf diese zu organisieren.
Um Java-Entwicklern eine transparente Nutzung verschiedener Namens- und Verzeichnisdienste zu ermöglichen, stellt Sun Microsystems die Standarderweiterung Java Naming Directory Interface (JNDI) bereit.
Diese Arbeit soll den Einstieg in den Umgang mit dem JNDI erleichtern. Dafür werden strukturelle Zusammenhänge innerhalb und zwischen den Paketen verstärkt durch Grafiken illustriert, dafür wurde auf Programmierbeispiele verzichtet.

2 Namens- und Verzeichnisdienste (F.Mayer)

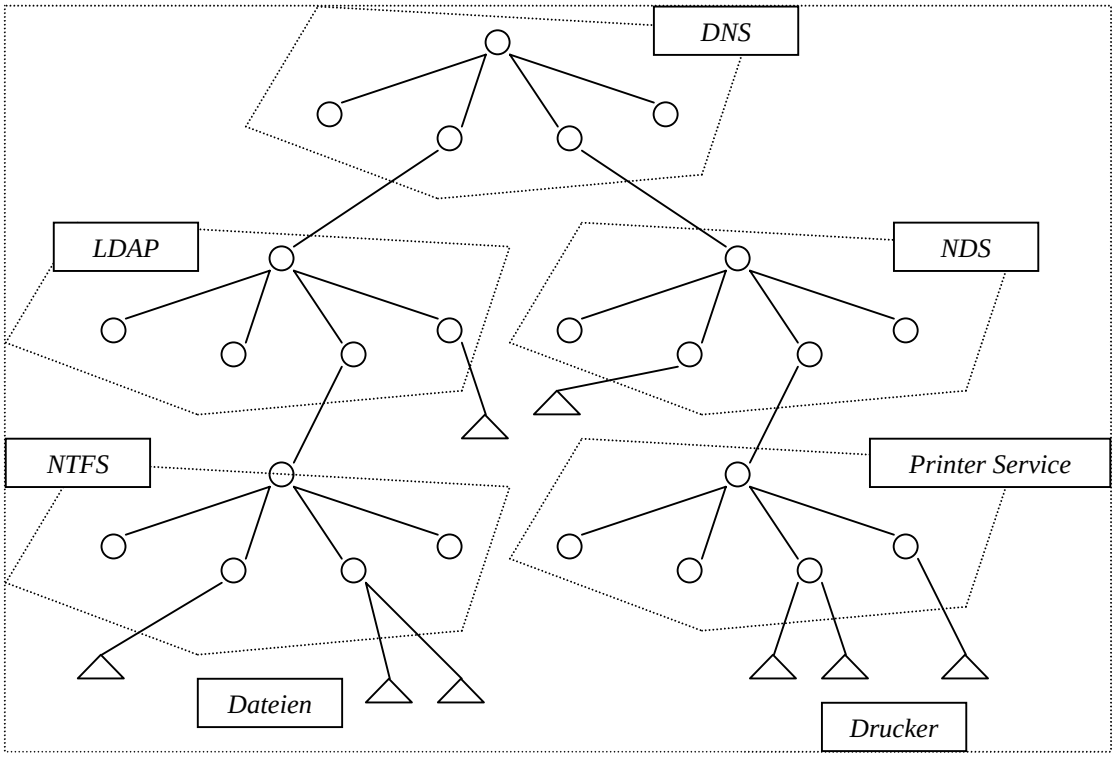

Abbildung 2.1 Namens- und Verzeichnisdienste

2.1 Namensdienste

Ein wichtiger Bestandteil von Computersystemen und Netzwerken sind Namensdienste. Diese kommen meistens in Verbindung mit anderen Diensten zum Einsatz, da sie die Identifikation von Objekten, zum Beispiel einer Datei in einem Dateisystem oder einem Drucker in einem Firmennetzwerk, ermöglichen.

Die wichtigste Funktion eines Namensdienstes ist das Abbilden von Namen auf Objekte jeglicher Art.

Namen, in diesem Sinne, werden nach syntaktischen Regeln (*Namenskonventionen, naming conventions*) generiert. Namenskonventionen definieren unter anderem atomare Namen (*atomar names*) als unteilbare Komponente eines Namens.
Zusammengesetzte Namen (*compound names*) sind Sequenzen von null oder mehreren atomaren Namen, die nach den syntaktischen Regeln einer Namenskonvention generiert werden.
Eine Bindung (*binding*) ist die Assoziation zwischen einem Namen und einem Objekt. Ein Kontext (*context*) ist eine Menge von Bindungen mit jeweils unterschiedlichen atomaren Namen, generiert nach der selben Namenskonvention, und stellt eine Menge von einer oder mehreren Operationen bereit. Die offensichtlich wichtigste Operation ist die Identifikation des jeweils an einen Namen gebundenen Objekts, weitere Implementierungen sind möglich.

Atomare Namen in einem Kontext können anstatt an ein Objekt auch an einen Kontext (*Subkontext, sub context*) desselben Typs gebunden werden. Hierdurch wird eine hierarchische Organisation der Objekte ermöglicht. Weiterhin erschließt sich in diesem Zusammenhang die Bedeutung und Verwertung von zusammengesetzten Namen, da deren Auflösung beginnend mit dem Kontext auf der obersten Hierarchiestufe (*initial context*) sukzessiv durch Auswertung der an die atomaren Namen gebundenen Subkontexte geschieht.
Ein Namenssystem (*naming system*) identifiziert eine Menge von zusammenhängenden Kontexten mit gleichen Namenskonventionen und gleichen Operationen.
Ein Namensraum (*name space*) beinhaltet alle Namen eines Namenssystems.
Gemischte Namen (*composite names*) spannen mehrere Namensräume auf.
Ein Beispiel für einen gemischten Namen ist folgende URL:

www.fh-bonn-rhein-sieg.de/index.htm

Hier ist der zusammengesetzte Name www.fh-bonn-rhein-sieg.de ein Element des Domain Naming System (*DNS*) und identifiziert die Maschine, über die der Webserver der Fachhochschule Bonn-Rhein-Sieg im World Wide Web adressiert wird. Der atomare Name index.htm repräsentiert ein Element des logischen Dateisystems, das auf der Maschine existiert. Über den im Dateisystem integrierten Namensdienst wird das an den Dateinamen index.htm gebundene Objekt, die korrespondierende HTML-Datei, identifiziert.
Um die Nutzung von gemischten Namen zu unterstützen, wird im JNDI eine spezielle Syntax definiert, und es werden verschiedene Werkzeuge zur Verarbeitung von gemischten Namen bereitgestellt.

2.2 Verzeichnisdienste

Verzeichnisdienste ermöglichenden den Zugang zu verschiedensten Informationen über Nutzer und Ressourcen in einer Netzwerkumgebung. Dazu benutzt ein Verzeichnisdienst ein Namenssystem, durch das er Verzeichnis-Objekte identifizieren kann.
Ein Verzeichnis-Objekt ist ein spezieller Objekttyp, der eine Vielzahl an Informationen über die Netzwerkumgebung beinhalteten kann. Verzeichnis-Objekte können Attribute, bestehend aus einem *identifier* und einer Wertemenge, zugewiesen bekommen. Weiterhin stellen Verzeichnis-Objekte Operationen zur Verfügung, um jeweils assoziierte Attribute zu generieren, hinzuzufügen, zu entfernen und zu modifizieren.

2.3 Events (C. Malinowski)

Zur Administration oder Überwachung von Namens- und Verzeichnisdienste, existieren Administrations- und Monitoring- Tools. Um Änderungen in Namens- oder Verzeichnisdiensten zu überwachen, benötigt man ein Benachrichtigungsmodell (event notification).
Ein solches Modell ermöglicht es Anwendungen bestimmte Bereiche des Verzeichnisdienstes auf Veränderungen zu überwachen, ohne das ständig der gesamte Namensraum neu eingelesen und mit dem bisherigen Stand verglichen werden muss.

3 Java Naming Directory Interface

3.1 JNDI-Architektur (F.Mayer)

Das Java Naming Directory Interface Application Programming Interface (JNDI API) stellt Mechanismen für Java-Applikationen bereit, welche die transparente Nutzung von Namens- und Verzeichnisdiensten ermöglichen.
Das Java Naming Directory Interface Service Provider Interface (JNDI SPI) ermöglicht ein dynamisches Plugin verschiedener Namens- und Verzeichnisdienste unter dem JNDI API.

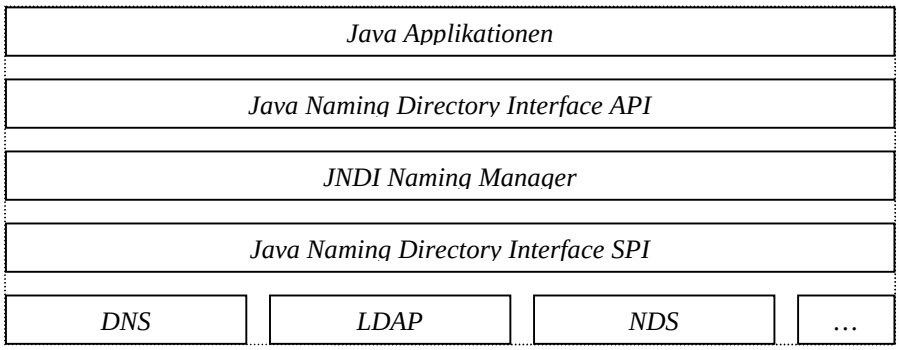

Abbildung 3.1 JNDI-Architektur

3.2 Paketübersicht (F.Mayer)

Das JNDI API besteht aus folgenden Paketen:

javax.naming beinhaltet Klassen und Interfaces, um auf Namensdienste zuzugreifen.

javax.naming.directory ermöglicht den Zugang zu Verzeichnissen

javax.naming.event stellt Klassen und Interfaces zur Interaktion mit Namens- und Verzeichnisdiensten bereit

javax.naming.ldap unterstützt LDAP (Lightweight Directory Access Protocol) v3

Das JNDI SPI besteht aus folgenden Paketen:

javax.naming.spi ermöglicht dynamische Plugins verschiedener naming and directory service provider.

3.3 Prinzipielle Nutzung des Java Naming Directory Interface

3.3.1 javax.naming (F.Mayer)

Dieses Paket stellt die Basisklassen- und Interfaces des JNDI bereit, welche den Zugang zu Namensdiensten ermöglichen. Es beinhaltet die Interfaces *Context, Name, NameParser, NamingEnumeration, Referenceable* sowie die Klassen *CompositeName, CompoundName, InitialContext, NameClassPair, Reference und die abstrakte Klasse RefAddr*, welche Erweiterungen der Klasse Object sind. Weiterhin enthält das Paket Erweiterungsklassen. Die Struktur des Pakets wird durch folgende Abbildung deutlich:

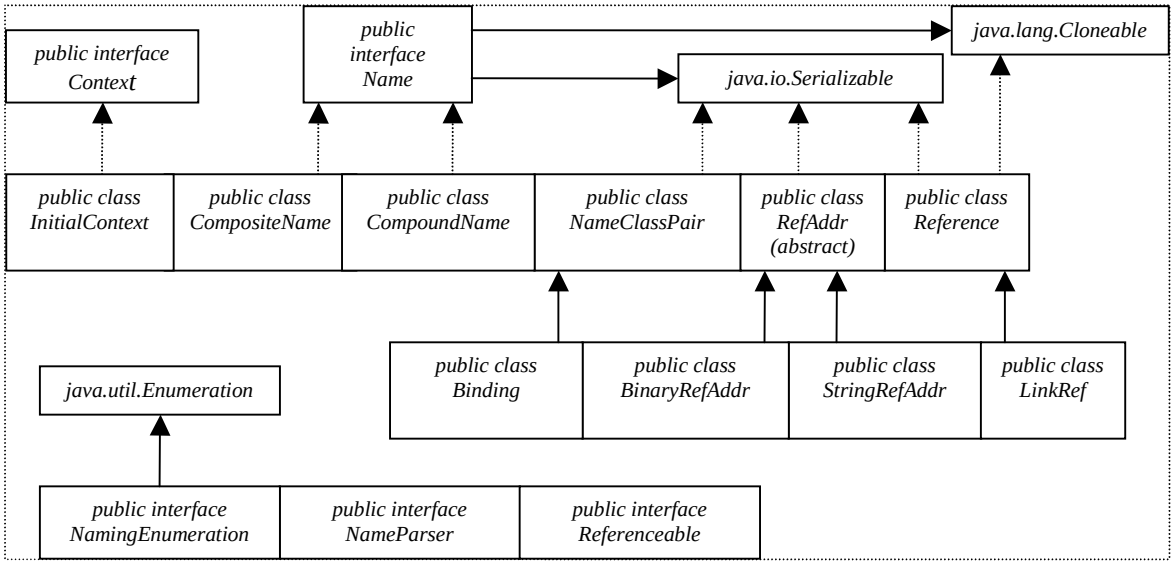

Abbildung 3.2 Paketstruktur javax.naming

Das Interface *Name* repräsentiert einen generischen Namen in Form einer geordneten Sequenz von Komponenten. Es wird von den Klassen *CompoundName* und *CompositeName* implementiert. Instanzen der Klasse *CompoundName* sind Namen eines hierarchischen Namensraums, deren Komponenten atomare Namen sind. Instanzen der Klasse *CompositeName* sind Sequenzen von String Namen, deren Komponenten Elemente hierarchischer Namensräume repräsentieren.
Bei dem Interface *Context* handelt es sich um das zentrale Interface des Pakets javax.naming, welches von der Klasse *InitialContext* implementiert wird. Hier werden Basisoperationen zu Analyse und Manipulation von Bindungen zwischen Namen (*zusammengesetzt/compound, gemischt/composite*) und Objekten definiert. Die Auflösung der Namen geschieht implizit bei der Parameterübergabe. Methoden werden somit auf dem über den Namen identifizierten Kontextobjekt ausgeführt. Wird der leere Name übergeben, so wird die Methode auf dem aufrufenden Kontextobjekt ausgeführt. Weiterhin existiert zu den meisten Methoden, die einen Namen als Parameter erwarten, eine Methode, die einen String als Übergabeparameter erwartet. String Namen repräsentieren in diesem Fall gemischte Namen. Applikationen, die lediglich Leseoperationen über Kontexten ausführen, um verschiedene Objekte zu identifizieren, arbeiten effizienter, wenn sie nur String Namen verwenden.

Eine weitere wichtige Klasse ist die Klasse *Bindings*, eine Erweiterung der Klasse *NameClassPair*. Während Instanzen von *NameClassPair* nur Verbindungen zwischen Namen und Objektklassennamen in Form von Strings repräsentieren und auch keinen synchronisierten Zugriff gestatten, ermöglichen Instanzen der Klasse *Bindings* synchronisierten Zugriff auf gebundene Objekte. Operationen auf Instanzen der Klasse *Bindings* sind daher aufwendiger in ihrer Ausführung.
Um JNDI-Clients die Illusion zu verschaffen, Objekte aus Verzeichnisdiensten ohne Objektunterstützung, wie zum Beispiel X.500, in Namens- und Verzeichnisdienste einbinden zu können, gibt es die Klasse *References*. Wird nach einem Methodenaufruf eine Instanz der Klasse *References* zurückgegeben, so versucht JNDI diese zunächst in ein Objekt zu konvertieren. Daher müssen Klassen, deren Objekte durch Instanzen der Klasse *References* repräsentiert werden können, das Interface *Referenceable* implementieren.
Namenssysteme können durch Referenzen auf Kontexte in anderen Namenssystemen verbunden werden. Das Ergebnis sind gemischte Namensräume (*composite namespaces*).
Dazu verwalten Instanzen der Klasse *Reference* eine Liste über Adressen und Informationen, über die das jeweils referenzierte Objekt identifiziert werden kann.
Die abstrakte Klasse *RefAddr* definiert Methoden

3.3.2 javax.naming.directory (F.Mayer)

Dieses Paket stellt Klassen und Interfaces des JNDI bereit, welche den Zugang zu Verzeichnisdiensten ermöglichen. Es beinhaltet die Interfaces *Attribute, Attributes* und *DirContext* sowie die Klassen *BasicAttribute, BasicAttributes, ModificationItem, SearchControls*, die Erweiterungen der Klasse *Object* sind. Weiterhin enthält das Paket Erweiterungsklassen für das Paket javax.naming. Die Struktur des Pakets javax.naming.directory wird durch folgende Abbildung deutlich:

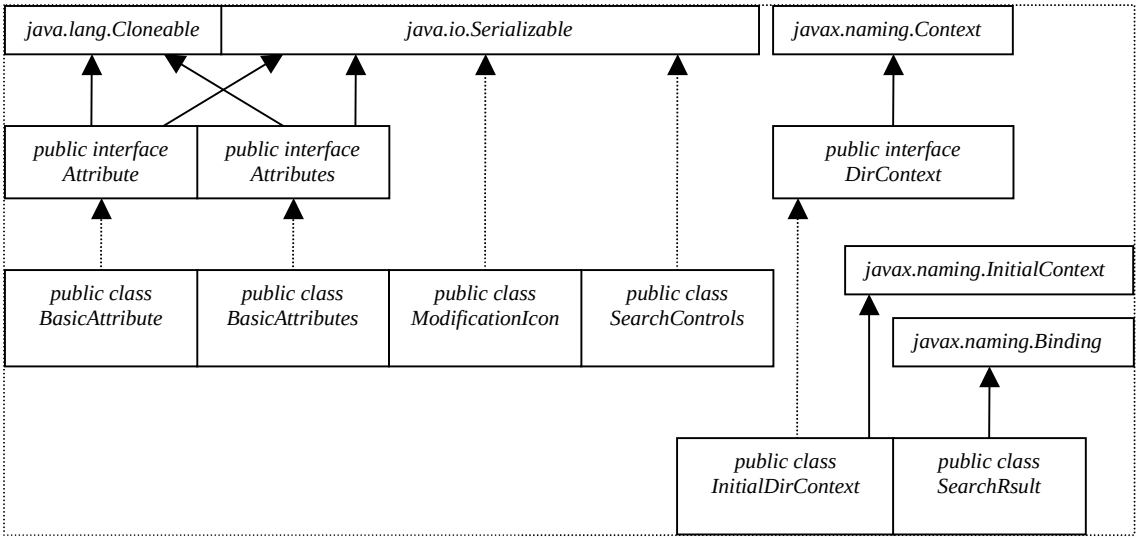

Abbildung 3.3 Paketstruktur javax.naming.directory

In den Interfaces *Attribute* und *Attributes* werden Grundfunktionalitäten für die Verwaltung von Attributen definiert. Sie werden von den Klassen *BasicAttribute* und *BasicAttributes* implementiert. Attribute werden über einen String *identifier* identifiziert und können null oder mehrere Objekte als Werte zugewiesen bekommen. Die Werte können geordnet oder ungeordnet sein. Attribute mit geordneten Werten können dupliziert werden.
Das Interface *DirContext* ist das zentrale Interface in diesem Paket. Es erweitert das Interface *Context* im Paket javax.naming um Methoden, mit denen Attribute, Assoziationen zwischen String Namen und Objekten, generiert und manipuliert werden können. Die Klasse *InitialDirContext* implementiert das Interface *DirContext*. Modifikationen an Attributen können nur über einen Verzeichnis-Kontext direkt, oder mittels einer Instanz der Klasse *ModificationIcon* in Form eines Codes vorgenommen werden. Da *DirContext* das Interface *Context* im Paket javax.naming erweitert, können hybride Operationen über Namens- und Verzeichnissystemen implementiert werden. Weiterhin definiert das Interface Suchfunktionen, die verschiedene Informationen über den aufrufenden Kontext liefern.
Suchfunktionen werden durch die Klasse SearchControls unterstützt. Methoden dieser Klasse erlauben zum Beispiel das Setzen von Zeitschranken für die Dauer einer Suchoperation in einem Verzeichnissystem.
Um Suchergebnisse effizient behandeln und auswerten zu können, implementiert javax.naming.directory die Klasse *SearchResults*.
Objekte der Klassen *ModificationIcon*, *SearchControls* und *SearchResults* sind nicht für die Nutzung durch Threads synchronisiert.

3.3.3 javax.naming.event (C.Malinowski)

Dieses Paket stellt Klassen und Interfaces des JNDI bereit, welche die Interaktion mit Namens- und Verzeichnisdiensten ermöglichen. Es beinhaltet die Interfaces *EventContext, EventDirContext, NamespaceChangeListener, NamingListener* und *ObjectChangeListener* sowie die Klassen *NamingEvent* und *NamingExceptionEvent*, die Erweiterungen der Klasse *EventObject* sind. Die Struktur des Pakets javax.naming.event wird durch folgende Abbildung deutlich:

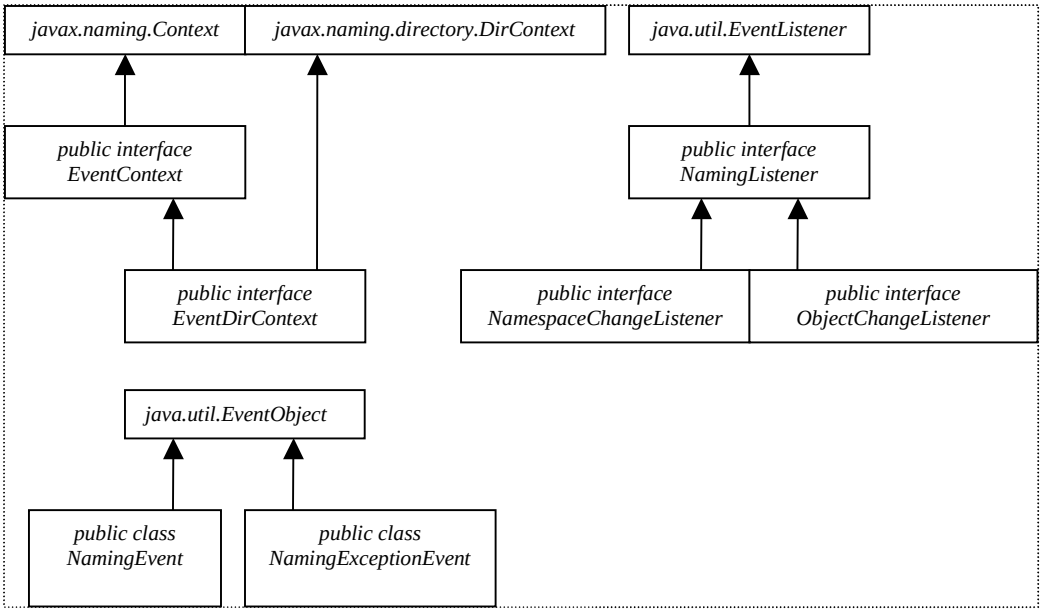

Abbildung 3.4 Paketstruktur javax.naming.event

Dieses Paket beinhaltet Unterstützung für Events. *Event notification* bedeutet, dass eine beliebige, diese API Verwendende Applikation in der Lage ist ihr Interesse an bestimmten Änderungen im zu überwachenden Namensraum zu bekunden. Es werden dann Methoden definiert, die beim Eintreten solcher Ereignisse ausgeführt werden sollen.

1. Ereignisse, die den Namensraum betreffen, dazu zählen
   a) Hinzufügen von Objekten
   b) Entfernen von Objekten
   c) Umbenennen von Objekten

2. Ereignisse, die den Inhalt der Objekte ändern.

Es wäre also zum Beispiel möglich zu überprüfen beim Löschen von Objekten eine Methode zu starten, die zusätzliche Daten löscht, oder die ein Backup des Objektes sichert.
Um dies zu erreichen, definiert dieses Paket eine Klasse *NamingEvent*, die von einem Namens- oder Verzeichnisdienst geworfen wird. Es werden zusätzlich Subinterfaces von *Context* und *DirContext* definiert, um Applikationen zu gestatten, Events, die von einem Kontext geworfen werden, abzufragen. Diese Subinterfaces heißen EventContext und EventDirContext.

Das zentrale Interface EventContext beeinhaltet die Methoden für das registrieren und deregistrieren von Listenern auf Objektänderungen. Die *addNamingListener*() Methode akzeptiert dabei als „Beschreibung“ ihrer Zielobjekte (oder des Zielobjekts) eine Kombination aus einem *Target* und einem *Scope*. Das *Target* bezeichnet den Namen eines Objektes und die konstanten für *Scope* reichen von „nur das einzelne Objekt“, bis z.B. allen *events*, die ein Objekt oder ein hierarchisch untergeordnetes Objekt betreffen, oder alle mit dem *Target* in Verbindung stehenden Objekte, das *Target* jedoch nicht.
Denkbar wäre also zum Beispiel, wenn in einem Namensraum verschiedene Objekte einem Objekt zugeordnet sind, z.B. Teilnehmer eines Kurses, oder Mitarbeiter in einer bestimmten Abteilung, nur die Kurse oder Abteilungen anzuwählen oder z.B. alle Mitarbeiter der Abteilung „Beschaffung“, ohne jedoch die Abteilung selbst mit auszuwählen.
Die Funktionen, die *EventDirContext* bereitstellt sind im Prinzip äquivalent, beziehen sich jedoch auf eine Verzeichnisstruktur. Dadurch werden statt einem *Target* und einem *Scope* *Filter* und *SearchControls* übergeben.
Die hier unterstützten Filter sind definiert in der RFC 2254, die Filter spezifiziert und im Zusammenhang mit LDAP verabschiedet wurde.
NamingListener ist das *SuperInterface* für *NamespaceChangeListener* und *ObjectChangeListener*. *NamingListener* selbst wird dabei aber (so gut wie) nie benutzt. Stattdessen wird in der Regel eines der abgeleiteten Interfaces *NamespaceChangeListener* oder *ObjectChangeListener* benutzt. *NamingListener* selbst besitzt nur eine einzige Methode *namingExceptionThrown*(), die implementiert sein muss. Sie erlaubt dem *Listener* Zugriff auf E*xceptions*, die während er Informationen sammelt, vom *service provider* geworfen werden. Wenn eine solche *Exception* geworfen wurde, ist der *Listener* automatisch vom *EventContext* für den er registriert war getrennt worden. *NamespaceChangeListener* kümmert sich um die Änderungen im Namensraum, also hinzufügen, umbenennen und entfernen von Objekten, während *ObjectChangeListener* überprüft, ob der Inhalt selbst geändert wird.
Die beiden Klassen des Paketes sind *NamingEvent* und *NamingExceptionEvent*. *NamingEvent* repräsentiert die Events. *NamingEvent* enthält Parameter und Methoden, um Informationen über das aufgetretene Event abzufragen. Es lässt sich abfragen in welchem Kontext ein Event aufgetreten ist, von welchem Typ der Event ist und Informationen über das Objekt vor und nach der Änderung. Dies ist vor allem hilfreich, wenn man bestimmte Objekte vor Änderungen bewahren, oder bestimmte Änderungen zurücksetzen möchte.
Die Events der Klasse *NamingExceptionEvent* werden geworfen, falls ein Verzeichnisdienstserver die Verbindung nach Erstellen des *Listeners* beendet und so eine *NamingException* auftritt.

3.3.4 javax.naming.ldap (C.Malinowski)

Dieses Paket stellt Klassen und Interfaces des JNDI bereit, welche die Nutzung von LDAP v3-speziefischen Optionen ermöglichen, die von dem Paket javax.naming.directory nicht unterstützt werden. Die Entwickler von SUN empfehlen dieses Paket nur dann zu verwenden, wenn wirklich Dinge benötigt werden, die im generischeren java.naming.directory nicht sowieso abgedeckt sind. Die Struktur des Pakets javax.naming.ldap wird durch folgende Abbildung deutlich:

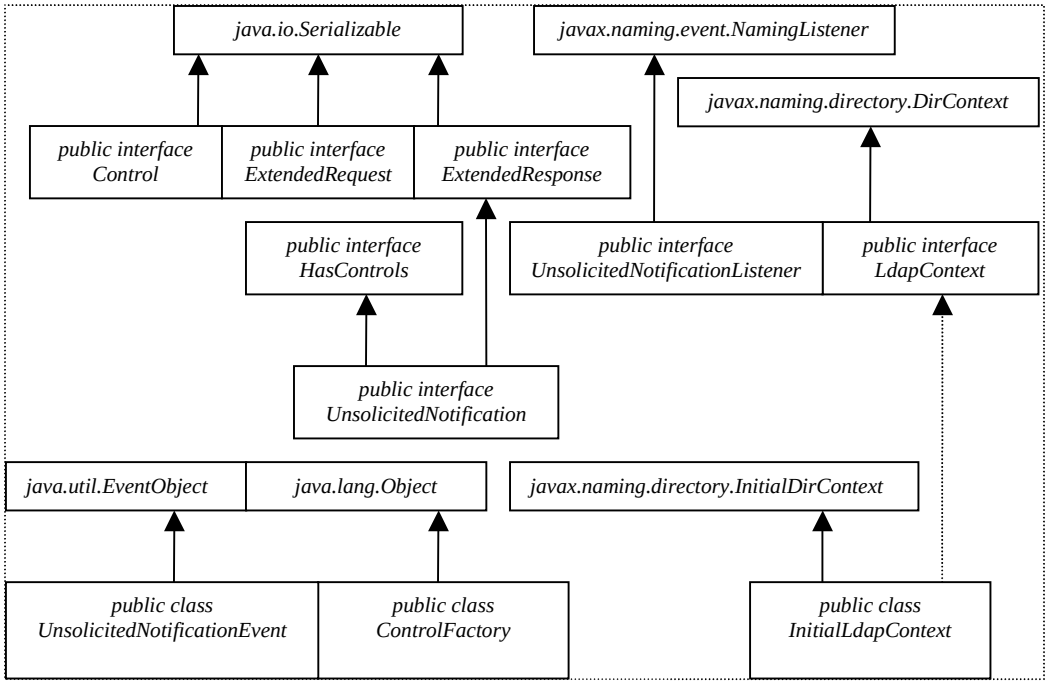

Abbildung 3.5 Paketstruktur javax.naming.ldap

Die zentralen Interfaces dieses Pakets sind *LdapContext*, *ExtendedRequest* und *ExtendedResponse*. Die Namen sind dabei relativ intuitiv.
LdapContext definiert den Kontext auf dem wir unsere LDAP Anfragen stellen. LdapContext ist abgeleitet von DirContext und dadurch von Context, die Handhabung ist also immer noch ähnlich zu java.naming(.directory).
Die komplementären Interfaces *ExtendedRequest* und *ExtendedResponse* sind der Kern der eigentlichen komplexen LDAP v3 Abfragen. *Request* ist die Anfrage an einen LDAP Server, während *Response* die antwort des LDAP Servers an einen Client ist.
Wichtig ist dabei: Es kann keine *ExtendedResponse* ohne *ExtendedRequest* geben, da die Response als Antworte auf unser Request generiert werden. Es gibt jedoch auch Requests auf die man keine Response erhält.
Auch hier lassen sich schon beim generischeren javax.naming.directory Controls generieren. Diese sind jedoch nun LDAP-spezifischer. Beispiel für die Anwendung einer Control, wäre ein SortControl, dass auf das Attribut „Alter“, der Objekte im Namensraum angewandt wird und so die Suchergebnisse immer nach dem Alter sortiert zurückgeben wird.

In dieser komplexeren Variante gibt es sowohl RequestControls, die entweder die Verbindung zum Server, oder die Anfrage an den Server beeinflussen, als auch ResponseControls, die die Änderungen an den Controls, oder das Ergebnis beeinflussen.

3.3.5 javax.naming.spi (C.Malinowski)

Dieses Paket stellt Klassen und Interfaces des JNDI bereit, um verschiedene Namens- und Verzeichnisdienste dynamisch an das JNDI API anzupassen und damit Kompatibilität zu Applikationen, die auf der JNDI API aufsetzen zu ermöglichen. Die Struktur des Pakets javax.naming.spi wird durch folgende Abbildung deutlich:

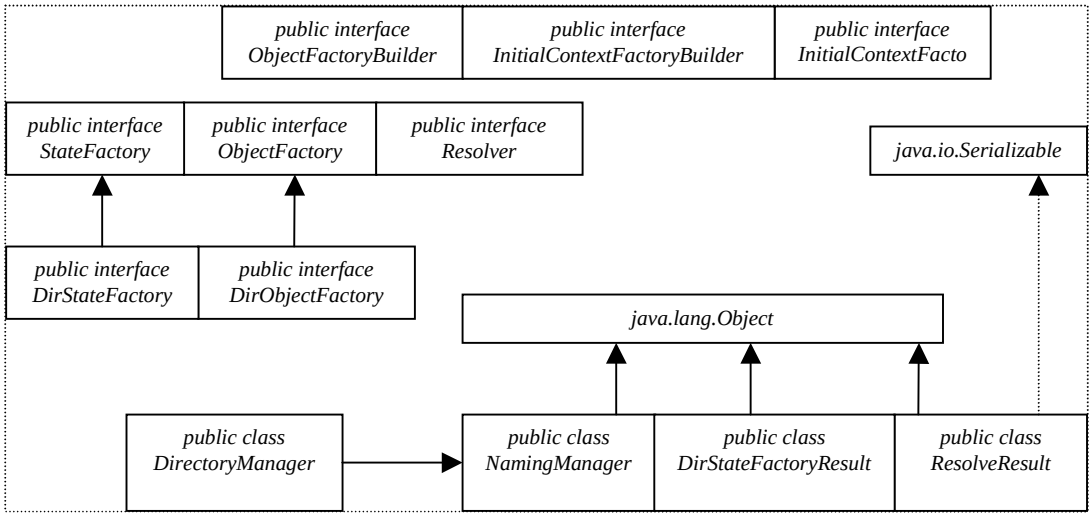

Abbildung 3.6 Paketstruktur javax.naming.spi

Besonders ist hierbei, dass JNDI erlaubt, dass Namen in mehreren Namensräumen existieren und das ein *service provider* eventuell mit mehreren weiteren Providern zusammenarbeiten muss um das *Request* zu erfüllen. Die Methoden, um einen Client Request durch mehrere „Provider – Implementationen“ zu schleusen liegen ebenfalls im SPI Paket (Federation).
Die Klasse beinhaltet daher einige Interfaces für Objekt-„Factories“, um die Objekte im Namensraum in für Java zugänglicher Form zugänglich zu machen. Mithilfe dieser objekt factories können Objekte jederzeit dynamisch geladen werden.
Zum Beispiel wäre es möglich, dass ein Druckerservice, der den Druckservice eigentlich an einen anderen Service weiterleiten müsste dynamisch aus einer Referenz zu dem anderen Drucker ein eigenes Drucker-Objekt erstellt und an den Client liefert, so dass dieser bereits sofort nach seiner Anfrage losdrucken kann, obwohl die Druckanfrage an sich weitergeleitet werden muss.
Genau umgekehrt kann auch bereits beim Binden eines neuen Druckers ein Druckerobjekt im anderen Naming System erzeugt werden.

4 Zusammenfassung (C.Malinowski)

Abschließend lässt sich sagen, dass das JNDI dem Java-Programmierer zum Teil durchaus angenehm mundgerecht die Tools liefert, die er benötigt um in Java Programmen Namensdienste erreichbar zu machen und zueinander kompatible Namensdienste zusammen zu verwenden.
Sehr positiv fällt auf, dass besonders javax.naming und javax.naming.directory sehr generisch gehalten wurden und damit Unterstützung für sehr, sehr viele Namens– und Verzeichnisdienste aus einer Hand bieten können.
Dennoch wird Java auf jedem Fall seinem Anspruch auch für komplexere heterogene Umgebungen ab einer gewissen Schicht einheitliche Routinen zu bieten gerecht, da die eigentliche Verteilung der Namens- und Verzeichnisdienstanfragen auf einer sehr niedrigen Ebene der API geschieht und das ganze dann, wenn man von außerhalb daraufsieht sehr transparent gestaltet ist. Für den Anwendungsentwickler ist es sehr häufig nicht ersichtlich, welcher Dienst ihm nun seine Antworten lieferte.
Es gibt jedoch auch einige wichtige Dinge zu beachten, zum Beispiel das häufig eine Thread-Synchronisation nicht gegeben ist und dadurch parallele Anwendungen nur dann fehlerfrei laufen können, wenn der Applikationsentwickler selbst darauf achtet, dass seine Threads nicht durcheinander kommen. Das kann besonders dann unangenehm werden, wenn man sich entschließt mehrere Verzeichnisdienste nach etwas zu durchsuchen und die Ergebnisse dann im besten Falle ungeordnet eintrudeln.
Um Anwendungen zu entwickeln, die innerhalb der JNDI-API ansetzen ist es jedoch leider nicht zu vermeiden, sich mit einer relativ komplizierten und verzweigten API auseinanderzusetzen, die nicht für jedes Problem eine einfache Lösung bereitstellt, sondern den Entwickler auch häufiger zwingt sich sehr an die API anzupassen.
Das JNDI-SPI ist keineswegs leichter Stoff, den sich ein Entwickler an einem Tag vollständig aneignen kann, ich denke jedoch, das die Einarbeitungszeit dennoch im Handhabbaren Bereich liegen dürfte.

5 Index

6 Literatur- und Abbildungsverzeichnis

6.1 Literatur (C. Malinowski)

www.java.sun.com
Die „Homepage“ von Java

http://java.sun.com/products/jndi
Die „Homepage“ des JNDI Projektes

ftp://ftp.javasoft.com/docs/j2se1.3/jndispi.pdf
das „JNDI SPI Document“ im pdf Format.

http://java.sun.com/products/jndi/tutorial
Das JNDI – Tutorial

ftp://ftp.javasoft.com/docs/j2se1.3/jndi.pdf
Die JNDI – API im pdf Format.

6.2 Abbildungen